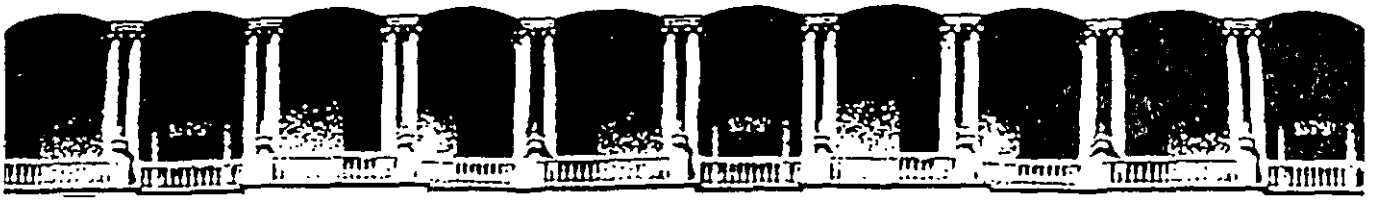


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

APUNTES GENERALES

PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ
PALACIO DE MINERÍA
ABRIL DE 1999

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO Y CONSTRUCCION DE ROBOTS MOVILES

Coordinador Académico:

Dr. JESUS SAVAGE CARMONA

Profesores:

DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCIA
ING. CARLOS MUNIVE VAZQUEZ

MEXICO D.F.

INDICE

Capítulo 1 Objetivo y Generalidades

Capítulo 2 Microprocesadores y Microcontroladores

Capítulo 3 Sensores

Capitulo 6 Diseño del Robot Móvil

Capitulo 7 Control y aplicaciones del robot móvil

Apéndices

Microcontrolador MC68HC11E9

CAPITULO No. 1
Objetivo y Generalidades

1.1 Objetivo

El objetivo de este trabajo es presentar los elementos necesarios para la construcción y programación de un robot móvil, abarcando conceptos importantes de la robótica, los microprocesadores, microcontroladores, la forma de operación de los sensores, los dispositivos electrónicos y su adaptación con el sistema, así como la programación del mismo. Para este caso en particular, se desarrolla en base al MC68HC11, porque contiene varios subsistemas internamente, es de fácil programación; como consecuencia ahorra espacio y cantidad de dispositivos que se tendrían que agregar, por lo tanto reduce su costo sustancialmente (sus características se detallan en el apéndice A).

Se diseñarán interfaces de control y sensado, tanto de fácil construcción como de adaptación a otro tipo de microcontrolador. De esta manera se pretende sirva como guía para que personas interesadas en esta área, tengan la oportunidad de construir o diseñar su propio robot.

1.2 Generalidades

El diseño y la práctica con los robots móviles, tiene desde hace algunos años un gran auge y más adeptos en esta disciplina, logrado por la diversidad de modelos y actividades que pueden realizar. El campo de aplicación se ha extendido en casi todas las áreas tecnológicas; para la comunidad científica es un medio de gran ayuda en los experimentos que se realizan; en el aspecto académico se contemplan en planes de estudio del área físico matemáticas materias relacionadas con la robótica, y en el área comercial es posible adquirir algunos modelos sencillos como entretenimiento.

Todo comenzó con la idea y sueño, de contar con máquinas capaces de reproducir los movimientos de los seres humanos y de los animales, además de servir como medio de enlace entre un medio hostil y el hombre. El desarrollo de esta actividad es de carácter multidisciplinario, compartiéndose en áreas tan diversas y afines, como la automotriz, la mecánica, la electrónica, y la computación. Como consecuencia; una investigación en esta disciplina, exige una estrecha colaboración entre ellas.

La llegada de la electrónica digital y de la microelectrónica, ha logrado la reducción en costos y tamaños de los robots, con una funcionalidad y aplicación, cuyo único limite es la imaginación.

La robótica puede considerarse que hace referencia a una automatización importante de numerosos sectores de la actividad humana, en los cuales se ha estimado imprescindible, hasta hace poco tiempo la presencia del hombre. Los progresos extraordinarios realizados, particularmente en informática, asociados a las necesidades económicas del día en los países industrializados, permiten por un lado, construir robots y por otro verlos implantar en numerosas ramas de actividades.

1.2.1 Los campos de la robótica

El campo de aplicación de la robótica influye profundamente la forma y las propiedades de las máquinas y robots; puede facilitarse la comprensión considerando tres grandes campos de aplicación:

I) El campo de la producción:

Concretamente aquí es donde los industriales han desarrollado el máximo esfuerzo, ya que en la utilización de los robots ven numerosas ventajas; entre ellas la disminución de la mano de obra.

La asociación entre los robots y las máquinas, aportan dos ventajas con relación a los modos de producción tradicionales:

a) La automatización casi integral de la producción, que puede acompañarse de una mejor calidad del producto terminado, de una mayor fiabilidad en el mantenimiento de la calidad, y de una mejor adaptación de la cantidad producida a la demanda.

b) La rapidez de reconfiguración de la unidad de producción cuando se pasa de la fabricación de un producto, a la de un producto similar.

II) El campo de la exploración:

Se trata de un campo diferente, se quieren ejecutar trabajos en un lugar al que el hombre no puede permanecer, porque el medio es de difícil acceso y peligroso; es el caso para:

a) El medio submarino;
b) El medio espacial;
c) El medio irradiado de las centrales nucleares, etc.

La robótica permite ensayar dos tipos de solución para estas intervenciones:

a) El robot autónomo:

Es el que se va a enviar al medio hostil, programándole su misión; por ejemplo: recoger ciertas piedras y estudios en el planeta Marte, o bien inspeccionar las soldaduras de una central nuclear, etc.

b) La teleoperación (llamada también telepresencia):

Consiste en enviar una maquinaria (un robot que se llama maquina esclava), al medio hostil y poder controlarla a distancia desde otro lugar llamado puesto maestro, al mando del cuál se encuentra un hombre, que es el operador.

Es pues el hombre quien efectúa todas las tareas de reflexión y de la activación de los movimientos de la maquinaria esclava, esto es posible realizarlo con un sistema de visión que por lo general son cámaras de vídeo.

III) El campo de la asistencia individual:

El campo donde se desarrolla, es principalmente en la robótica médica, que permite mejorar las condiciones de vida de los paralíticos (parapléjicos y tetrapléjicos) o de los amputados.

1.2.2 Origen de la palabra robot

El término robot parece que empezó a utilizarse hacia los años 1920-1930, a raíz de una obra de teatro del autor checo Karel Tschapek titulada R.U.R. (Rossum´s Universal Robot). Esta obra pone en juego pequeños seres artificiales antropomorfos, que responden perfectamente a las ordenes de su maestro. Estos seres llevan el nombre de robot, del checo *robota (siervo),* idéntico al término ruso y que significa *trabajo*.

Se consideran las siguientes definiciones para un robot:

a) La de Oxford English dictionary:

Un aparato mecánico que se parece y hace el trabajo de un ser humano.

b) La del Robot Institute of America:

Un manipulador reprogramable y multifuncional concebido para transportar materiales, piezas, herramientas o sistemas especializados, con movimientos variados y programados, con la finalidad de ejecutar tareas diversas.

1.2.3 Propiedades características del robot

Más que buscar una definición, se presentan dos propiedades que caracterizan a un robot:

a) La versatilidad; es la potencialidad (posibilidad) estructural (mecánica) de ejecutar tareas diversificadas y/o ejecutar una misma tarea diversificada.

b) La autoadaptabilidad al entorno; un robot debe, por si solo, alcanzar su objetivo (la ejecución de una tarea), a pesar de las perturbaciones imprevistas (pero limitadas) del entorno, a lo largo de la ejecución de la tarea.

Un robot operacional puede representarse por cuatro entidades unidas entre sí como se indica en la figura 1-1.

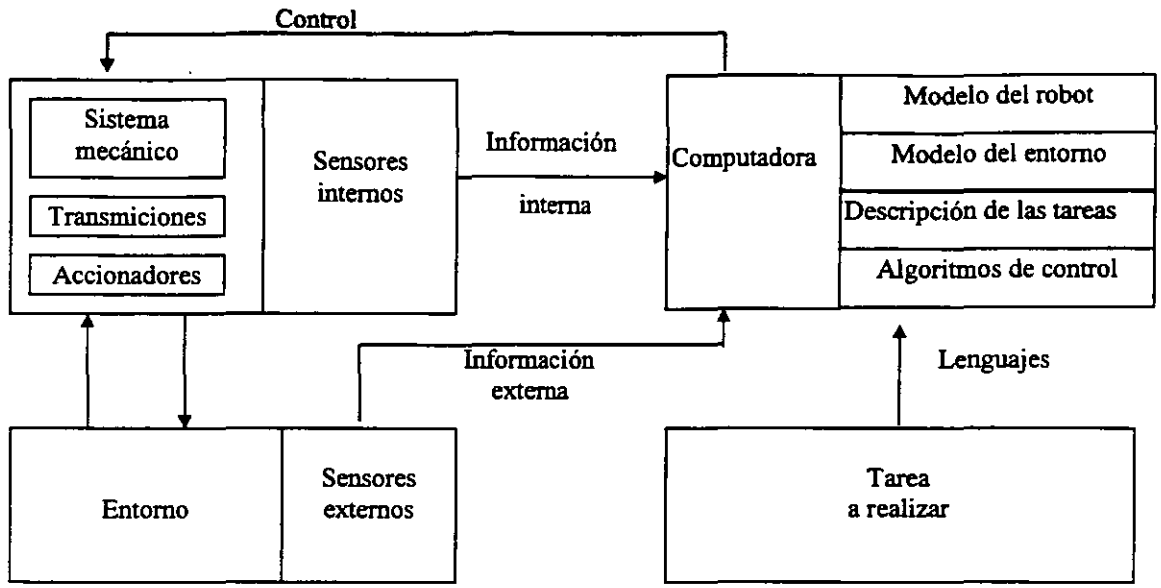


Figura 1-1. Descripción de un robot en funcionamiento.

1) *El sistema mecánico articulado*: Dotado de sus motores (que se llaman también accionadores o actuadores); estos pueden ser eléctricos, hidráulicos o neumáticos. Arrastran a las articulaciones del robot, mediante transmisiones, también de diversa naturaleza (cables, cintas, correas con muescas, engranes, tornillos sin fin, etc.).

Para conocer en todo instante la posición de las articulaciones, se recurre a los captadores (potenciométricos, codificadores ópticos, etc.). Estos captadores se denominan propioceptivos, por analogía con el sistema humano.

2) *El entorno*: Es el universo en el que está sumergida la primera entidad. Para los robots con puesto fijo, se reduce a lo que se encuentra en el espacio alcanzable del robot, definido por el volumen barrido cuando éste pasa por todas las configuraciones posibles. En este entorno, el robot encontrará obstáculos que debe evitar y objetos de interés, es decir sobre los que debe actuar.

En consecuencia, existe una interacción entre la primera entidad (el robot físico) y el entorno; se toman informaciones sobre el estado de ese entorno (comparable al estado del robot), mediante los captadores que denominamos exteroceptivos (es decir, que permiten situar lo que hay en el exterior del robot físico con relación a el); encontrándose en esta rama, las cámaras, detectores de fuerza, captadores de proximidad, captadores táctiles, etc.

3) *Las tareas a realizar*: Es el trabajo que se desea que haga el robot . Es preciso por lo tanto poder describirlo; esto se hace mediante lenguajes que pueden ser por gestos (se enseña al robot lo que debe hacer), orales (se le habla) o por escrito (se le escribe en un lenguaje que comprenda).

4) *El cerebro del robot*: Es el órgano de tratamiento de la información, para los robots menos evolucionados casi siempre es un autómata programable. En el caso de los robot avanzados es una computadora, microprocesador o microcontrolador; el cerebro del robot cuenta con los siguientes elementos:

a) *Un modelo del robot físico*; es decir, las relaciones entre las señales de excitación de los accionadores y de los desplazamientos que son consecuencia de ellas.

b) *Un modelo del entorno*; es decir, una descripción de lo que se encuentra en el espacio que puede alcanzar, por ejemplo, las zonas que no debe atravesar ya que hay obstáculos.

c) *Programas*; los cuales le permiten comprender las tareas que se le pide que realice, además de controlar el robot físico, con el fin de que éste ejecute lo que debe; son algoritmos de control. El microprocesador en todo instante:

i) Percibe el estado del robot gracias a la información interna;

ii) Percibe el estado del entorno gracias a la información externa;

iii) Recurre a diversos modelos y programas registrados;

iv) Genera una orden (es decir, las señales de potencia de los accionadores), que hace progresar al robot físico hacia la ejecución correcta de la tarea que se le ha pedido.

1.3 Prototipo en diseño

La figura 1-2 muestra un robot móvil que tiene como base a un microcontrolador, quien contará con diversos sensores y actuadores; en los capítulos siguientes se analizarán los más adecuados para nuestro diseño y las posibles opciones con los que podemos contar.

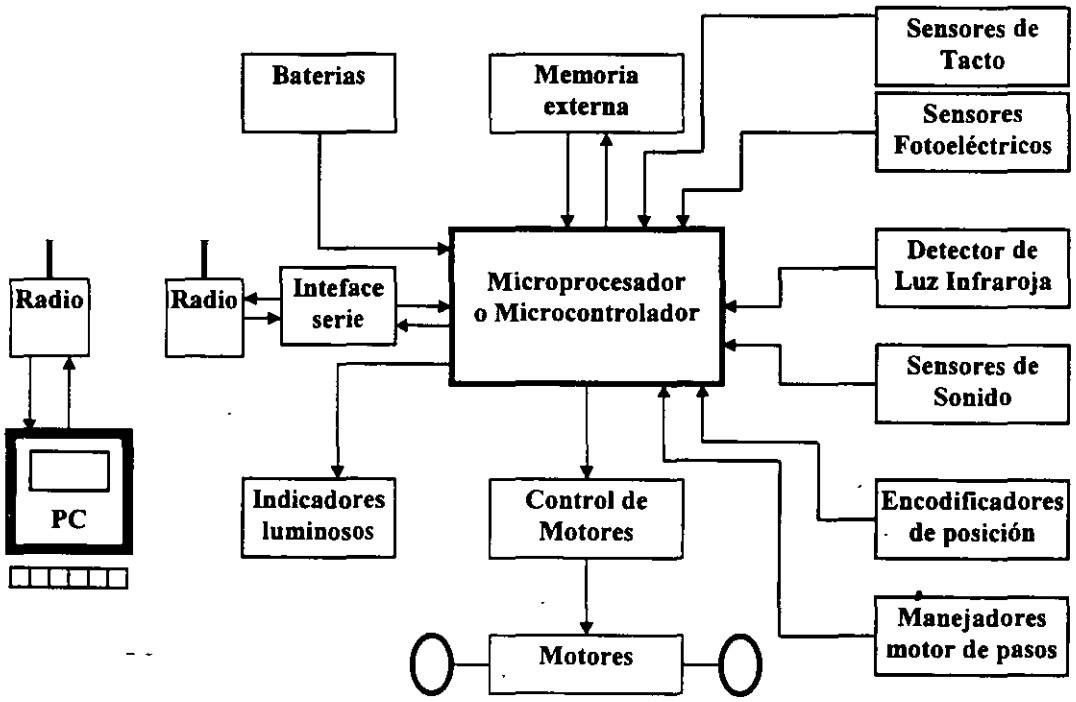


Figura 1-2. Diagrama general del robot móvil.

CAPITULO 2
Microprocesadores y Microcontroladores

Es difícil dar una definición exacta de lo que es un microprocesador, sobre todo por que en los últimos años el desarrollo de la electrónica en esta área ha dado lugar de un sin número de estos dispositivos, diseñados con enfoques hacia diversos campos de aplicación. Sin embargo todos los microprocesadores tienen ciertos atributos comunes que los distinguen de otros elementos electrónicos.

Un microcontrolador y un microprocesador son una combinación de dispositivos y circuitos digitales, que pueden realizar una secuencia programada de operaciones, a la cuál se le llama programa, con un mínimo de intervención humana, el programa lo ejecuta sobre datos que recibe, de los cuales obtiene resultados.

Las características principales de un microprocesador son su *universalidad* y su *programabilidad* que hacen de él un dispositivo tan versátil y poderoso, que puede utilizarse como elemento inteligente o *cerebro* en aplicaciones que van desde una computadora personal hasta un rastreador de satélites o un sistema de control de los robots móviles.

2.1 Conceptos Básicos de los Microprocesadores

Esencialmente, un microprocesador es un circuito de alta escala de integración (LSI), compuesto de muchos circuitos simples como son flip-flop, contadores, registros, decodificadores, comparadores, etc. Todos ellos dentro de la misma pastilla de silicio, de modo que el microprocesador puede ser considerado un dispositivo lógico de propósito general o universal.

2.1.1 Primeras definiciones

La programabilidad se refiere a la capacidad que tiene un microprocesador para que su función sea definida en un programa. El programa consta de una serie de instrucciones relacionadas, ejecutadas secuencialmente (una a la vez) por el microprocesador y que pueden implicar operaciones lógicas o aritméticas. Las instrucciones se especifican por medio de un código especial que constituye el lenguaje del microprocesador.

De lo anterior se puede deducir la existencia de dos géneros diferentes de elementos que se complementan y que juntos delimitan el concepto del microprocesador como un dispositivo útil. El primero de ellos lo forman el circuito mismo, sus componentes electrónicos, sus alambres de interconexión y, en general todo aquello determinado por la configuración física del circuito integrado. A este género se le conoce como el **HARDWARE** o soporte físico del microprocesador, ya que su misma naturaleza no admite modificaciones. Por otro lado, se encuentran el código de instrucciones, los algoritmos y los programas que determinan el comportamiento del microprocesador, los cuales pueden ser alternados cuantas veces sea necesario, con el fin de adaptar y optimizar el desempeño del dispositivo a algún campo de aplicación en particular. Este segundo género recibe el nombre de **SOFTWARE** o soporte lógico.

2.2 Unidades funcionales básicas de un microprocesador

Aun cuando el microprocesador se considera un complejo sistema de microprocesamiento, constituido por una gran variedad de circuitos interrelacionados, es posible agrupar estos circuitos de acuerdo a la finalidad del trabajo específico que desempeñan dentro del sistema. Hay otras unidades funcionales que también pueden existir en un microprocesador; éstas son las Memoria de datos y los Puertos de entrada/salida.

Para que a un circuito se le pueda dar el nombre de microprocesador, debe contener en una sola pastilla, al menos los siguientes sistemas: la unidad de control, la unidad aritmético/lógica y registros internos.

Cuando un microprocesador posee únicamente las tres unidades funcionales básicas, se le conoce como UNIDAD DE PROCESAMIENTO CENTRAL (CPU) o simplemente MICROPROCESADOR (MP). Si un mismo circuito integrado tiene, además de las unidades básicas, otras tales como memoria (de programas y/o datos) y puertos, este circuito ya no se considera estrictamente un microprocesador, porque las unidades que contiene le dan la capacidad de una computadora. Por esta razón a estos microprocesadores se les llama también MICROCOMPUTADORAS EN UNA PASTILLA (Single Chip Microcomputers) o MICROCONTROLADOR.

En la figura 2-1 se muestra un diagrama de bloques de un microprocesador y de una microcomputadora en una pastilla.

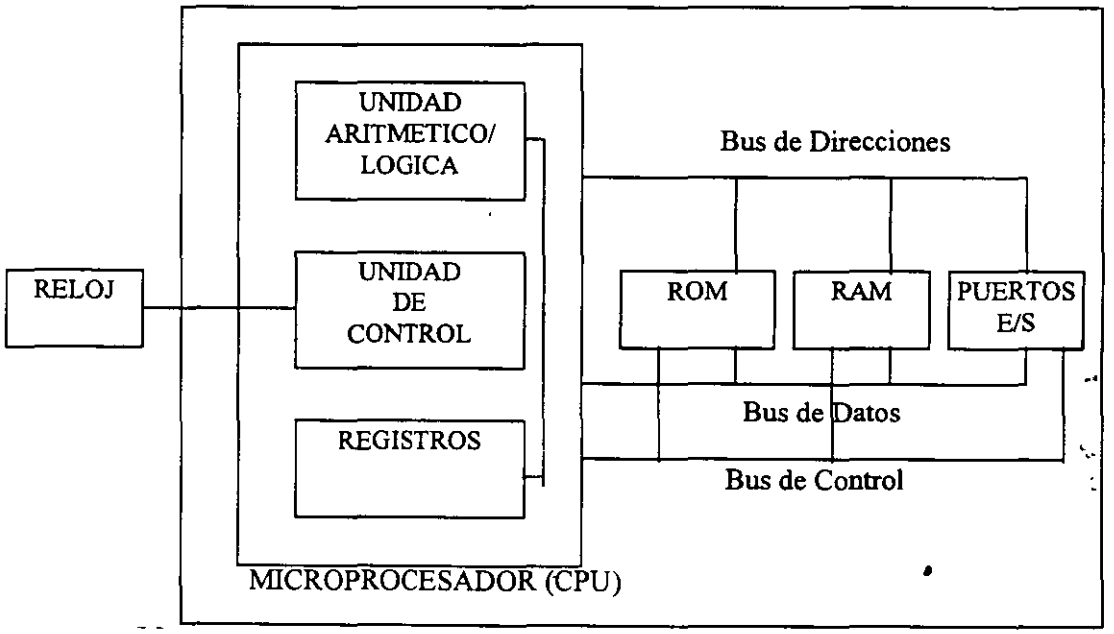


Figura 2-1. Diagrama de bloques de un microprocesador y de una microcomputadora en una sola pastilla.

A continuación se describirán las unidades funcionales que encuentran dentro de la CPU.

2.2.1 La unidad de control

La circuitería de control es la unidad funcional primaria dentro del microprocesador. Utilizando señales de reloj, la unidad de control mantiene la secuencias de eventos apropiada para llevar a cabo cualquier tarea de procesamiento; es decir, el microprocesador es un dispositivo síncrono.

La actividad fundamental de un microprocesador, regulada por la unidad de control, es cíclica y consiste en la búsqueda y obtención de datos e instrucciones, y en la ejecución secuencial de estas últimas. Después de que una instrucción ha sido obtenida y decodificada, la circuitería de control envía las señales apropiadas a dispositivos internos como externos a la CPU.

Para iniciar la acción de procesamiento indicada por la instrucción. Frecuentemente, la unidad de control es capaz de responder a señales externas que alteran el estado del microprocesador, ya sea interrumpiendo temporalmente su funcionamiento o provocando la ejecución de instrucciones especiales.

La parte más importante de la unidad de control lo constituye el GENERADOR DE CICLO DE MAQUINA (GCM), que se encarga de producir las señales de control, derivándolas de un reloj u oscilador maestro como referencia; la figura 2-2 presenta el diagrama de bloques de la unidad de control y su relación con otros elementos de la CPU.

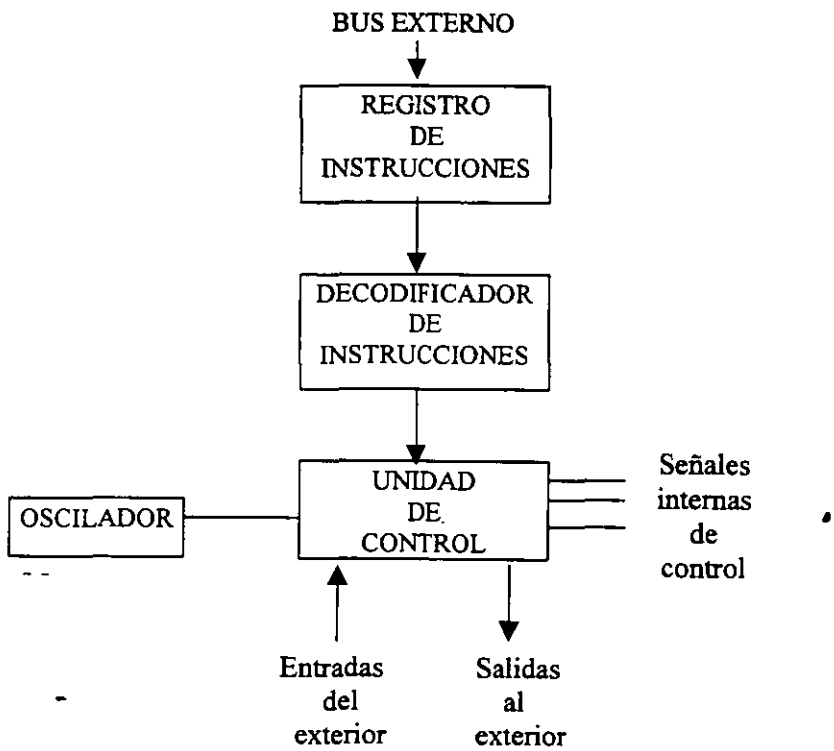


Figura 2-2. Unidad de Control.

2.2.2 Unidad aritmético/lógica

Todos los microprocesadores contienen una unidad aritmético/lógica que con frecuencia se conoce como ALU (Arithmetic/Logic Unit). La ALU como su nombre lo indica, es la parte del microprocesador que lleva a cabo las operaciones aritméticas y lógicas en los datos binarios (figura 2-3). Algunas de ellas se aplican sobre dos operandos, mientras otros requieren solo uno.

La ALU generalmente ejecutará las siguientes operaciones:

1.- Suma aritmética;
2.- Funciones lógicas AND, OR, EXOR;
3. Complemento;
4.- Rotación hacia la derecha o izquierda, etc.

Además, la ALU contiene un conjunto de flip-flops llamados BANDERAS (flags), los cuales guardan información relacionada con el resultado de una operación aritmética o lógica.

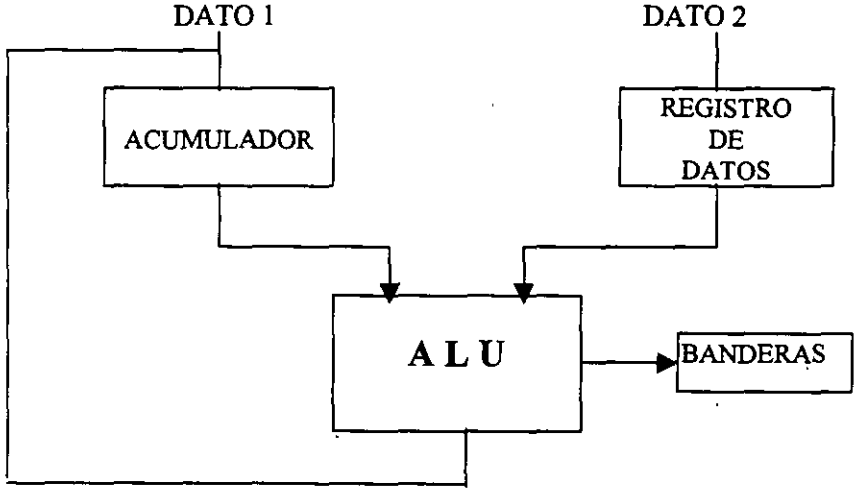


Figura 2-3. Unidad Aritmético/Lógica.

2.2.3 Los registros internos

Los registros internos son unidades de almacenamiento temporal dentro de la CPU, Algunos de ellos tienen usos específicos, mientras que otros son de propósito general.

Para llevar la cuenta de cuál instrucción es la que debe ejecutarse en seguida, la unidad de control mantiene un registro con la dirección de la siguiente instrucción en el programa. A este registro se le llama CONTADOR DE PROGRAMA (program Counter) o PC. El microprocesador actualiza el contenido del PC incrementándolo cada vez que obtiene una instrucción de la memoria.

Una de las entradas de control al microprocesador es la entrada de RESET (restablecer); cuando se restablece el microprocesador, la unidad de control carga el contador del programa con ceros. Este valor inicial establece la dirección de memoria de donde se va a obtener la primera instrucción.

Después que se ha obtenido una instrucción de la memoria, la CPU la almacena en un registro conocido como REGISTRO DE INSTRUCCIONES (Instruction Register) o IR. La instrucción almacenada en el IR es decodificada y usada para activar una de varias líneas. Cada línea representa un conjunto de actividades asociadas con la ejecución de una instrucción particular. El dispositivo que traduce la instrucción en acciones concretas es el DECODIFICADOR DE INSTRUCCIONES (Instruction Decoder).

La primera palabra de una instrucción es el código de operación para esa instrucción. El CODIGO DE OPERACION indica a la unidad de control las operaciones requeridas en la ejecución de la instrucción.

Las instrucciones de un microprocesador frecuentemente requieren más información de lo que puede contener una sola palabra de memoria. Por lo tanto, es común que las instrucciones consten de dos o tres palabras.

Las otras palabras son datos que representan ya sea una dirección o una constante. Después de que el código de operación se ha leído de la memoria y puesto en el registro de instrucciones, su decodificación indica si la instrucción requiere información adicional. Las partes de una instrucción de varias palabras están contenidas en localidades sucesivas de la memoria.

Existe un registro íntimamente relacionado con la ALU, denominado ACUMULADOR. Generalmente el acumulador contiene uno de los operandos que serán manipulados por la ALU y, también muy frecuentemente, el resultado de la operación se deposita en ese registro, reemplazando a uno de los operandos originales (figura 2-3).

1.2.4 La memoria del programa

Aunque anteriormente se dijo que la memoria no era parte integrante del microprocesador considerándolo como CPU, es conveniente mencionar la memoria del programa, puesto que sin ella el microprocesador se convierte en un dispositivo inservible.

La MEMORIA DEL PROGRAMA es una memoria de lectura solamente. Como por ejemplo una ROM o una EPROM que contiene el programa, es decir, la secuencia de instrucciones que va a determinar las tareas que realice el microprocesador. En algunos casos la memoria del programa también almacena parámetros o tablas de datos que no sufren modificaciones. Usualmente, las EPROM se utilizan durante la etapa de desarrollo del prototipo del sistema, para así poder depurar y alterar el programa.

La ROM se prefiere cuando el sistema ya se encuentra en producción y se ha llegado a la versión final del programa.

2.3 Conceptos Adicionales de Microprocesadores

Con frecuencia un microprocesador contiene un número de registros adicionales que no tienen asignada ninguna función en particular, sino que más bien se usan el tareas generales como lugares de almacenamiento temporal para guardar operandos o resultados intermedios.

2.3.1 Los registros internos de propósito general

La disponibilidad de registros de propósito general elimina la necesidad de mover los datos de un lado a otro entre la memoria y el acumulador, mejorando así la velocidad de procesamiento así como la eficiencia.

2.3.2 La memoria de datos

La memoria de datos es una memoria de lectura escritura (RAM), cuyo objetivo es permitir el almacenamiento temporal de datos o programas de aplicación. Por sus características de lectura/escritura, la información que reside se puede alterar fácilmente; sin embargo se pierde cuando se apaga la fuente de alimentación.

La mayoría de datos se puede considerar como una extensión de los registros internos de propósito general, ya que cada una de sus localidades es precisamente un registro. La diferencia está en que los registros de la memoria de datos son externos al microprocesador, y por lo tanto la velocidad de la transferencia de datos es más lenta.

2.3.3 Los puertos de entrada/salida

Los puertos de entrada/salida son los circuitos que se encargan de establecer el contacto del microprocesador (CPU) con el mundo exterior.

Los puertos se conectan a dispositivos periféricos que generan información para ser procesada por la CPU (dispositivos de entrada) o que aceptan datos provenientes del microprocesador y los transforma en información significativo para el mundo exterior (dispositivos de salida). Estos periféricos incluyen una gran variedad de elementos electrónicos y electromecánicos, cuya complejidad va desde unos simples interruptores y lámpara indicadoras, hasta complejos sistemas de adquisición de datos, pantallas de vídeo, etc.

2.3.4 Los buses de interconexión

Un microprocesador se comunica con los otros circuitos del sistema a través de grupos de líneas o buses de interconexión. Con base en el tipo de información que conducen, las líneas de interconexión se pueden agrupar en tres buses: datos, direcciones y control.

El BUS DE DATOS es un conjunto de líneas bidireccionales, que transportan información del microprocesador hacia la memoria o puertos y de éstos hacia el microprocesador.

El bus BUS DE DIRECCIONES es unidireccional, es decir, por él solamente circula información proveniente del microprocesador. Comprende a las líneas que transmiten una dirección generada por el CPU, la cuál selecciona a un puerto o a una localidad de memoria. Esta dirección especifica el origen o destino de la información que transmitirá por el bus de datos.

La sincronización y el sentido de la transferencia de información es el bus de datos (hacia dentro o hacia afuera del microprocesador) y el tipo de transferencia se indica por medio de las señales de control originadas por la CPU. Todas ellas conforman el llamado BUS DE CONTROL. Cada una de las señales en el bus de control es unidireccional. Algunas de ellas son salidas del microprocesador, mientras otras son entradas a el.

2.4 Clasificación de los microprocesadores

Existen dos criterios principales para la clasificación de los microprocesadores, uno se basa en la longitud de palabra y el otro es la tecnología de fabricación.

La longitud de palabra se refiere al número de bits que puede procesar simultáneamente un microprocesador y está determinada por su arquitectura, es decir, por el tamaño de los registros, de la ALU y de sus buses internos. La longitud de palabra ha crecido a través de los años, desde 4 bits del primer microprocesador hasta los 32 bit de los microprocesadores más recientes.

En lo que toca a la tecnología de fabricación, los primeros microprocesadores se implantaron con tecnología PMOS; sin embargo, actualmente la tecnología de fabricación más difundida es la NMOS. Ultimamente se ha desarrollado bastante la tecnología CMOS para dispositivos con bajo consumo de energía.

2.5 Características adicionales de los microcontroladores

De acuerdo a la tecnología y funcionalidad de un microcontrolador, agrega en la mayoría de los casos, los subsistemas del Temporizador; convertidor A/D; Comunicación Serial Síncrona y Asíncrona, Sistemas de manejo de puertos paralelos, y un manejo de instrucciones más extenso.

La característica principal de un microcontrolador, es la inclusión en un chip de la gran mayoría de recursos, necesarios para controlar el funcionamiento de dispositivos o instrumentos, lo que es permitido por los subsistemas que lo conforman, y hace de fácil adaptación la expansión a un sistema con mayores recursos.

Un diagrama típico de un microcontrolador es el que se muestra a continuación, donde se trata del diagrama interno del microcontrolador MC68HC11E9, del cuál se proporcionará mayor información de su forma de operación en el apéndice A.

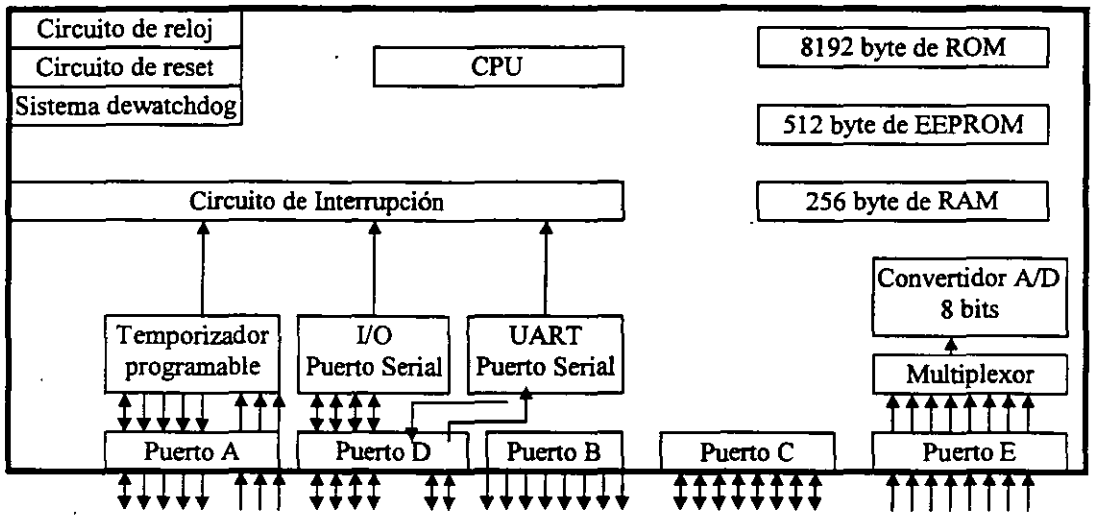


Figura 2-4. Diagrama interno del microcontrolador MC68HC11.

CAPITULO 3
Sensores

3.1. Transductores, sensores y actuadores

Se denomina transductor, a todo dispositivo que convierte una señal física en una señal correspondiente pero de una forma física distinta; es por lo tanto, un dispositivo que convierte un tipo de energía en otro.

La tendencia actual, particularmente en robótica, es emplear el término sensor para designar el transductor de entrada, y el término de actuador o accionamiento para designar el transductor de salida. Los primeros pretenden la obtención de información, mientras que los segundos buscan la conversión de la energía.

Un transductor es un dispositivo que convierte un tipo de variable física (fuerza, presión, temperatura, velocidad, intensidad luminosa, etc.) en otro tipo de variable; las más comunes voltaje o corriente eléctrica, ya que de esta manera es más conveniente su uso e implementación en cualquier sistema electrónico.

Un sensor es un transductor que es usado para hacer la medición de una variable física de interés. La mayoría de los sensores o transductores requieren de calibración, para que estos resulten lo más confiables al realizar la medición en turno. La calibración es el procedimiento por el cuál se fijan los parámetros es decir; tener un valor de referencia entre la variable medida y la señal de salida.

Dado que hay seis tipos de señales: mecánicas, térmicas, magnéticas, eléctricas, ópticas y moleculares (químicas), cualquier dispositivo que convierta una señal de un tipo en una señal de otro tipo debería considerarse un transductor, y la señal de salida podría ser de cualquier salida forma física útil. En la práctica, no obstante, se consideran transductores por antonomasía aquellos que ofrecen una señal de salida eléctrica. Esto se debe al interés de este tipo de señales en la mayoría de procesos de medida. Los sistemas de medida electrónicos ofrecen, entre otras las siguientes ventajas:

1. Debido a la estructura electrónica de la materia, cualquier variación de un parámetro no eléctrico de un material, viene acompañada por la variación de un parámetro eléctrico; eligiendo el material adecuado, permite realizar transductores con salida eléctrica para cualquier magnitud física no eléctrica.

2. Dado que en el proceso de medida no conviene extraer energía del sistema donde se mide, lo mejor es amplificar la señal de salida del tranductor.

3. Además de la amplificación, hay una gran variedad de recursos, en forma de circuitos integrados para acondicionar o modificar las señales eléctricas. Incluso hay transductores que incorporan físicamente en un mismo encapsulado como parte de estos recursos.

4. Existen también numerosos recursos para presentar o registrar información si se hace electrónicamente, pudiendo manejar no solo datos numéricos, sino también textos, gráficos y diagramas.

5.- La transmisión de señales eléctricas es más versátil que la de señales mecánicas, hidráulicas o neumáticas .

Los transductores analógicos proveen una señal de este tipo, ya sea voltaje o corriente, esta señal puede ser interpretada como el valor de la variable física; mientras los transductores digitales producen una señal de respuesta de tipo digital, que incluyen niveles lógicos. De hecho estos son los más utilizados en sistemas de automatización y procesos de control, ya que este tipo de señal es compatible con los microprocesadores .

3.1.2 Interfaces, dominios de datos y conversiones

Con el término interfaz se designa, al conjunto de elementos que modifican las señales, cambiando incluso el dominio de datos, pero sin cambiar su naturaleza, es decir; permaneciendo siempre en el dominio eléctrico.

Se denomina dominio de datos al nombre de una magnitud mediante la que se representa o transmite información.

En el dominio analógico, la información está en la amplitud de la señal, bien se trate de carga, corriente, tensión o potencia. En el dominio temporal, la información no está en las amplitudes de las señales, sino en las relaciones temporales: periodo o frecuencia, anchura de pulsos, fase, etc. En el dominio digital, las señales tiene sólo dos niveles lógicos, con valor de 0 o 1 . La información puede estar en el número de pulsos, o venir representada por palabras serie o paralelo codificadas.

3.2 Tipos de sensores

Según la señal de salida los sensores se clasifican en analógicos o digitales. En los analógicos la salida varía, a nivel macroscópico de forma continua. En los sensores digitales, la salida varía en forma de saltos o pasos discretos; no requiere conversión A/D y la transmisión de su salida es más fácil, tienen también mayor fidelidad, fiabilidad y muchas veces mayor exactitud.

Atendiendo al modo de funcionamiento, los sensores pueden ser de deflexión o de comparación. En los sensores que funcionan como deflexión, la magnitud medida produce algún efecto físico, que engendra algún efecto similar, pero opuesto, en alguna parte del instrumento, y que esta relacionado con alguna variable útil, por ejemplo un dinamómetro.

En los sensores que funcionan por comparación, se intenta mantener nula la deflexión mediante la aplicación de un efecto bien conocido, opuesto al generado por la magnitud a medir; hay un detector del desequilibrio y un medio para restablecerlo, por ejemplo una balanza manual.

Según el tipo de relación entrada-salida, los sensores pueden ser de orden cero, de primer orden, de segundo orden o de orden superior. El orden está relacionado con el número de elementos almacenadores de energía independientes que incluye el sensor, y repercute en su exactitud y velocidad de respuesta. Esta clasificación es de gran importancia cuando el sensor forma parte de un sistema de control de lazo cerrado.

Desde el punto de vista de ingeniería electrónica, es más atractiva la clasificación de los sensores de acuerdo con el parámetro variable: resistencia, capacidad, inductancia, añadiendo luego los sensores generadores de tensión, carga o corriente, etc.

3.3 Características estáticas de los sistemas de medida

Características deseables de los sensores:

1. Exactitud;
2. Precisión;
3. Rango de operación;
4. Velocidad de respuesta;
5. Calibración;
6. Fiabilidad;
7. Costo, tamaño reducido y comodidad de operación.

3.3.1 Exactitud, fidelidad, sensibilidad

La exactitud es la cualidad que caracteriza la capacidad de un instrumento de medida de dar indicaciones que se aproximen al verdadero valor de la magnitud medida. El valor exacto, verdadero o ideal, es el que se obtendría si la magnitud se midiera con un método ejemplar.

La exactitud de un sensor se determina mediante la denominada calibración estática. Consiste en mantener todas las entradas excepto a una en una a un valor constante. La entrada en estudio se varía entonces lentamente, tomando sucesivamente valores constantes dentro de un margen , y se van anotando los valores que toma la salida. La representación de estos valores en función de los de la entrada define la curva de calibración.

La discrepancia entre la indicación del instrumento y el verdadero valor de la magnitud medida se denomina error. La diferencia entre la indicación del instrumento y el verdadero valor se denomina error absoluto.

$$\text{error absoluto} = \text{resultado} - \text{valor verdadero}$$

Sin embargo, lo más común es especificar el error como cociente entre el error absoluto y el verdadero valor de la magnitud medida, cociente que se denomina error relativo.

$$\text{error relativo} = \frac{\text{error absoluto}}{\text{valor verdadero}}$$

La fidelidad (precisión) es la cualidad que caracteriza la capacidad de un instrumento de medida, de dar el mismo valor de la magnitud medida, al medir varias veces en unas mismas condiciones determinadas, prescindiendo de su concordancia o discrepancia con el valor real de dicha magnitud.

La sensibilidad o factor de escala es la pendiente de la curva de calibración, que puede ser o no constante a lo largo de la escala de medida. Para un sensor cuya salida esté relacionada con la entrada x mediante la ecuación:

$y = f(x)$, la sensibilidad en el punto x_a, $S(x_a)$, es

$$S(x_a) = \left.\frac{dy}{dx}\right|_{x=x_a}$$

En los sensores interesa tener una sensibilidad alta y, si es posible, constante. Para un sensor con respuesta:

$$y = Kx + b$$

la sensibilidad es S = k, para todo el margen de valores de x aplicables. Para uno cuya respuesta sea

$$y = kx^2 + b$$

la sensibilidad es S = 2kx, y varía a lo largo de todo el margen de medida.

En este capítulo se tratarán los tipos de sensores utilizados en el diseño del Robot Móvil y como se realizará la interfaces con el microcontrolador.

Se considera para la instrumentación; de la mayor parte de los sensores, la característica del convertidor Analógico/Digital del HC11, el cuál es sensible solamente en el rango de 0 a 5 Volts, con 8 bits de resolución, de tal forma que se podrán tener 256 posibles niveles discretos.

3.4 Sensores fotoeléctricos

En los últimos años, técnicas comunes a los campos de la óptica y la electrónica, han originado una nueva tecnología denominada *optoelectrónica*; este ha sido uno de los campos de crecimiento más rápido en la industria moderna, puesto en evidencia en las siguientes aplicaciones.

- Maquinas de fotocopia;
- Infrarrojos para el seguimiento de cohetes y para visión nocturna;
- Sistemas de reconocimiento desde aviones o cápsulas espaciales, mapas meteorológicos y detección de incendios forestales;
- Aplicación de láser, tales como curación de desprendimiento de retina, curación de cáncer, comunicaciones, identificación de objetivos y la holografía;
- Control fotoeléctrico industrial como contadores, sistemas de parada y arranque, control de nivel de proximidad, medida de grosores, procesos de alimentos y clasificación;
- Lectura en computadoras de cintas y tarjetas perforadas;
- Encendido automático del alumbrado público y sistemas de exposición fotográfica;
- Detección de incendio y robo;
- Espectrometría;
- Reconocimiento de caracteres;
- Posicionado y alineación de servosistemas, etc.

Los dispositivos fotoelectrónicos utilizan la luz como medio de transmisión entre un emisor y un receptor, es decir, una fuente luminosa y un elemento fotosensible. La luz es una radiación electromagnética comprendida en una banda de longitudes de onda de unos 100 a 8000 nm. La longitud de onda caracteriza el color de la luz. En una banda de 400 nm (violeta) a 700 nm (rojo), la luz es perceptible por el ojo humano (luz visible). Por encima de los 700 nm, la luz se denomina infrarroja (IR) y por debajo de los 400 nm, ultravioleta (UV).

3.4.1 Elementos fotosensibles.

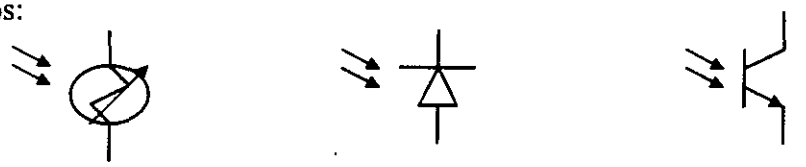
Los receptores de dispositivos fotoelectrónicos trabajan con elementos fotoeléctricos, que en la actualidad son casi exclusivamente semiconductores.

En la fotoresistencia se aprovecha la variación de la resistencia que se produce al incidir la luz y que es independiente del sentido de la corriente.

En el fotodiodo se aprovecha la variación de resistencia que se produce en la iluminación, posee polaridad determinada.

En el fototransistor se forman, al incidir la luz, pares de portadores de carga, que se refuerzan por la corriente engendrada por este.

Símbolos:



a) b) c)

Figura 3-1. Símbolos de los elementos fotosensibles: a) Fotoresistencia, b) Fotodiodo y c) Fototransistor.

3.4.1.1 Fotoresistencia

Son componentes cuya resistencia disminuye cuando son expuestas a la luz, de construcción sencilla, económicas, de gran sensibilidad, aunque de respuesta lenta. Su resistencia es muy elevada en la obscuridad. La variación del valor de la resistencia , al ser iluminado este elemento, es independiente del sentido de la corriente, por cuya razón es posible su empleo en corriente alterna.

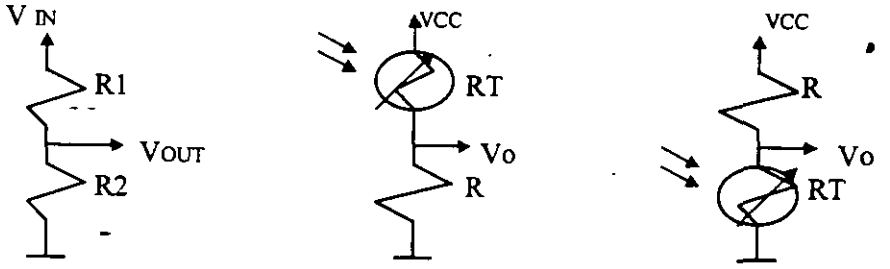


Figura (a) Figura (b) Figura (c)

Figura 3-2. Circuito de detección para una fotoresistencia.

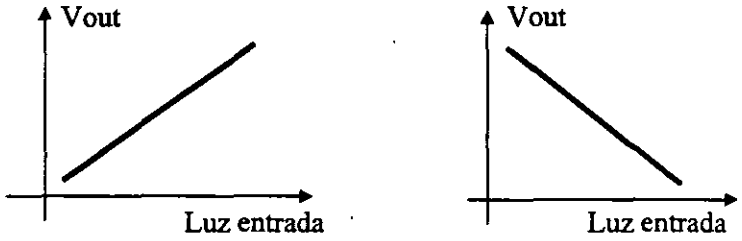
Analizando los circuitos de la figura 3-2a, 3-2b y 3-2c, se obtienen las siguientes ecuaciones:

$$VOUT = VIN\ (\ R2/R1+R2) \quad \text{ecuación 3.1}$$

$$Vo= VCC(\ R/R+RT\ ) \quad \text{ecuación 3.2}$$

$$Vo= VCC(RT/RT+R) \quad \text{ecuación 3.3}$$

Encontrando las gráficas entre el voltaje resultado de la incidencia de luz que se presenta en la fotoresistencia, según se muestra en la figura 3-3a y 3-3b donde se observa para la primer gráfica; que a mayor luz incidente en el fotoelemento, se obtendrá a la salida un mayor valor de voltaje, y para la gráfica 3-2b; se encuentra que a mayor intensidad de luz, se obtendrá a la salida un valor de voltaje menor.



Circuito (a) Circuito (b)

Figura 3-3. Gráfica de Voltaje de salida Vs. Intensidad de luz.

3.4.1.2 Fotodiodos

Los fotodiodos se hacen trabajar con una tensión inicial en sentido de bloqueo, Sin dicha tensión trabajan, en sentido de paso, como fotoelementos. La corriente en la obscuridad es reducida. Con resistencias de trabajo de elevado valor óhmico pueden engendrarse variaciones de tensión que alcanzan la plena tensión de servicio.

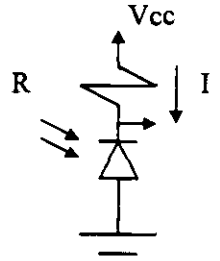


Figura 3-4. Circuito de detección de un fotodiodo.

3.4.1.3 Fototransistores

En un fototransistor actúa la luz sobre la zona de la base como una corriente de base, la que controla el espacio base-emisor y con ella la corriente del colector. Los fototransistores son más sensibles que los fotodiodos, aunque de respuesta más lenta.

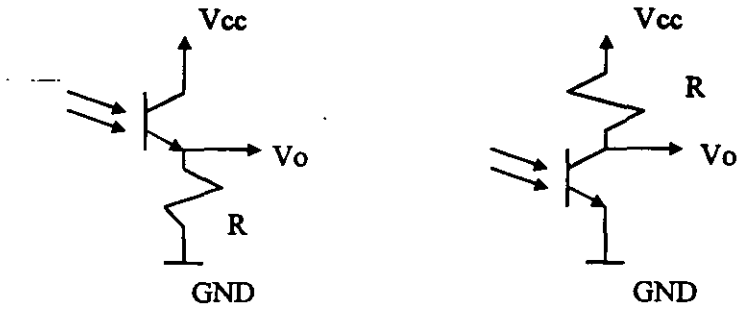


Figura 3-5. Circuito de detección de un fototransistor.

3.4.1.4 Detector de luz infrarrojo

Estos sensores detectan a una longitud de onda de 880 nm aproximadamente, donde el espectro no es perceptible al ojo humano, en conjunto para este circuito se requiere de un diodo emisor de luz infrarrojo y un detector (GP1U52X), donde este último contiene en el mismo empaque su amplificador, filtro y un limitador, esta unidad es distribuida por Radio Shack y Sterling Electronics.

Este detector responde a una señal modulada transmitida por el led infrarrojo, su frecuencia de operación es de 40 KHz. Esta frecuencia se puede realizar con varios arreglos de circuitos, en este caso, se implemento con el circuito integrado LM555; la figura 3-6 presenta el circuito utilizado.

El objetivo de este arreglo es detectar objetos que estén próximos al robot y que al encontrar dicho obstáculo, el robot pueda tomar la decisión adecuada para evitar la coalición.

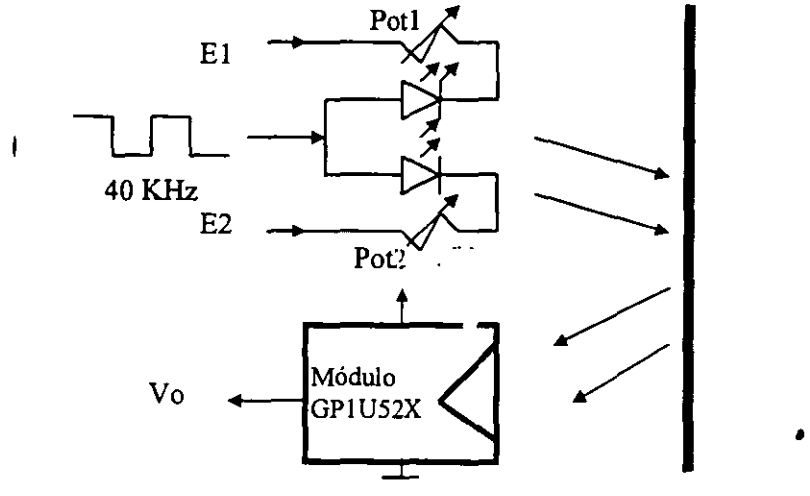


Figura 3-6. Transmisor y receptor infrarrojo.

Las señales ópticas de tipo infrarrojas, por su longitud de onda, pueden presentar una mayor inmunidad a interferencias de señales externas no deseadas por el sistema de detección de un objeto, mediante el sensado del rebote de un haz óptico. Además ofrece una mayor facilidad de manejo a un menor costo sobre otras señales ópticas que pueden ofrecer mayor inmunidad al ruido.

La interfaz infrarroja consta de las siguientes partes:

i) Etapa emisora.- En esta etapa se va a generar una señal de rayos infrarrojos codificada, la cuál se va a transmitir desde un punto en dirección al área donde se desea detectar la presencia de un objeto.

ii) Etapa receptora.- Su función será la de captar la señal infrarroja de rebote desde el área de detección , filtrar todas aquellas señales ópticas del medio que no sean deseables para una correcta detección, la de eliminar aquellas señales infrarrojas que no contengan el código de la señal emitida, así como de evaluar la intensidad del rebote del haz recibido.

3.4.1.5 Sensores Piroeléctricos

Uno de los sensores más útiles para dotar al robot, con un mecanismo para interacturar con humanos, es un sensor piroeléctrico. Este tipo de sensor es esencial en la detección de movimiento y útil en alarmas antirrobo.

La salida de un sensor piroeléctrico, cambia cuando pequeñas variaciones en la temperatura de este sensor ocurre en un intervalo de tiempo, lo cual es detectado por su elemento principal cristal de litio-tantalio; por lo tanto la energía es inducida cuando el cristal cambia de temperatura.

Los sensores piroeléctricos son económicos, diseñados para detectar radiación en el rango de 8-10 micrómetros (el rango de energía infrarroja emitida por los humanos), por ello que sea conveniente su uso en alarmas de seguridad.

El sensor Eltec 442-3 incorpora dos cristales de litio-tantalio. El amplificador diferencial suministra los voltajes a través de los cristales a la salida del sensor. En el caso que ambos cristales estén a la misma temperatura, el sensor produce una señal de salida que permanece a un valor constante de 2.5 Volts (considerando una fuente de 5 Volts).

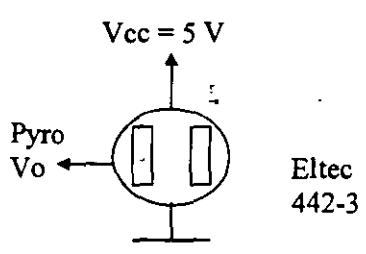


Figura 3-7. Sensor piroeléctrico y su acondicionamiento.

Si una persona camina en frente del sensor, moviéndose de izquierda a derecha, la señal levanta a un valor de 1 Volt y después bajará el mismo voltaje, para finalmente regresar al voltaje inicial; por otro lado cuando la persona se desplaza de derecha a izquierda, sucederá lo inverso, es decir la señal primero bajará, entonces subirá y después se establecerá, como se muestra en la figura 3-8.

3.4.1.6 Cámaras

La tecnología actual de las cámaras de vídeo ha evolucionado rápidamente, por lo que hoy en día se cuentan con cámaras muy compactas y de costo no tan elevado, la figura 3-9 presenta un ejemplo de una carrera de este tipo. La mayor utilidad se encuentra en los circuitos cerrados de televisión, implantados como sistemas de seguridad.

La adaptación de las cámaras de vídeo para la aplicación de robots móviles, se realiza transmitiendo esta señal a una estación de trabajo, utilizando un transmisor y un receptor, conectados al robot y a la estación de trabajo respectivamente, para poder realizar el procesamiento de la imagen.

Figura 3-9. Cámara de vídeo.

3.5 Sensores de contacto

Entre los más sensores más conocidos de este tipo se encuentran los microswitches, los flexibles y los detectores de fuerza.

3.5.1 Microswitch

Los microswitches son pequeños dispositivos, empleados con la finalidad de determinar cuando el robot ha chocado directamente con un objeto; son de fácil adaptación electrónica, ya que al tener contacto con el obstáculo, es posible obtener una señal que puede ser capturada sin necesidad de agregar una interfaz compleja, para ser procesada la información y realizar la acción programada (figura 3-10).

Se pueden utilizar tantos microswitches como se desee, ya que se pueden conectar en todo el alrededor del robot para que este tenga una mayor gama de puntos de contacto, para tener un control y monitoreo más exacto del entorno que rodea al robot.

Se pueden realizar dos tipos de arreglos con los microswitches, uno tomando la salida de cada uno de estos switches, para que ingresen al microcontrolador de manera independiente cada uno, la otra es haciendo un arreglo con varios de ellos, de manera que al hacer contacto cualquiera de ellos, se tenga como resultado un divisor de voltaje, para que al ser capturado por el convertidor analógico/digital del microcontrolador, se pueda realizar una tarea específica.

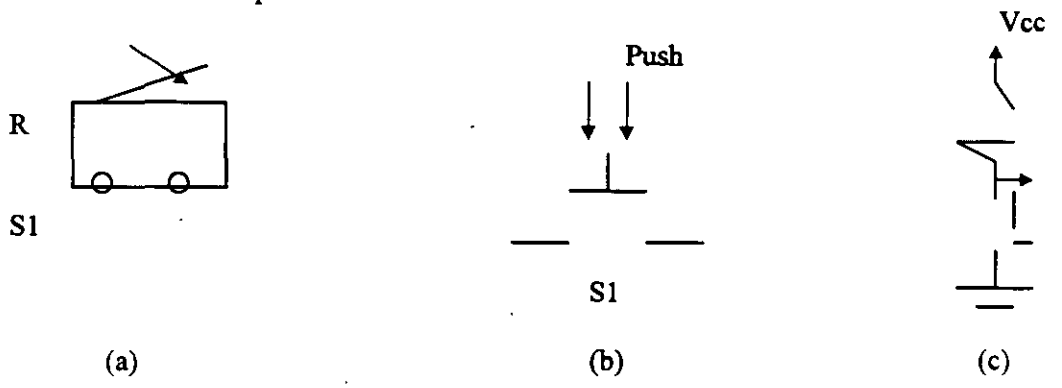


Figura 3-9. a) Esquema de un microswitch, b) Símbolo,
c) Circuito de acondicionamiento.

3.5.2 Sensores Flexibles

Otro sensor de gran ayuda para la detección de obstáculos, es el sensor de tipo flexible, un ejemplo se muestra a continuación. Este dispositivo utiliza pistas conductivas, conectadas entre dos electrodos, para entregar una resistencia variable, dependiendo del grado de flexibilidad. Esta resistencia se puede adaptar al microcontrolador, de la misma forma que una fotoresistencia, para tener un divisor de voltaje y conectarse a un canal del convertidor A/D. El inconveniente que se tiene es el rango reducido de variación entre cuando se encuentra sin flexionar y cuando se tiene la máxima inclinación (figura 3-11).

3.5.3 Sensores detectores de fuerza

Interlink Electronics fabrica una línea de sensores de fuerza, que son similares a los sensores de inclinación, basados en tecnología de pistas conductivas. La resistencia varía en relación a la fuerza aplicada en su superficie; presentan un rango muy grande de valores, a su vez están disponibles en varios tamaños y formas, que van desde 0.2 pul hasta 25 pulgadas de longitud.

3.6 Posición y orientación

Es de gran ayuda el saber la posición de un robot dentro de un entorno, así como conocer la distancia que se ha recorrido, desde una posición inicial a una final; en esta sección se estudiará el tipo de transductores que se emplearon para este objetivo .

3.6.1 Shaft Encoders

Un shaft encoders es un sensor que mide la posición o la velocidad de rotación de un motor, comúnmente son montados a la salida de un motor, a señal derivada de este sensor se puede considerar como la orientación de una rueda, la señal que entrega se considera como un tren de pulsos.

3.6.1.1 Potenciómetro

Un potenciómetro puede ser utilizado como un encodificador de posición absoluta, cada posición de la rueda produce una resistencia única. Los encodificadores absolutos son comúnmente usados para determinar la posición en brazos de robot, la figura 3-12 muestra un dispositivo de este tipo.

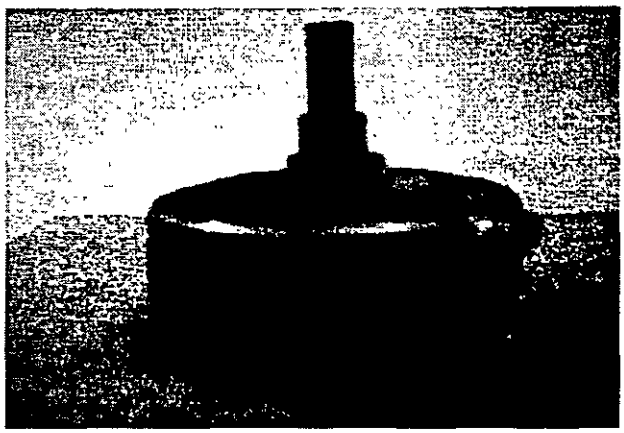


Figura 3-12. Potenciómetro.

3.6.1.2 Fotointerruptor

Una forma para lograr sincronizar la velocidad de giro de las ruedas del robot, se conecta un shaft encoder a cada motor. Se componen de dos dispositivos integrados en un mismo empaque, por un lado un led emisor de luz infrarroja y un fototransistor por el otro lado.

Se coloca un disco con ranuras, tantas como sea posible, para que se pueda obtener una mejor resolución para tener una aproximación más exacta de la velocidad, distancia o ángulo de giro de nuestro robot, de manera que cuando existe transmisión entre el infrarrojo y el fototransistor, se tenga una señal 0 o 1 lógico, dependiendo del arreglo implementado para este sensor y se pueda procesar por el microcontrolador; a este tipo de dispositivo se le conoce como fotointerruptor.

3.6.1.3 Fotoreflector

Otra implementación de un shaft encoder es con el dispositivo conocido como un fotoreflector, el cuál refleja la luz que emite en LED infrarrojo sobre un disco rayado, la cual es reflejada hacia un fototransistor. Un circulo marcado de líneas blancas y negras alternadas , será el mecanismo que realice la tarea de reflexión, según el color que esté pasando en ese momento por el sensor.

Un fotoreflector comercial que se puede utilizar, es el fabricado por Hamamatsu P306201, contiene internamente el circuito detector, el comparador y el amplificador, con lo que con dos resistencias externas es posible su implementación, la figura 3-13 presenta un ejemplo de modelos típicos, tanto de un foto interruptor como del fotoreflector, ya que es posible encontrarlos en diferentes modelos.

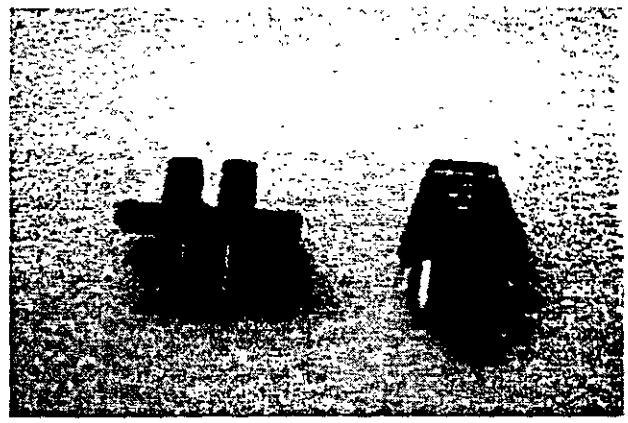


Figura 3-13. A la izquierda el foto interruptor; a la derecha el fotoreflector.

3.6.2 Gyros

Otro sensor que es de gran ayuda para determinar la velocidad de rotación de robot son los giroscopios. Son dispositivos mecánicos que emplean el principio de conservación de momento angular para almacenar uno o mas puntos de cortes en la misma dirección como el exterior del giroscopio.

3.6.3 Brújulas

Una brújula proporciona la manera de obtener información real de la orientación del robot, los cuales operan bajo el principio de atracción magnética; este tipo de sensores es de gran ayuda cuando se escriben algoritmos de navegación.

En áreas al aire libre, las brújulas son muy confiables, se calibran con respecto el campo magnético del polo norte, pero si el robot es operado dentro de cuartos, el uso de la brújula llaga a ser más problemático, esto se debe principalmente a los campos magnéticos producidos por los cables eléctricos, estructuras de acero de las construcciones y los componentes de metal del mismo robot, pueden todos ellos producir grandes errores en la lectura de la brújula, en el apéndice f se presenta el circuito de interfaz de un dispositivo de este tipo.

3.7 Sensores de autoevaluación

Un sensor de autoevaluación es cualquier sensor utilizado para medir el estado interno del robot; este tipo se sensores monitorean, de tal manera que pueden indicarnos cuando es tiempo de reemplazar la batería, cuando un motor se esta sobrecalentando o cuando un componente no está funcionando correctamente.

3.7.1 Sensado del nivel de una batería

Para sensar el nivel de voltaje de una batería, el robot indicará cuando debe recargarse, una forma sencilla de obtenerlo, es colocando un indicador de nivel de voltaje, colocando un arreglo conectado antes de ingresar el voltaje al regulador de voltaje, o después del mismo, como se presenta en la figura 3-14.

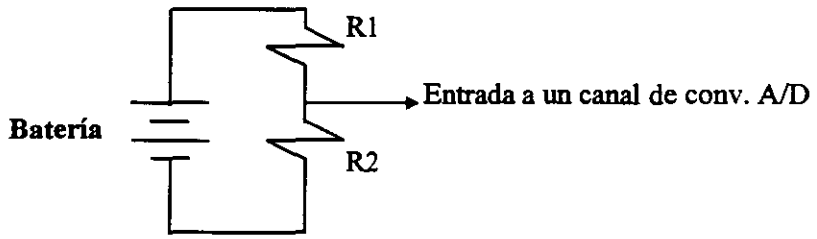


Figura 3-14. Circuito para detectar la carga de una batería.

3.7.2 Temperatura

Este sensor es importante para medir la temperatura interna del robot, ya que de esta manera se evitarían los sobrecalentamientos de los circuitos que lo conforman, por ejemplo, para altas temperaturas, la vida útil se reduce para el microcontrolador, los motores y las baterías; los dispositivos mas comúnmente utilizados pueden ser los termistores, quienes cambian su valor de resistencia dependiendo del la intensidad de calor existente; un sistema de ventilación correcto se requerirá en este caso.

CAPITULO 4
Motores y manejadores de potencia

4.1 Elementos motrices de los robots

Los dispositivos que en general producen el movimiento en los robots, pueden clasificarse en tres grandes grupos, según la energía que consumen:

a) Hidráulicos;
b) Neumáticos;
c) Eléctricos.

Los actuadores neumáticos e hidráulicos hacen uso de aire comprimido y de un flujo de presión, respectivamente. Dada su importancia y la creciente implantación en todas las áreas de la robótica, los elementos más utilizado son motores eléctricos.

Dentro de la gran variedad de tipos de motores eléctricos, los más adecuados para el movimiento son los de corriente directa y los motores de paso a paso.

Un motor eléctrico convierte energía eléctrica a energía mecánica; es posible encontrarlos de diversas formas y tamaños, existen motores de corriente directa CD y motores de corriente alterna CA, así como una gran variedad de modelos en cada uno de ellos.

Los motores de CA son típicamente utilizados para equipo industrial y son energizádos con la toma de la línea de corriente, ocupando para su funcionamiento, una , dos o tres fases, estos no son regularmente empleados en los robots móviles, ya que lo que se intenta es que estos sean lo más autónomos posibles, consiguiendo lo último suministrando la potencia con baterías de corriente continua.

Los motores de CD son comúnmente usados para trabajos relativamente pequeños, aunque sin dejar de ser importantes, ya que la mayor parte de los aparatos electrónicos y juguetes cuentan con cuando menos un motor de este tipo; así mismo, es posible encontrarlos fácilmente en el mercado, de varios tamaños, formas y en una gran variedad de voltajes de operación.

Existe un tipo de motores de CD con más de dos terminales, llamados servomotores, son los más utilizados en el diseño y construcción de robots; en nuestra aplicación se emplearon los motores de tres terminales, que son usados en el control de los aeroplanos de juguete, en carros de radio control y en el movimiento de los brazos manipuladores.

En el ensamble de estos motores, incorpora internamente un motor de CD, un sistema de engranes, un límite de giro, un potenciómetro de posición (feedback) y un circuito integrado para el control general; las terminales involucradas son polarización, tierra y el control de entrada, donde la señal de ancho de pulso indicara la velocidad del motor.

4.2 Interfaz con los motores

Un microprocesador no puede manejar directamente un motor, puesto que no puede suministrar la corriente requerida, para su correcto funcionamiento. Para incluir este control, es necesario adaptar algunos circuitos de interfaz, con el objetivo que el microcontrolador tenga el control de los motores. Esta interfaz de circuitos puede ser implementada con una gran variedad de tecnologías, como relevadores, transistores, MOSFETS y circuitos manejadores de potencia; en todas las tecnologías, la topología básica de circuitos es el mismo.

4.2.1 Empleo de transistores de potencia

Existen dos configuraciones básicas:

a) Configuración tipo T; y

b) Configuración tipo H.

4.2.2 Configuración tipo T

Se muestra en la figura 4-1, precisa de dos transistores de potencia y dos fuentes de alimentación.

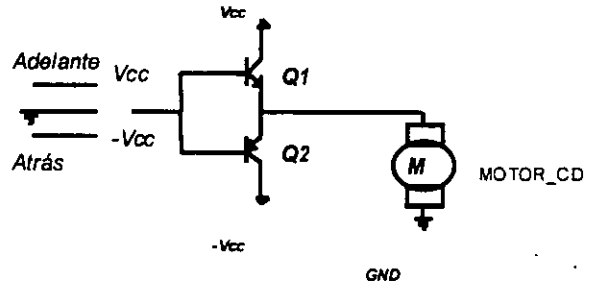


Figura 4-1. Configuración de tipo T, con dos transistores.

4.2.3 Configuración tipo H

Está compuesto de cuatro elementos, como se ejemplifica en la figura 4-2, si el switch S1 y S4 en la figura son cerrados mientras los switches S2 y S3 están abiertos, fluye la corriente de izquierda a derecha, en el caso contrario, si los switches S2 y S3 son cerrados y los switches S1 y S4 están abiertos, la corriente circulará de derecha a izquierda, funcionando el motor en sentido inverso.

La velocidad de conmutación se realiza controlando el switch S5.

En el puente H, los switches son abiertos y cerrados de una forma, que al polarizar un par de ellos, la corriente circulará en un sentido y en el otro caso al polarizar el otro par, circulara la corriente en sentido opuesto.

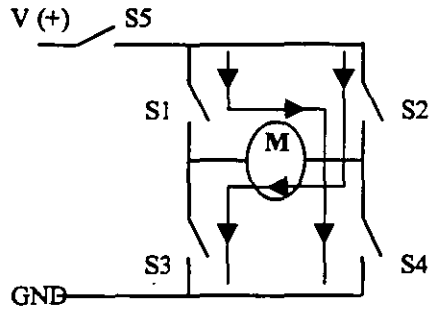


Figura 4-2. Configuración H con switches.

Emplea cuatro transistores de potencia y una sola fuente de alimentación, como se aprecia en la figura 4-3.

El funcionamiento de los transistores de la configuración H es tal, que siempre están conduciendo o bloqueados por parejas. Por ejemplo, si T1 y T4 conducen, se requiere que T2 y T3 no conduzcan y viceversa. Regulando el tiempo de conducción se varía la velocidad del motor.

En el puente H, los switches son abiertos y cerrados de una forma, que al polarizar un par de ellos, la corriente circulará en un sentido y en el otro caso al polarizar el otro par, circulara la corriente en sentido opuesto.

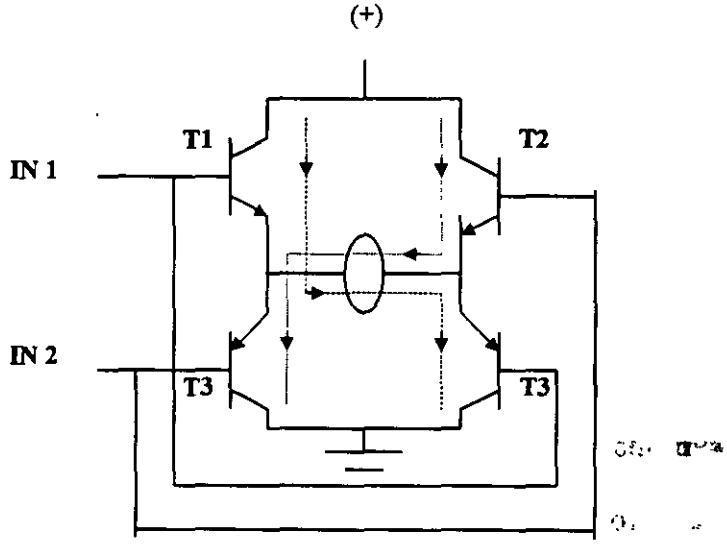


Figura 4-3. Configuración tipo H con cuatro transistores.

4.3 Métodos de control de potencia

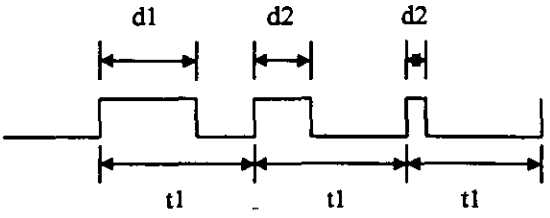
Se pueden implementar dos métodos para controlar la potencia:

1.- Modulación por ancho de pulso, PWM (Pulse Width Modulation);

2.- Modulación de la frecuencia de pulso, PFM (Pulse Frecuency Modulation).

4.3.1 Sistema PWM

Este sistema emplea, normalmente, una sola fuente de alimentación, para conseguir variaciones de frecuencia, el amplificador de conmutación se diseña para una frecuencia constante y sólo se cambia el ancho del pulso, como se aprecia en la figura 4-4.



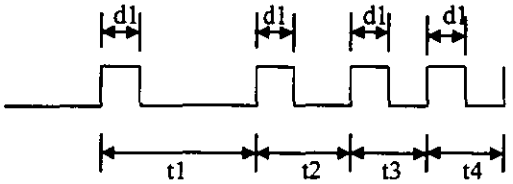
t1: constante
d1, d2, d3: diferente

Figura 4-4. Variación de ancho del pulso
Velocidad de ancho de pulso = ton/tperiodo

El amplificador conecta a la fuente de alimentación con una frecuencia fija, pero con un ángulo variable, de esta manera se consigue la tensión media más apropiada en la carga.

4.3.2 Sistema PFM

Emplea una ancho del pulso constante, pero varía la frecuencia. Los resultados obtenidos, empleando este método, son muy parecidos a los del PWM, según se muestra en la figura 4-5.



d1 : contante
t1, t2, t3, t4: diferente

Figura 4-5. Variación de la frecuencia, con ancho de pulso contante.

4.4 Circuitos integrados manejadores de potencia

Estos manejadores hacen muy practico la interfaz de los motores con el microcontrolador, ya que reducen sustancialmente el consumo de energía y el espacio que ocupan, además cuenta con circuitos limitadores de corriente y protección contra sobrevoltajes.

Existe un Chip fabricado por Motorola, el MPC1710A que se muestra en la figura 4-6, el cuál presenta la capacidad de poder controlar un motor de corriente directa, operando el último en ambos sentidos.

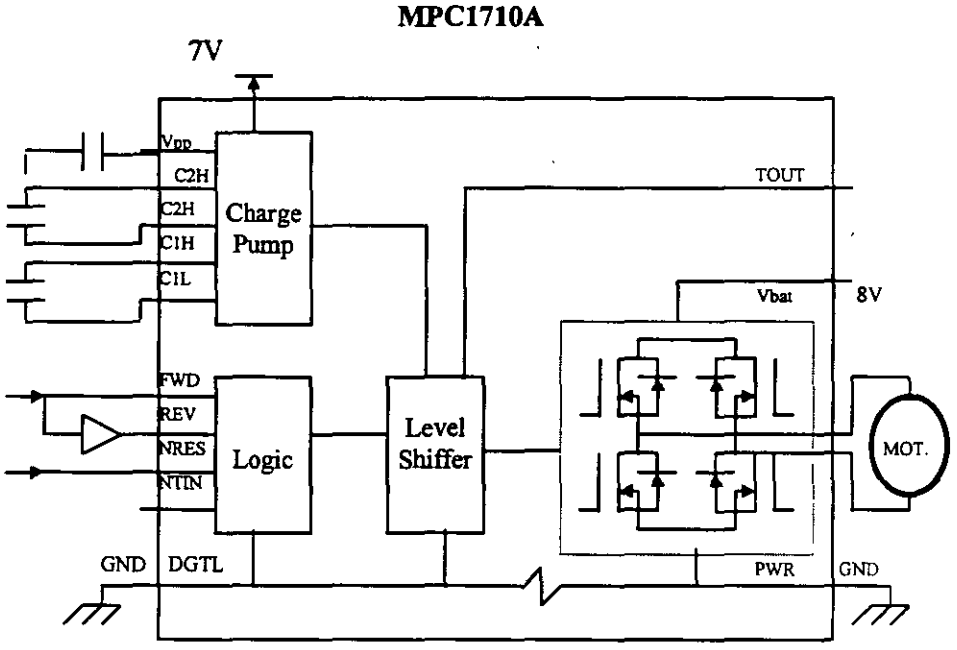


Figura 4-6. Circuito MPC1710

Por otro lado, existe el circuito, conocido como L293, que es un circuito integrado de 16 pines, fabricado por National, internamente cuenta con dos arreglos tipo H, se ilustra en la figura 4-7 el diagrama a bloques de una parte de este circuito.

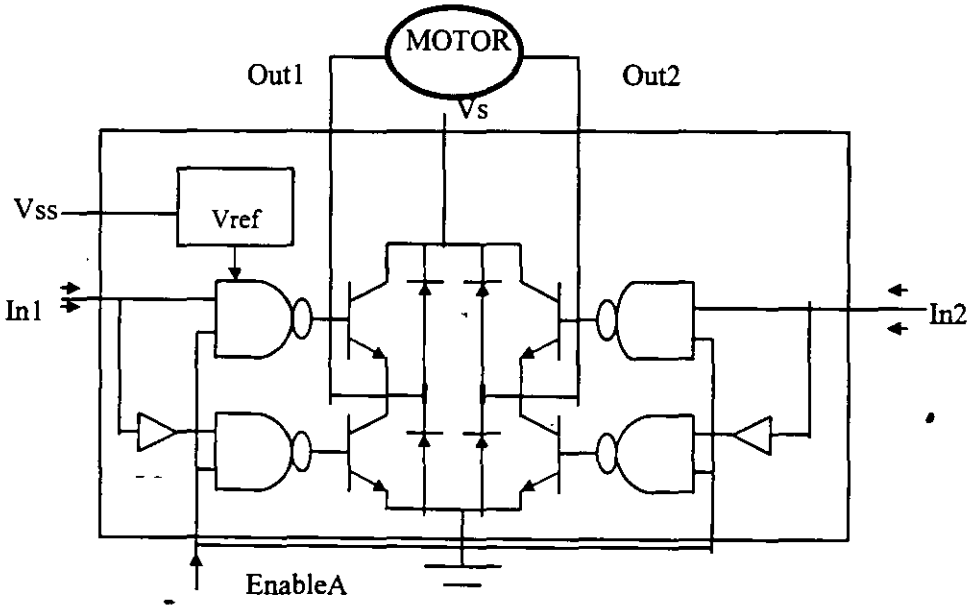


Figura 4-7. Circuito manejador de motor de CD, L293.

La ventaja de este circuito radica en que se pueden controlar por su arquitectura interna, dos motores al mismo tiempo girando en sentido horario y antihorario, o como se presenta en el apéndice B, se pueden controlar hasta cuatro motores al mismo tiempo, pero únicamente en un solo sentido; además este circuito opera con un mínimo de componentes externos, en este caso solo se conectan dos circuitos inversores; para asegurar el estado de corte y conducción de las parejas de transistores.

4.5 Motores de paso a paso

Los motores paso a paso o simplemente PAP, son un tipo especial de motores que permiten el avance de su eje en ángulos muy precisos y por pasos en las dos posibles direcciones de movimiento, izquierda y derecha. Aplicando a ellos una determinada secuencia de señales digitales, avanzan por pasos hacia un lado u otro y se detienen exactamente en una determinada posición.

Cada paso tiene un ángulo muy preciso, determinado por la construcción del motor, lo que permite realizar movimientos exactos sin necesidad de un sistema de control de lazo cerrado.

Los motores paso a paso presentan grandes ventajas con respecto a la utilización de los servomotores debido a que se pueden manejar digitalmente sin realimentación, su velocidad se puede controlar fácilmente, tienen una larga vida, son pequeños, robustos y poseen un elevado torque en bajas revoluciones, lo que permite un bajo consumo tanto en vacío como en plena carga, su mantenimiento es mínimo debido a que no tiene escobillas.

Debido a esta ventaja, tienen una gran variedad de aplicaciones dentro de las cuales se podrían mencionar las siguientes: máquinas herramientas, robots, impresoras para computadoras, graficadoras, máquinas de coser, unidades de disco, etc.

4.5.1 Funcionamiento

El funcionamiento de los motores paso a paso se basa en el simple principio de atracción y repulsión que ocurre entre los polos magnéticos. Como se sabe, un imán tiene dos polos llamados Norte y Sur. El principio básico del magnetismo establece que los polos iguales se repelen y los polos diferentes se atraen.

4.5.2 Tipos de motores paso a paso

Según su construcción, existen tres tipos de motores PAP:

a) De imán permanente

En este tipo de motor, su rotor es un imán permanente que posee una ranura en toda su longitud y el estator está formado por una serie de bobinas enrolladas alrededor de un núcleo o polo. Su funcionamiento se basa en el principio explicado anteriormente de atracción y repulsión de los polos magnéticos.

b) De reluctancia variable

En estos motores el rotor está fabricado por un cilindro de hierro dentado y el estator está formado por bobinas que crean los polos magnéticos. Como este tipo de motores no tiene un imán permanente, su rotor gira libremente cuando las bobinas no tienen corriente, lo que puede ser inconveniente en un momento dado si han una carga que presione el eje; estos motores puede trabajar a mayor velocidad que los anteriores.

c) Híbridos

Son motores de pasos, que incluyen algún mecanismo para amplificar el par o la potencia. Algunos emplean engranes para lograr pares, aunque reducen la amplitud de los pasos.

Los motores paso a paso, tanto unipolares como bipolares, pueden trabajar en dos modos de operación, de paso completo y de medio paso. En el primer caso, con cada secuencia el rotor gira un determinado ángulo que depende de la fabricación del motor. En el modo de medio paso, cada secuencia produce un giro en grados correspondiente a la mitad de su paso normal.

En la práctica, generalmente los motores PAP poseen cinco terminales, debido a que existe una terminal común, que es el punto medio de los devanados de las bobinas.

4.6 Sistemas de control para motores de pasos

Una buena regulación del movimiento de un motor de pasos exige un sistema de control apropiado, estando relacionados estos dos elementos; en el sistema de control se requiere de las señales que controle la velocidad, sentido de giro y la generación de la secuencia de polarización, como se muestra en la figura 4-8.

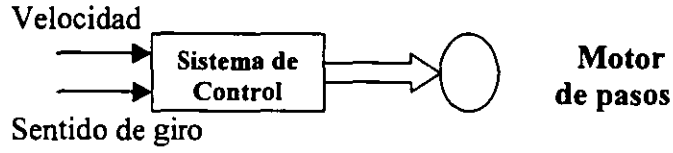


Figura 4-8. Esquema de sistema del control de un motor de pasos

CAPITULO 5
Lenguajes de programación

5.1 Lenguajes de programación de los microcontroladores

Para escribir un programa que va a ser ejecutado por un microprocesador existen básicamente tres alternativas: lenguaje de maquina, lenguaje ensamblador y lenguaje de alto nivel. Solo los programas escritos en lenguaje de maquina pueden ser ejecutados directamente por el microprocesador.

Los programas escritos en lenguaje de alto nivel tienen que ser traducidos primero a lenguaje de maquina a fin de que puedan ser ejecutados.

5.1.1 Lenguaje de máquina

Un programa en lenguaje de maquina es simplemente una secuencia de instrucciones y datos en código binario (unos y ceros), estos programas son por su naturaleza difíciles de leer, escribir, entender y corregir. Las computadoras únicamente trabajan en lenguaje de maquina, como cada instrucción está compuesta de un valor numérico, que componen el código de operación de la instrucción a realizar. Escribir directamente en lenguaje de maquina es un tanto difícil, ya que se substituyen los mnemonicos por sus valores numéricos.

5.1. 2 Lenguaje ensamblador

El lenguaje ensamblador utiliza mnemónicos para representar los códigos de operación de las instrucciones y símbolos alfanuméricos para representar las direcciones y los datos del programa.

La ventaja de utilizar nombres simbólicos para las direcciones, en lugar de un valor numérico, es que no solo facilita la escritura del programa, sino también simplifica la inserción o eliminación de instrucciones.

Cuando el programa está escrito en lenguaje ensamblador usando direcciones simbólicas, el añadir o quitar una instrucción no causa ningún problema por que al traducir el programa a lenguaje de maquina, automáticamente se hacen los ajustes necesarios.

La desventaja de los programas escritos en lenguaje ensamblador es que requieren un programa especial llamado ensamblador, que se encarga de traducir lo mnemónicos y los símbolos alfanuméricos a lenguaje de maquina; además sigue siendo necesario un conocimiento detallado de la arquitectura del microprocesador.

5.1.3 Lenguaje de alto nivel

Los lenguajes de alto nivel permiten la escritura del programa en una forma más relacionada con el problema a resolver que con las características del microprocesador; generalmente cada instrucción en lenguaje de alto nivel es equivalente a varias instrucciones en lenguaje ensamblador.

Sin duda, con los lenguajes de alto nivel el tiempo invertido en el diseño y codificación de un programa es mucho menor. Sin embargo, para que los programas puedan ser ejecutados se requiere un traductor que convierta las instrucciones de alto nivel en lenguaje de maquina. A este programa traductor se le conoce como COMPILADOR.

Otra ventaja de los programas escritos en lenguajes de alto nivel es su independencia respecto a un procesador en particular, lo único que se necesita es el sistema en el que se desea ejecutar el programa y se posea un compilador para el lenguaje en el que el programa está escrito.

Conociendo todas las ventajas de los lenguajes de alto nivel entonces se plantea la pregunta: ¿Por qué complicar las cosas utilizando ensamblador?. Existen varias razones para esto, una de ellas es que los compiladores son más complejos y por lo tanto requieren mayor cantidad de memoria y son más lentos que los ensambladores. Otra es que, dado el gran número de posibilidades que deben tomar en cuenta los compiladores, el código en lenguaje de maquina que genera tiende a ser ineficiente.

Los programas recientes, regularmente se escriben utilizando, el programa Edit de Msdos, o el editor de los lenguajes de programación, donde se crea el programa fuente, conteniendo instrucciones en Lenguaje Ensamblador y con saltos relativos; para que con este programa fuente, con ayuda de un ensamblador, se trasformen las instrucciones a lenguaje de maquina y genere los programas de tipo S19 y LST.

5.2 Formato de los programas S19 y LST

5.2.1 Archivos con formato S19

El formato S19 fue creado por motorola con el propósito de codificar programas o archivos de datos en un formato sencillo para la transferencia de datos entre sistemas de computadoras. El proceso de transportación puede ser entonces monitoreado visiblemente y los archivos en formato S19 formados por registros S pueden ser editados fácilmente. Los registros S son esencialmente cadenas constituidas por diferentes campos que permiten identificar el tipo de registro, la dirección de memoria, los códigos o datos y el checksum. Cada byte de datos binarios está codificado por dos caracteres que representan un número hexadecimal .

Los cinco campos que forman un registro S son:

TIPO	LONGITUD	DIRECCION	CODIGO/DATOS	CHECKSUN

Los campos están compuestos de la siguiente manera:

CAMPOS	NUMERO DE CARACTERES	DESCRIPCION
TIPO	2	Tipo de registro S, S0, S1, S9, etc.
LONGITUD	2	Indica el número de pares de caracteres en el registro, excluyendo el tipo y longitud.
DIRECCION	4, 6 u 8	Los 2, 3, o 4 bytes de dirección que indican la dirección de memoria en donde se cargará en campos de datos.
CODIGO/DATOS	0 - 2_N	De 0 a n bytes de código ejecutable, datos cargables en memoria, o información descriptiva.
CHECKSUM	2	El byte menos significativo del complemento a uno de la suma de los valores representados por los pares de caracteres, tomando en cuenta los campos de longitud del registro, el de dirección y el de código/datos.

Ejemplo de un archivo S19:

```
S11301008E01FFCE10001C39901C3030011F30804E
S11130110FCB61032B110332B12B61033B110322B9F
S113012014B61034B110322B1620E1B61032B110CF
S1130130342B202A14B61033B110342B1C2A0AB6DF
S11301401034B110332B1820C3B61034B110322B35
S1130150EE20B9C601E70420B3C602E70420ADC609
S1008016004E70420A7E0
S9030000FC
```

5.2.2 Archivos con formato LST

El archivo de tipo lista (LST), es generado cuando el programa fuente se ha ensamblado; este archivo muestra los mnemónicos y códigos de maquina de cada instrucción.

Una parte de un archivo de tipo lista LST, es el siguiente:

```
0180                          *INICIO DEL PROGRAMA*
0181
0182
0183  2000          Inicio    org $2000          Dirección de inicio
0184  2000                    sect 0
0185          - -
0186
0187  2000  8e7ff0            LDS #STACK         Inicializa el SP.
0188
0189  2003  ce10 00       LDX #REGBAS      x <- 1000
0190  2006  8625               LDAA  #$25
0191  2008  a73c               STAA HPRIO,X       Selecciona modo expandido
0192
0193  200a  0f                 SEI                Deshabilita interrupciones
```

```
0194
0195                    * inicia el programa principal
0196   200b   bd24c5           jsr _main
0197
0198   200e   7ee00a wait       jmp $e00a          Salta al origen de buffalo
```

A continuación se explican cada una de sus partes:

a) Numero de Línea (Columna 1)

Es el número progresivo de las líneas de programación, el numero de línea esta dado en un valor decimal, y el numero 65536 es el mayor posible de cualquier programa, es de gran ayuda ya que en caso de error, el ensamblador indica el numero de línea en el que se encuentra este.

b) Dirección (Columna 2)

La dirección corresponde a la localidad de memoria destinada a cada instrucción, este valor está en notación hexadecimal.

c) Código Objeto (Columna 3)

Cada instrucción es transformada a lenguaje de maquina y los códigos se muestran en su valor hexadecimal, estas instrucciones pueden ser de 1, 2, o 3 Bytes de longitud dependiendo de la instrucción y modo de direccionamiento involucrado.

d) Etiquetas (Columna 4)

Las etiquetas son usadas en el programa fuente, para indicar la dirección de inicio de una subrutina, de un salto relativo o la localización de una constante.

e) Operadores (Columna 5)

Se indica las instrucciones en lenguaje ensamblador.

f) Operandos (Columna 6)

La columna de operandos puede contener un valor, una dirección o una etiqueta.

g) Comentarios (Columna 7)

Los comentarios ayudan para explicar brevemente que es lo que realiza cada línea de un programa, estos comentarios no generan códigos de maquina, son copiados directamente del programa fuente.

5.3 Procedimiento para ensamblar un programa

Existen varios programas ensambladores, como por ejemplo el AS11, IASM11, etc.

Para el primer caso, es necesario editar los programas utilizando lo mnemónicos, para generar un programa que se tenga la extensión ASM o ASC para que desde el sistema operativo se indica lo siguiente.

AS11 Nombre_programa.extensión

Entonces se genera el programa con el mismo nombre que el fuente, pero con formato S19.

Por otro lado, al emplear el ensamblador IASM11, tiene la ventaja de contar con un editor de texto, integrado con un sistema de comandos para configuración. La manera de obtener un archivo S19 se numera a continuación.

1.- Se teclea IASM11 Nom_prog;
2.- Editar el programa;
3.- Presionar la tecla F10 para pasar a otro sistema de comandos;
4.- Configurar la salida del archivo, seleccionando en ASSAMBLE el formato deseado;
5.- Para ensamblar, con la tecla de función F4;
6.- El ensamblador creará los formatos seleccionados, S19, LST, HEX, etc.

5.4 Compiladores

Los compiladores son una herramienta para programar una aplicación en un microcontrolador. Un Compilador es un programa similar a un ensamblador, que transfiere instrucciones en lenguaje de alto nivel a códigos en lenguaje de maquina; existen compiladores en Basic, Fortran, Pascal, y otros que por mucho tiempo han sido utilizadas en computadoras. En años recientes, el Lenguaje C se ha convertido en uno de los más populares.

Los compiladores son escritos para un lenguaje de programación especifico y para una unidad en particular. Los compiladores de C son escritos para que puedan operar con IBM-PCs, Apple, Macintoshes y muchos otras computadoras que usan sistema UNIX. La ventaja de los compiladores es su facilidad de poder hacer modificaciones al programa fuente, de una manera rápida y fácil.

Tal vez el inconveniente principal del compilador de C, es que genera programas más grandes que con el ensamblador, y en algunas ocasiones no produce un código eficiente como con el lenguaje ensamblador, pero en la mayoría de las ocasiones el C puede ser utilizado sin ningún problema.

La programación en C se ha convertido indiscutiblemente en una poderosa y popular herramienta de desarrollo para el ingeniero diseñador. Por esta razón, como parte de la extensa gama de herramientas de desarrollo con microprocesadores, cotidianamente se lanzan al mercado ***compiladores cruzados en C*** que permiten la programación en lenguaje de alto nivel.

5.4.1 Compilador cruzado para el HC11

El compilador cruzado para el MC68HC11 permite traducir el programa de aplicación de lenguaje C, en código de ensamblador para los microcontroladores MC68HC11. Este compilador cruzado consiste de lo programas siguientes: un programa manejador (ICC11.EXE), un preprocesador de C (ICPP.exe) y un compilador (ICCOM.EXE); como se muestra en el diagrama de la figura 5-1.

El compilador cruzado es un sistema compilador tradicional que acepta y se ajusta al lenguaje ANSII. El compilador cruzado invoca al programa manejador (ICC11.exe), a las funciones necesarias y al archivo que contiene las rutinas básicas de ensamblador (BASICS.S) para poder procesar el archivo en C (ARCHIVO.C).

El archivo BASICS.S contiene también el inicio del programa en donde se inicializa el apuntador de pila y se hace el llamado del *main()*, que es la rutina que se ejecuta al principio de un programa en C.

Si el compilador cruzado no detecta ningún error, produce un archivo objeto, que contiene el código en ensamblador (ARCHIVO.S). Si existió un error, se indica donde se encuentra y de que tipo se trata.

El programa manejador (ICC11.EXE) direcciona al preprocesador (ICPP.EXE) para que busque en los directorios especificados los encabezados de los archivos que contienen las funciones que se han desarrollado en C y que son necesarios para generar el archivo con el código en ensamblador.

El preprocesador (ICPP.EXE) agrega al archivo en C las funciones necesarias (headers files), generando un archivo en lenguaje de alto nivel C (ARCHIVO.I).

Finalmente el compilador (ICCOM.EXE) traduce el código del archivo en lenguaje de alto nivel en un código fuente para el ensamblador.

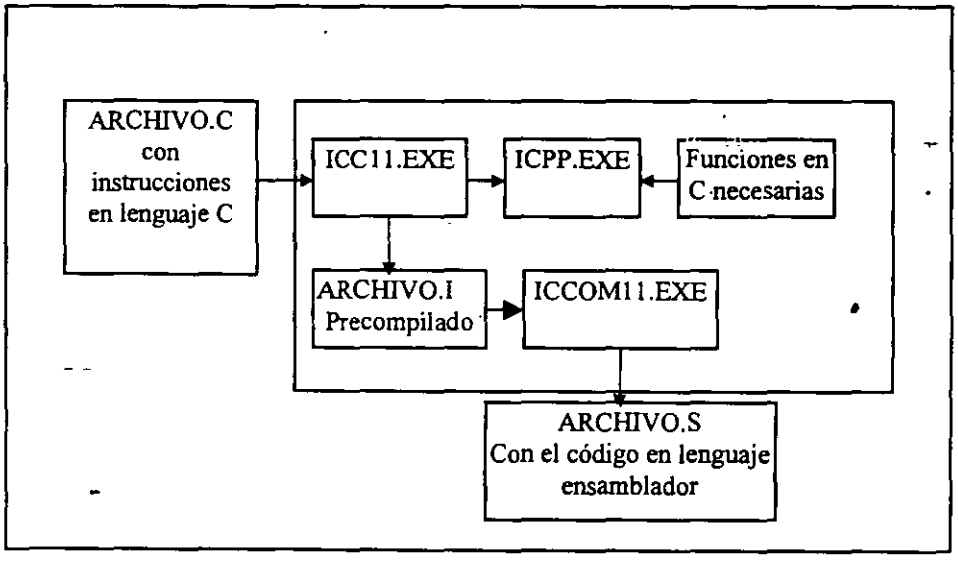


Figura 5-1. Compilador cruzado para el MC68HC11.

5.5 Ejecución de un programa

La manera de programar, generar y correr una aplicación en el microcontrolador 68HC11 es la siguiente:

1.- Se crea el programa fuente en C, utilizando el editor de textos de su preferencia.

2.- Se salva el programa utilizando la extensión C.

3.- Empleando el programa CC6811.bat, que realiza la liga entre el compilador, ensamblador y programas con rutinas y declaraciones en lenguaje ensamblador, se generan los programas con formato s, i,S19 y LST.

Se ejecuta de la siguiente manera:

CC6811 Nombre_programa

El archivo CC6811.bat contiene lo siguiente:

```
Icpp.exe –DHC11 –D__LCC__ -I/icc11/include %1.c %1.i
Iccom11.exe –v %1.i %1.s
Ias11 –o %1 –l basics.s %1.s
```

4.- Una vez teniendo el programa con el formato S19, está listo para ejecutarse.

5.- Con el PCBUG11, que es un programa que permite tener el control de nuestra unidad, se puede cargar, correr, monitorear el estado de los registros, manipulación de datos en memoria, etc.

6.- Se cuenta con otro archivo, EPROM.bat que realiza la tarea de configuración y selección del puerto de comunicación serial donde se conectó el microcontrolador, así mismo de cargar un macro que configura el modo de operación, para trabajar en expandido; el contenido del archivo EPROM es el siguiente:

PCBUG11 -E PORT= (1/2) MACRO=EPROM

7.- Una vez en el sistema de PCBUG11, se carga el programa con la siguiente instrucción:

LOADS Nombre_programa

8.- Se manda a ejecutar el programa:

G Dirección_inicio

9.- El sistema con la aplicación seleccionada comenzará a ejecutarse.

En el apéndice B se proporcionará mayor información.

Capitulo 6
Construcción del Robot Móvil

Este capítulo presenta los circuitos, que se obtuvieron como resultado de las pruebas y experimentos , con el objetivo de llegar al diseño final del robot móvil, para cubrir los requerimientos del diseño planteados al principio.

6.1 Fuente de alimentación

La alimentación de todos los circuitos se realizan con dos baterías de NiCd, que suministran 9.6 Volts a 500mAh, empleados regularmente en carros de radio control. Se reduce su voltaje nominal con circuitos reguladores de la familia 7805 y 7806, con la finalidad de obtener el voltaje de operación óptimo para cada sección del robot.

En la figura 6-1 se presenta el diseño simple de una fuente de alimentación de 5 volts, compuesto de dos capacitores que eliminan el rizo existente en la fuente principal; mismo que va conectada al circuito LM7805, que regula el voltaje de entrada a 5 V, cuya corriente entregada puede variar desde 300 mA hasta 10 Amperes, dependiendo de la versión de este circuito.

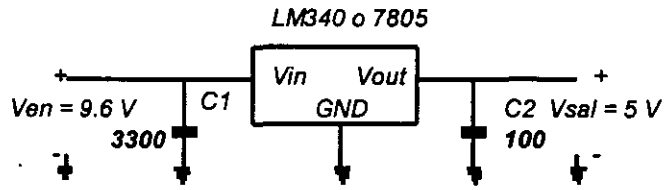


Figura 6-1. Fuente de 5 Volts para la alimentación del microcontrolador.

Para la figura 6-2 se muestra el diseño de la fuente que proporciona el voltaje tanto de los motores, como de los circuitos requeridos para esta etapa. La finalidad de emplear fuentes de alimentación distintas, es tener aisladas la parte de control que envía el HC11 y los circuitos propios del control de los motores.

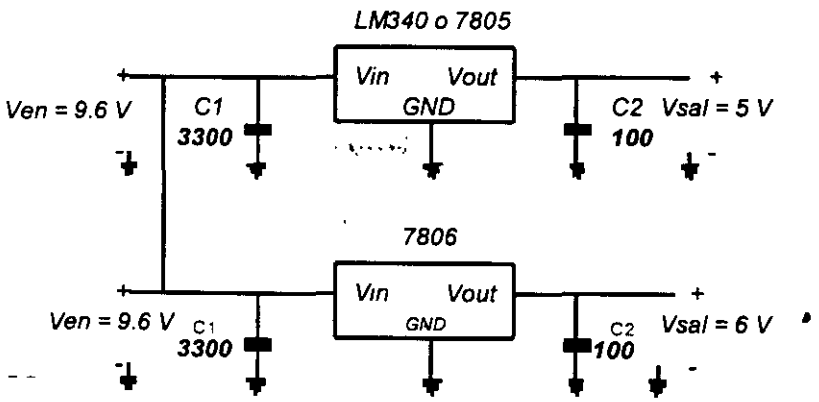


Figura 6-2. Fuente de 5 y 6 Volts para alimentación de los motores.

6.2 Adaptación del robot móvil

Las dos posibles opciones que se consideraron para la estructura del robot móvil, en la distribución de los motores y llantas son:

a) **Un motor en la parte tracera**; que controla el desplazamiento del robot, en cuyo eje se encontrarán las dos llantas y un motor en la parte delantera, conectado a una rueda giratoria, de tal forma que controle la dirección de giro del robot, como se muestra en la figura 6-3; estos motores pueden ser motores de corriente directa o motores de paso, para nuestro caso se emplearon motores de CD.

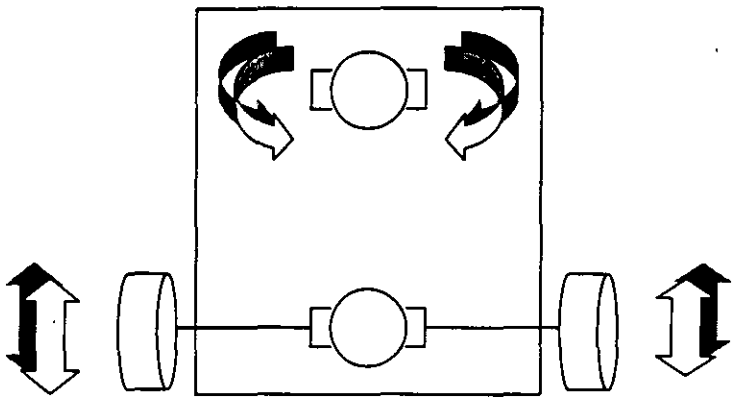
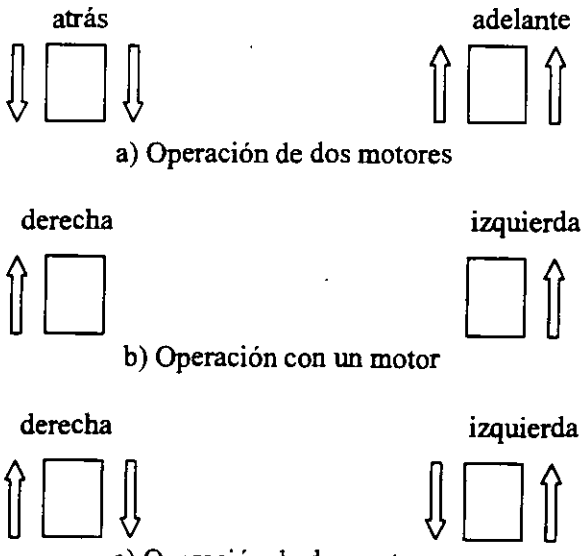


Figura 6-3. Esquema de un motor atrás - un motor adelante.

b) **Dos motores en la parte tracera**; conectando una llanta en cada uno de los motores y una rueda de movimiento libre en la parte delantera; el control de desplazamiento y de dirección lo realizan los dos motores; de manera que para hacer avanzar al robot hacia adelante y hacia atrás se hacen girar ambos motores en el mismo sentido, y en el caso del giro a la derecha e izquierda, se realiza haciendo funcionar los motores en sentido opuesto, donde el motor que gire hacia atrás será el que mandará en el movimiento del robot, los posibles movimientos se presentan en los esquemas contenidos en la fig. 6-4.



a) Operación de dos motores

b) Operación con un motor

c) Operación de dos motores

Figura 6-4 a, b, c. Posibles movimientos del robot.

La figura 6-5 presenta la implantación para este tipo de arreglo.

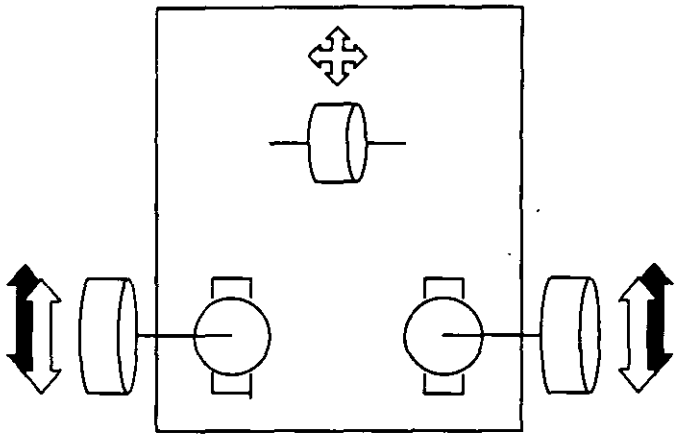


Figura 6-5. Esquema con dos motores en la parte tracera.

6.3 Control de los motores de corriente directa

Para controlar los motores se utilizó el circuito integrado L293D, que como se analizó en el capítulo cuatro, nos presenta la capacidad de controlar los dos motores, lo que nos ahorra el agregar una cantidad mayor de dispositivos electrónicos y como consecuencia reduce mucho el espacio ocupado para esta sección.

Como se muestra en la figura 6-6, y de acuerdo a las características de este circuito, se requieren de cuando menos cuatro señales de control, las cuales serán otorgadas por el microcontrolador. Para controlar un motor se requiere de una señal que entregará el comando de dirección del motor, es decir; hacia donde deseamos que gire (derecha o izquierda), y otra señal que nos proporcionará la velocidad de giro del motor. Para el control del otro motor, se necesita de la misma cantidad de señales.

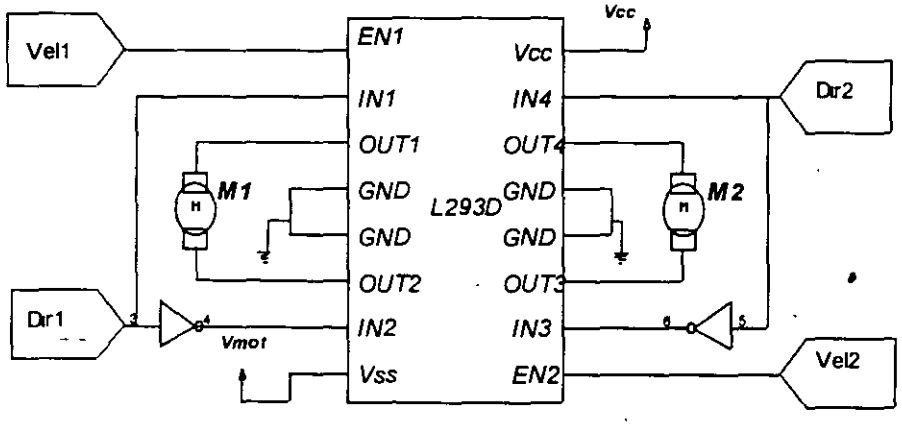


Figura 6-6. Circuito de control de los motores de CD.

El diseño más apropiado que se determinó emplear para controlar los motores, es el que se presenta en la figura 6-7, muestra el agregado de una etapa de acoplamiento y aislamiento, entre las señales que provienen del microcontrolador y la etapa de control de los motores; para tener este tipo de control se requieren de seis señales, cuatro para la dirección y dos para la velocidad de giro de los motores.

Así mismo, otra ventaja que presenta este circuito es contar con un pin asignado exclusivamente para suministrar el voltaje de polarización de los motores, el cuál puede tener un valor entre 0.2 hasta 32 volts, como se indica en la hoja de especificaciones del L293D en el apéndice C.

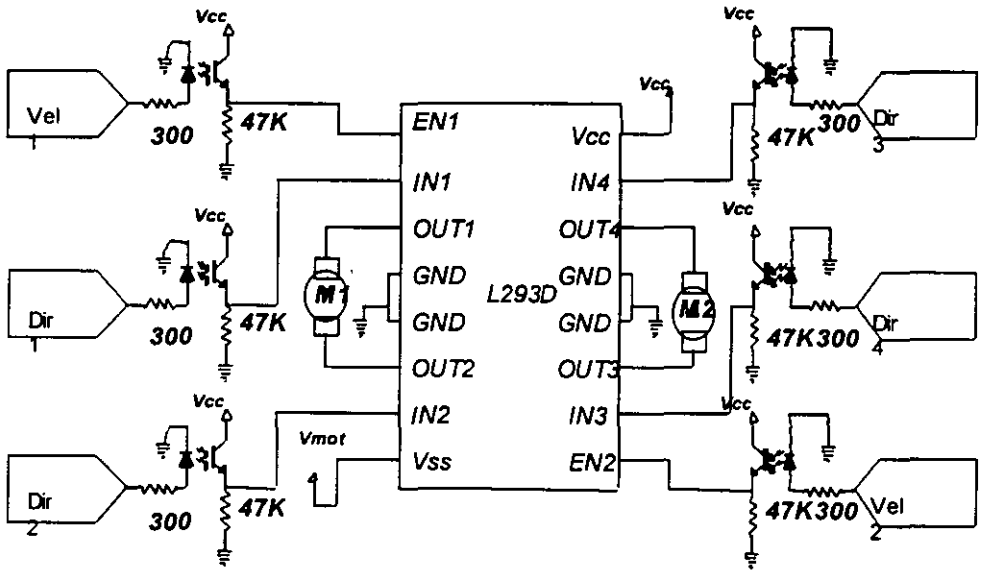


Figura 6-7. Circuito de control de los motores empleando optoacopladores.

6.4 Adaptación de sensores fotoeléctricos

La función que se busca con este tipo de sensores es interactuar con el medio ambiente, de acuerdo a la intensidad de luz que incide en ellos, el objetivo que se persigue, es que el robot realice acciones programadas, de acuerdo a la lectura que se encuentren en ellos.

a) Fotoresistencias

Las fotoresistencias son los sensores fotoeléctricos mas sencillos que existen, empleandos con una resistencia en serie se obtendrá un divisor de voltaje, cuyo valor será interpretado por el microcontrolador.

En el caso del HC11, se conectará al convertidor analógico digital; con este arreglo se obtendrán 256 posibles valores, mismos que nos tendrán informados del comportamiento del robot, con respecto a las fuentes luminosas que se encuentran en su entorno. Existen dos maneras de procesar esta información, una de ellas es obteniendo directamente el valor de la conversión digital (figura 6-8), que nos ayudaría si se desea programar con lógica difusa, y la otra manera será empleando a la salida del divisor de voltaje como una de las entradas de un comparador de voltaje, para obtener un nivel lógico (figura 6-9).

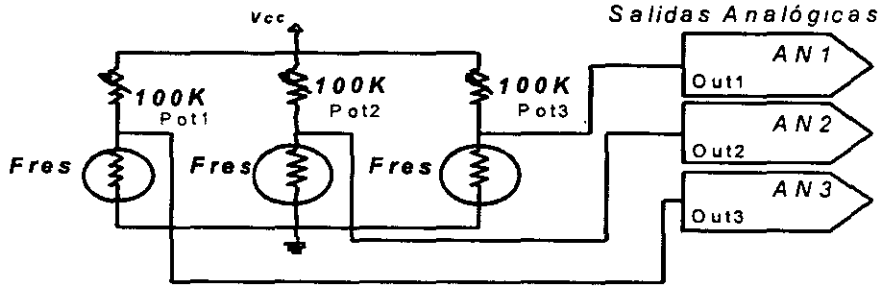


Figura 6-8. Empleo de fotoresistencias, obteniendo a su salida un valor analógico.

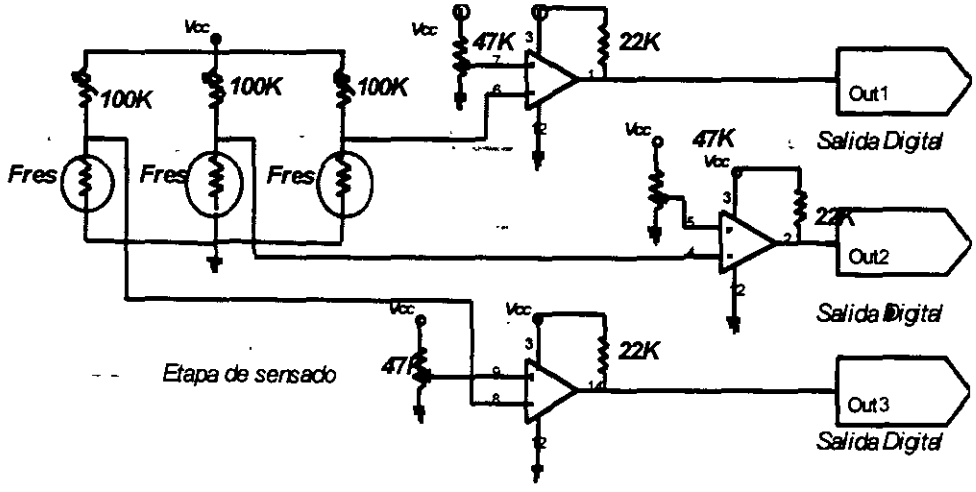


Figura 6-9.Circuito de interfaz con fotoresistencias, obteniendo una salida digital.

b) Fotoreflectores

Empleando los sensores de efecto fotoreflectivo, se aprovechan dos de las regiones de operación de los transistores, corte y saturación como se muestra en la figura 6-10.

Este tipo de sensores están compuestos de un led infrarrojo y un fototransistor, la interfaz con el microcontrolador se realiza en forma similar que las fotoresistencias.

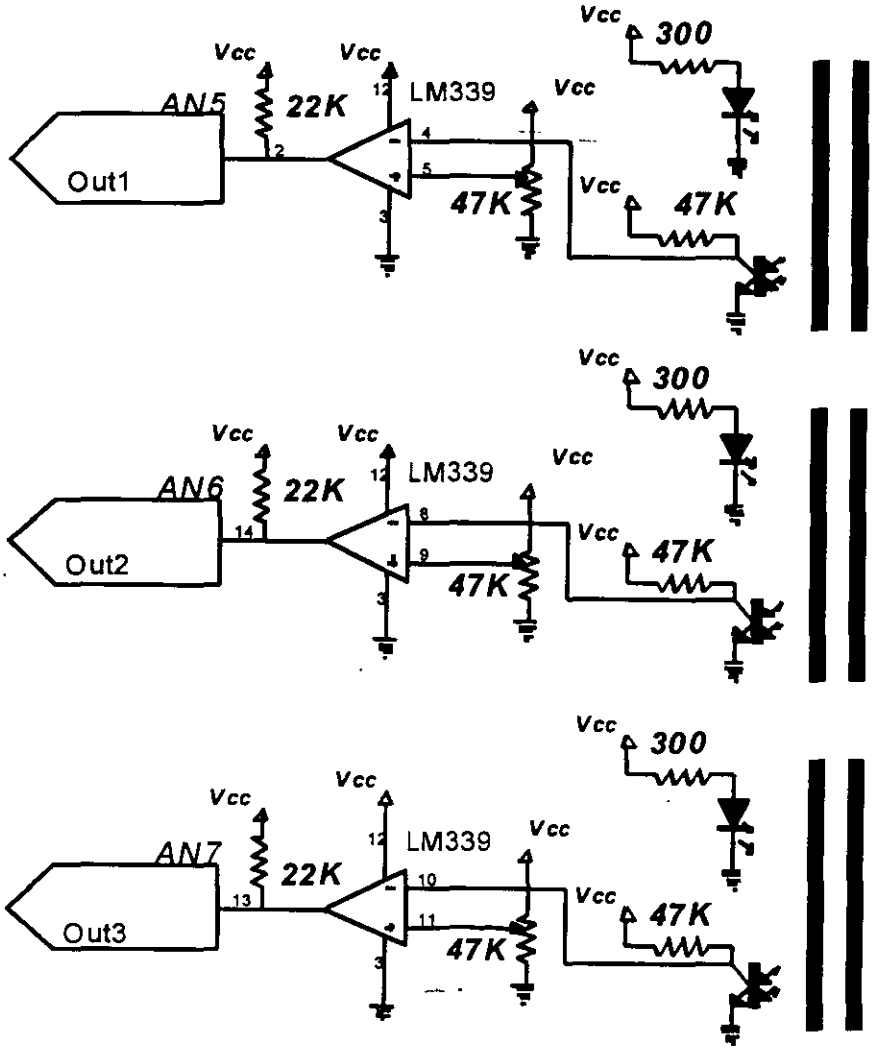


Figura 6-10. Interfaz con sensores de efecto fotoreflectivo.

6.5 Detección de obstáculos

Para la detección de obstáculos se emplean dos tipos de transductores, el primero de ellos utilizando el circuito detector de luz infrarroja GP1U52X, cuyo objetivo es detectar a distancia la presencia de un objeto, por lo tanto no tendrá contacto físico con el. En el segundo caso, cuando el robot tiene contacto físico con el obstáculo se emplearán microswitches.

a) Interfaz empleando microswitches

El uso de los microswitches facilita su adaptación para cualquier sistema de detección, en el caso del contacto directo, la interfaz se realiza agregando una resistencia en serie con este sensor, con la finalidad de abrir o cerrar el circuito, con lo cuál permitirá la circulación de corriente; según la figura 6-11, el voltaje de salida siempre estará entre los niveles de polarización y tierra, es decir 0 y 5 Volts.

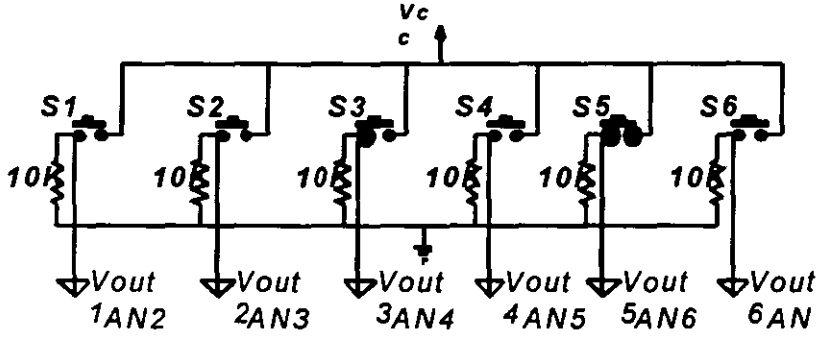


Figura 6-11. Detección de obstáculos utilizando microswitches.

Se pueden conectar tantos microswitches como se deseen, para obtener la información más precisa del entorno que rodea al robot, en nuestro caso se emplearon un total de seis sensores, como se muestra en la figura 6-12.

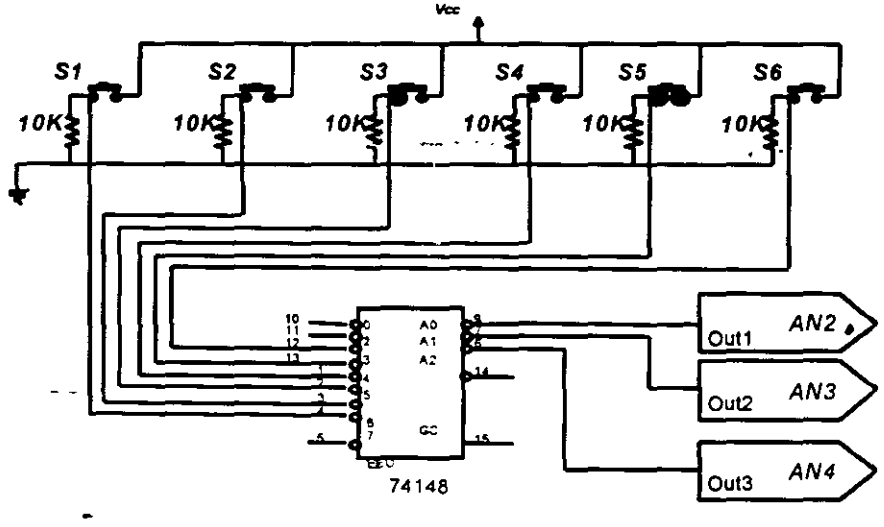


Figura 6-12. Detección de obstáculos empleando microswitches.

b) Interfaz infrarroja

Como se explica en el capítulo 3 y en el apéndice D esta interfaz se realiza generando una señal que oscile en el rango de los 40 Khz, que es la frecuencia de detección del módulo empleado para este fin (GP1U52X).

La frecuencia de oscilación se genera con el circuito integrado temporizador NE555, en modo de operación astable como se muestra en la figura 6-13; en el apéndice E se dará una breve descripción de este circuito integrado y de sus modos de operación; la distancia mínima de detección está controlada por el potenciómetro que se incluye en el circuito final.

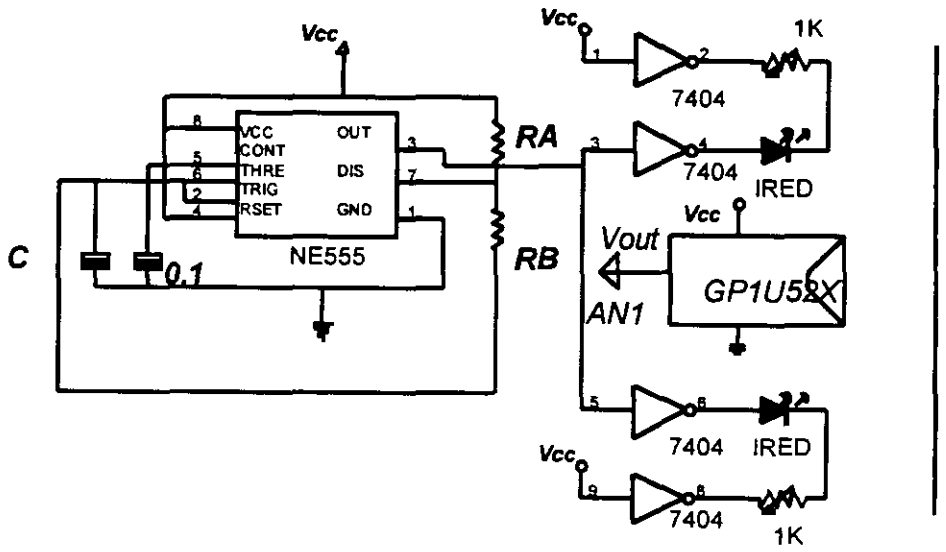


Figura 6-13. Interfaz infrarroja.

Para obtener los valores de las resistencias se empleo la siguiente ecuación, misma que se detalla en el apéndice E:

$$f = 0.695(RA+2RB)C1$$

Si se fijan los valores de C = 4.7pF y de RA = 4.7 KOhms; se despeja la ecuación, para obtener la siguiente:

$$RB = (T/0.695C - RA) ½$$

Sustituyendo valores, se obtiene el valor de la otra resistencia:

$$RB = 1476.72 \text{ ohms.}$$

6.6 Control de posición y orientación

Se describió en el capitulo de sensores, la forma de operación de los distintos dispositivos empleados para controlar el desplazamiento del robot. Se adaptaron por su facilidad de uso los fotoreflectores, pero se podrían emplear indistintamente los interruptores ópticos o los magnéticos. Se presentan en la figura 6-14 a, b y c los circuitos adaptados para tener el control de la distancia recorrida a sí como de la velocidad del robot móvil.

a) Interruptor óptico

Un interruptor óptico esta compuesto por un led emisor de luz infrarrojo y un fototransistor; dispositivos externos únicamente requiere de dos resistencias, una conectado al infrarrojo para limitar la corriente y otra resistencia de pull up conectada en el colector del fototransistor.

Existe una pequeña ranura que separa a los dos dispositivos, donde se introduce un disco ranurado o perforado, con la intención de cortar o permitir que el haz infrarrojo incida en el fototransistor.

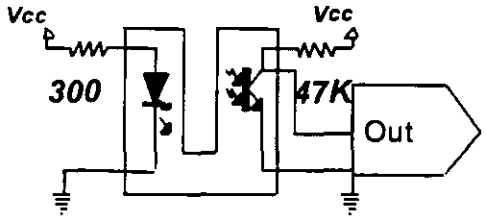


Figura 6-a. Control con un foto interruptor.

b) Fotoreflector

El compone de los mismos elementos que el interruptor óptico, tanto en lo interno, como en lo externo, pero colocados de distinta forma; se emplea el efecto de reflexión del haz infrarrojo; mismo que será capturado por el fototransistor, cortando o saturando a este último, dependiendo si existe reflexión.

En este caso se conecta frente a este sensor un disco con superficie blanca y rayas negras, con tantas como se desee; ya que entre mayor sea la cantidad de rayas se podrá tener un mejor control del movimiento y comportamiento del robot.

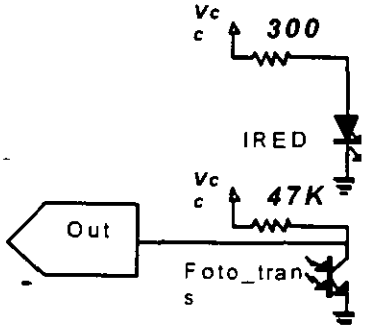


Figura 6-14b. Control con un fotoreflector.

c) Switch magnético

Esta interfaz está compuesto de un imán y un switch que se cierra al pasar el primero por el segundo; se requiere de una resistencia en serie con el switch. La única desventaja que presenta es que se deben de colocar una cantidad grande de pequeños imanes alrededor de diámetro de la llanta, para tener un control adecuado de los movimientos del robot.

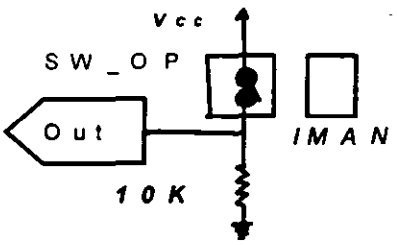


Figura 6-14c. Control con un interruptor magnético.

CAPITULO 7
Control y aplicaciones del Robot Móvil

Existe un sin fin de aplicaciones que se puede ejecutar con el Robot Móvil desarrollado en el presente trabajo, ya que tiene la facilidad de cambiar de una tarea a otra sin mucho problema y realizando pocas modificaciones. El uso del robot es de relativa sencillez, solo es necesario cargar el programa que se desea; el procedimiento para ejecutar la aplicación, es el mismo para todos los casos; se encuentra descrito en el capítulo 5.

Cuando se desarrollan las etapas de prueba, los programas son almacenados en la memoria RAM externa del microcontrolador, por lo que es posible hacer las modificaciones y compilar el programa tantas veces como sea necesario, hasta que se tenga una versión confiable y a la entera satisfacción del operador, para posteriormente ser programado en una memoria de tipo EPROM.

Las tareas y aplicaciones pueden ser tan sencillas o complicadas como se deseen; en este capítulo se presentan algunas aplicaciones probadas con el robot móvil; en el caso de requerir una tarea diferente a las mostradas, el robot estaría en condiciones de ejecutarla; únicamente se colocan las interfaces necesarias, así como realizar los programas para esa aplicación, en el apéndice G se presenta el código en lenguaje C, para algunas de las siguientes aplicaciones.

7.1 Control desde una computadora

La manipulación general del robot se realiza desde una computadora, controlando los movimientos y acciones deseados; de esta manera se podrá desplazar al robot, hacer que gire determinado ángulo, tomar la lectura de uno o varios canales del convertidor analógico digital del microcontrolador HC11.

Para esta tarea, además del programa previamente compilado y ensamblado, se requiere de otro programa escrito en lenguaje de alto nivel, que sea capaz de transmitir y recibir información de una terminal a otra, en nuestro caso se emplea el programa ejecutable de nombre RS232.

Existen dos maneras de realizar el control; una de ellas es empleando el cable de interface RS232 directa entre la PC y robot, la segunda acción es ocupando un par de radios RS232, transmisor y receptor, para evitar el uso del cable y contar con transmisión de radio frecuencia. El programa del robot contiene los siguientes comandos:

Acción	Comando
Avanzar hacia delante	**ad**
Avanzar hacia atrás	**aa**
Girar hacia la derecha	**gi**
Girar hacia la izquierda	**gd**
Lectura de los ocho canales del con A/D	**an**

7.2 Sigue un camino marcado

La tarea programada a realizar por el Robot Móvil, es seguir una trayectoria marcada sobre el piso, preferentemente pista blanca y fondo negro, o viceversa; aunque es posible realizar esta tarea con características distintas a las anteriores, solo es necesario calibrar y programar los valores que arrojan los sensores empleados en esta aplicación (figura 7-1).

El caminos a seguir puede tener diferentes características, como son caminar sobre línea continua, avanzar sobre una línea discontinua, contar con puntos de cruce o con bifurcaciones; avanzar sobre las diferentes condiciones requiere en su mayoría de un algoritmo propio, tomando como base la lectura de los sensores y las acciones a realizar por el robot.

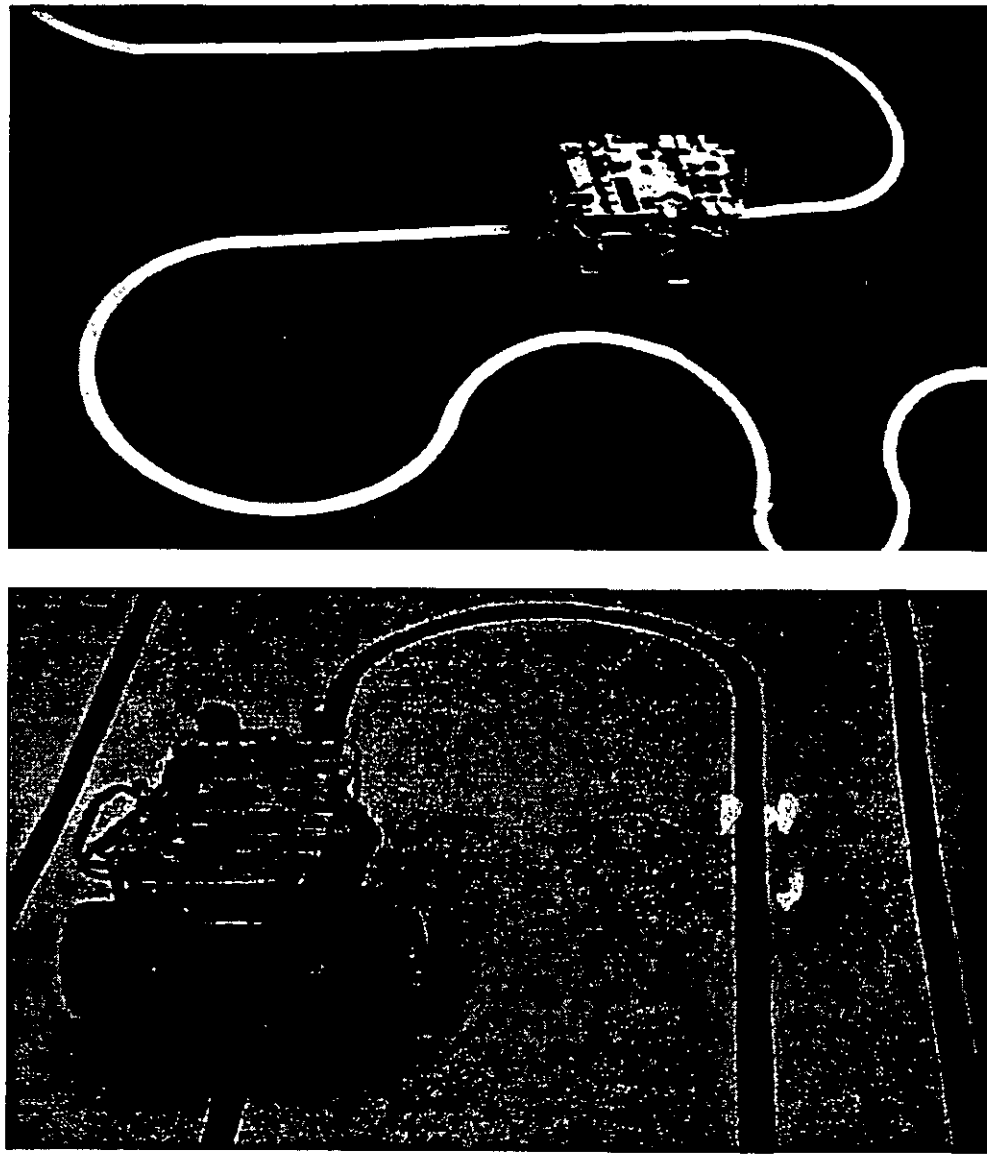


Figura 7-1. El robot sigue la trayectoria marcada en superficies de color distinto.

7.3 Desplazamiento del robot evitando obstáculos

Es posible realizar esta tarea de dos maneras distintas:

a) Detección de obstáculos empleando sensores de contacto;

b) Detección de obstáculos empleando interface infrarroja.

Para el primer caso, lo que se pretende es enviar al robot hacia un punto destino, haciendo el desplazamiento en el entorno de robot, teniendo contacto directo con los objetos, al producirse este, el robot realizará una acción previamente programada, como puede ser retroceder o girar determinado ángulo, para después continuar avanzando. En el segundo caso, la detección se realiza empleando los sensores infrarrojos, con los cuales es posible evitar los obstáculos y no tener contacto directo con ellos (figura 7-2).

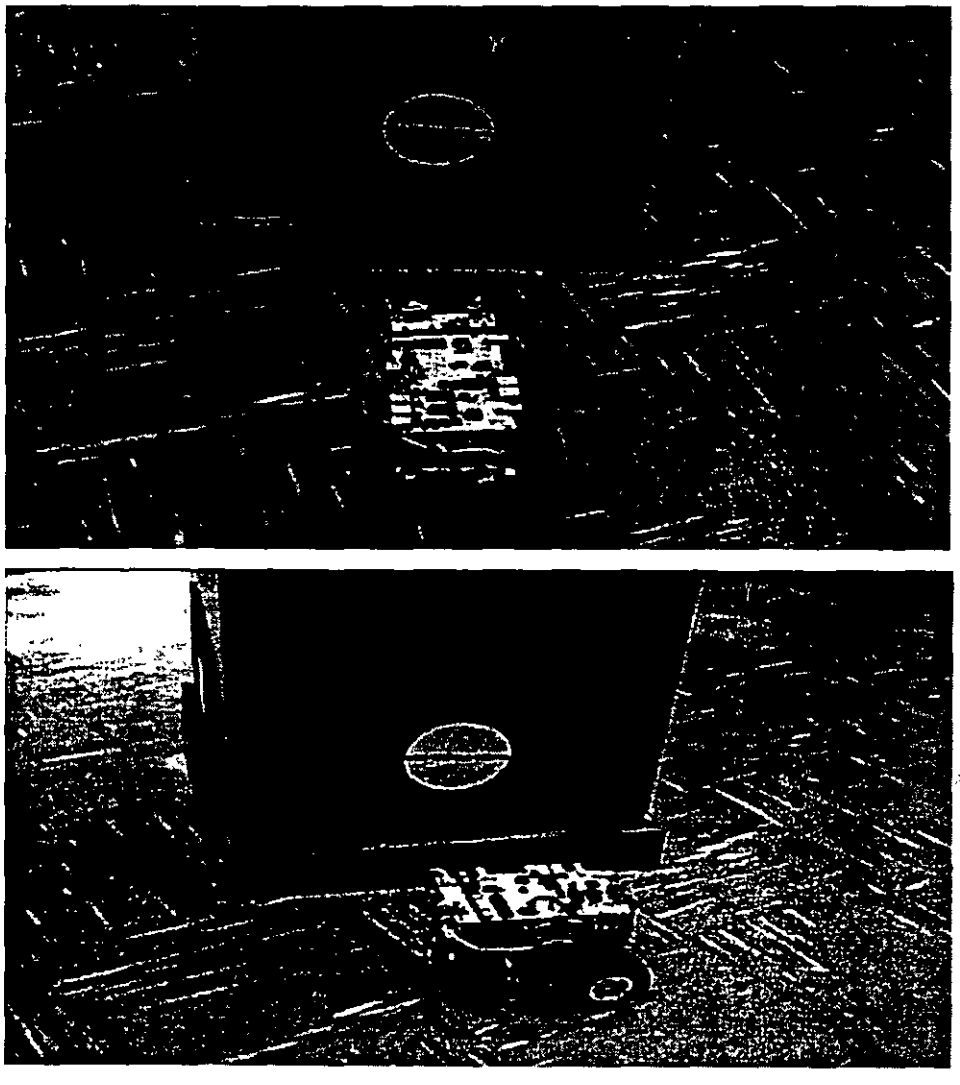


Figura 7-2. El robot detecta el objeto (figura superior), entonces realiza la acción de evitar el contacto (figura de abajo).

7.4 Movimiento del robot para realizar mapas

Se emplean las mismas funciones que el apartado anterior, se deja navegando al robot, de manera que obtenga la ubicación de los distintos objetos y elementos dentro de una oficina o habitación; se tendrá registrado la distancia, así como el ángulo de giro con el que se desplaza. La obtención de esta información se realiza transmitiendo los datos desde el robot hacia la PC, misma que se encontrará ejecutando un programa capaz de capturar y almacenar la información.

Una aplicación similar, es colocar al robot en un laberinto, de tal forma que encuentre la salida; el primer recorrido servirá de reconocimiento para encontrar la trayectoria correcta; en la segunda ocasión deberá realizar el recorrido en menos tiempo que el de reconocimiento y con el menor número de fallas.

7.5 Desplazamiento en áreas peligrosas o inaccesibles

Se controla el movimiento del robot desde una terminal, empleando una pequeña cámara de vídeo, un transmisor y un receptor de vídeo, conectado al equipo que monitorea, procesa y controla los desplazamientos del robot. Para realizar esta tarea, también se requiere contar con los transmisores de radio frecuencia, para prescindir de los cables.

7.6 Actividades de recreación

Se pueden incluir varias actividades curiosas o de entretenimiento para ser realizadas por el Robot Móvil, como se ha venido mencionando, depende de la inventiva e imaginación que se tenga, entre las que se pueden mencionar las siguientes:

a) Robot que siga a una persona utilizando una lámpara de luz como guía;

b) Robot que persiga a otro robot a donde el último se dirija;

c) Robot que permanezca desplazándose dentro de un escritorio o una mesa, de tal forma que no se caiga de ella; etc.

7.7 Aplicaciones avanzadas

Es posible la manipulación y utilización del Robot Móvil, en aplicaciones más avanzadas, donde se requerirán conocimientos más amplios en otras áreas de investigación, llamece procesamiento de voz, imágenes, telecomunicaciones y afines, como consecuencia de interfaces más complejas.

A continuación se presentan ejemplos de este tipo, que en alguna ocasión se ha logrado ejecutar por estudiantes con conocimientos en esas área:

a) Comando del robot empleando procesamiento de voz, es decir; se controlan todas las acciones que deseamos se ejecuten indicando con voz natural todos los movimientos y operaciones;

b) Planeación de rutas empleando sistemas expertos; el robot se desplazará con ayuda de los datos obtenidos del sistema experto, quien planeará el camino más adecuado y óptimo a seguir por el robot, para que se desplace de un punto denominado origen a otro punto llamado destino;

c) Aplicación y uso de redes neuronales, para que el Robot Móvil navegue en un entorno desconocido y se eviten las coaliciones; etc.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

SERVO CONTROL OF A *DC* – BRUSH MOTOR

PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ
PALACIO DE MINERÍA
ABRIL DE 1999

MICROCHIP

AN532

Servo Control of a DC-Brush Motor

Author: Tim Bucella, Teknic Inc.

INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC-like features, the PIC17C42 offers excellent computation speed needed for real time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 µS (@16 MHz) allowing control loop sample times in the 2 KHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals of the PIC17C42 allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs. However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems offering a number of advantages:

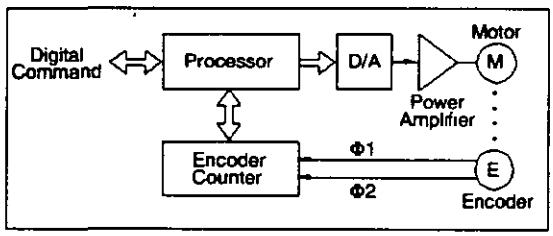
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of digital-to-analog converter, and a power amplifier, which delivers current or voltage to the motor.

FIGURE 1 - A TYPICAL SERVO SYSTEM



The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or the simple PID algorithm (implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The digital-to-analog conversion can be handled by a conventional DAC or by using pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because of its simplicity and reduced cost.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of Φ1 and Φ2 represents an increment of position. The phase of the signals is used to determine direction of rotation.

4

FIGURE 2 - THE PIC17C42 SERVO SYSTEM

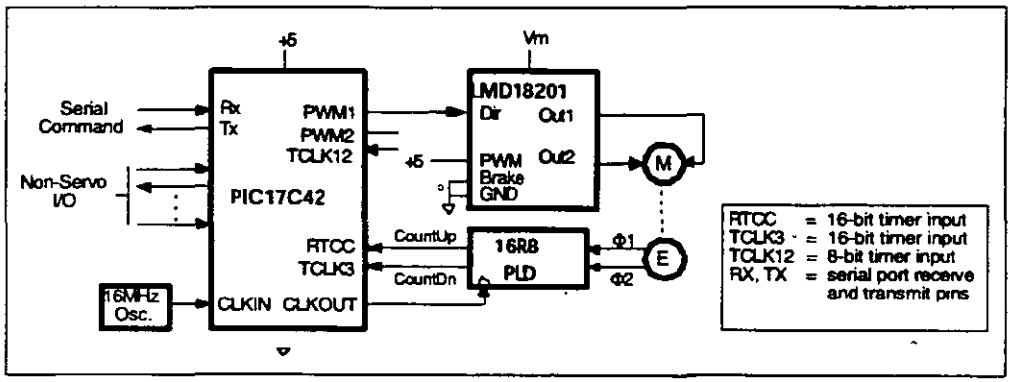
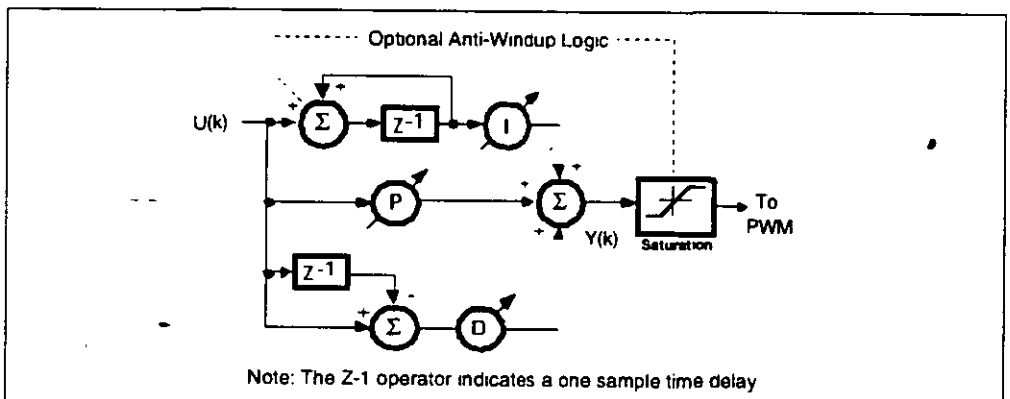


FIGURE 3 - THE PIC17C42 BASED SERVO CONTROL BOARD

Place Servo Control Board photo here.

FIGURE 4 - DIGITAL PID IMPLEMENTATION



Note: The Z-1 operator indicates a one sample time delay

These encoder signals are usually decoded using a small state machine into Count Up and Count Down pulses. These pulses are then routed to an N-bit, up-down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 3. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the digital-to-analog conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLD. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figures 5 and 6).

THE COMPENSATOR

PID is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, however it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 4. U(k) is the position or velocity error and Y(k) is the output.

This algorithm has been implemented using the PIC17C42 math library. Only 800 instruction cycles are required resulting in a 0.2mS PID execution time at 16 MHz.

Integrator wind-up is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown above avoids this problem by stopping the action of the integrator during output saturation.

MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5nS (at 16 MHz). This translates into 10-bit resolution at a 15.6KHz rate or 1 part in 800 (9 1/2-bit) resolution at 20KHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5mS or more and a mechanical time constant of 20.0mS or more. A 15KHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 2, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, V_m. In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 2. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 5). These signals are then fed to two 16-bit timers of the PIC17C42 (TMR3 and RTCC). A logic description for the PLD decoder is shown in Appendix B.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits deep, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. **For example, at a servo sample time of 1mS, the maximum encoder rate would be 3.2767 MHz.**

Counter wrap-around is not a concern because only the *difference* between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.

FIGURE 5 - SEQUENTIAL ENCODER SIGNALS

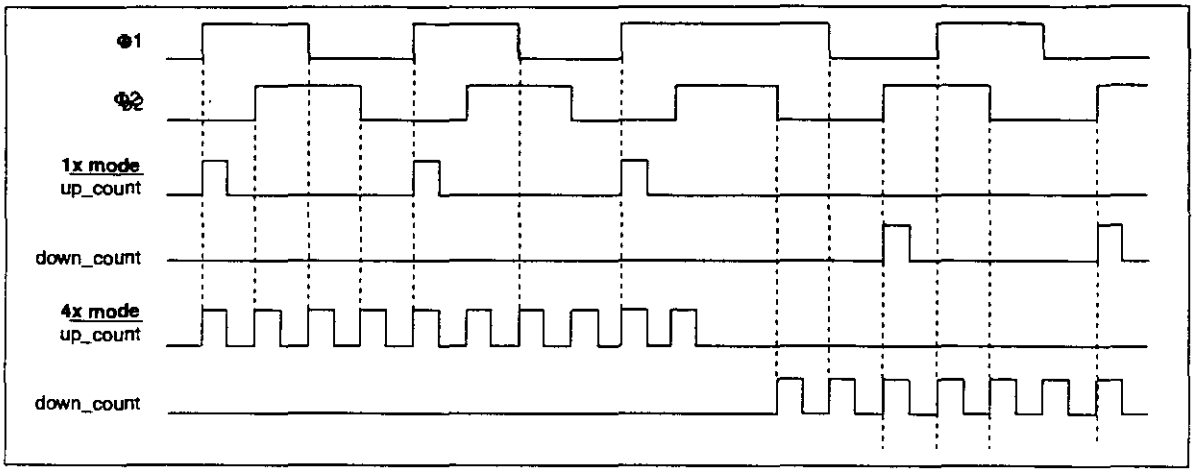
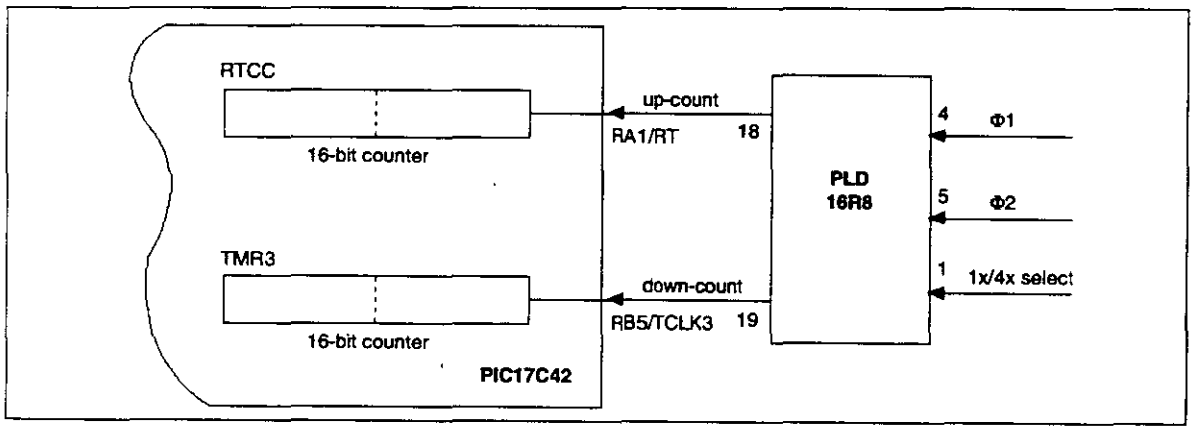


FIGURE 6 - ENCODER INTERFACE SCHEME



TRAJECTORY GENERATION

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular short move where maximum velocity was not reached (Figure 8).

The doPreMove subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The doMove routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

$V_K = V_{K-1} + A$ (A = Acceleration)

$P_K = P_{K-1} + V_{K-1} + A/2$

For more details on trajectory generation, see Appendix E.

IMPLEMENTATION DETAILS

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication and any exception processing (i.e. limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM, all at a constant, predefined rate. The ISR is initiated by a hardware timer (TMR2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N=16 (Figure 7).

FIGURE 7 - SAMPLING SCHEME

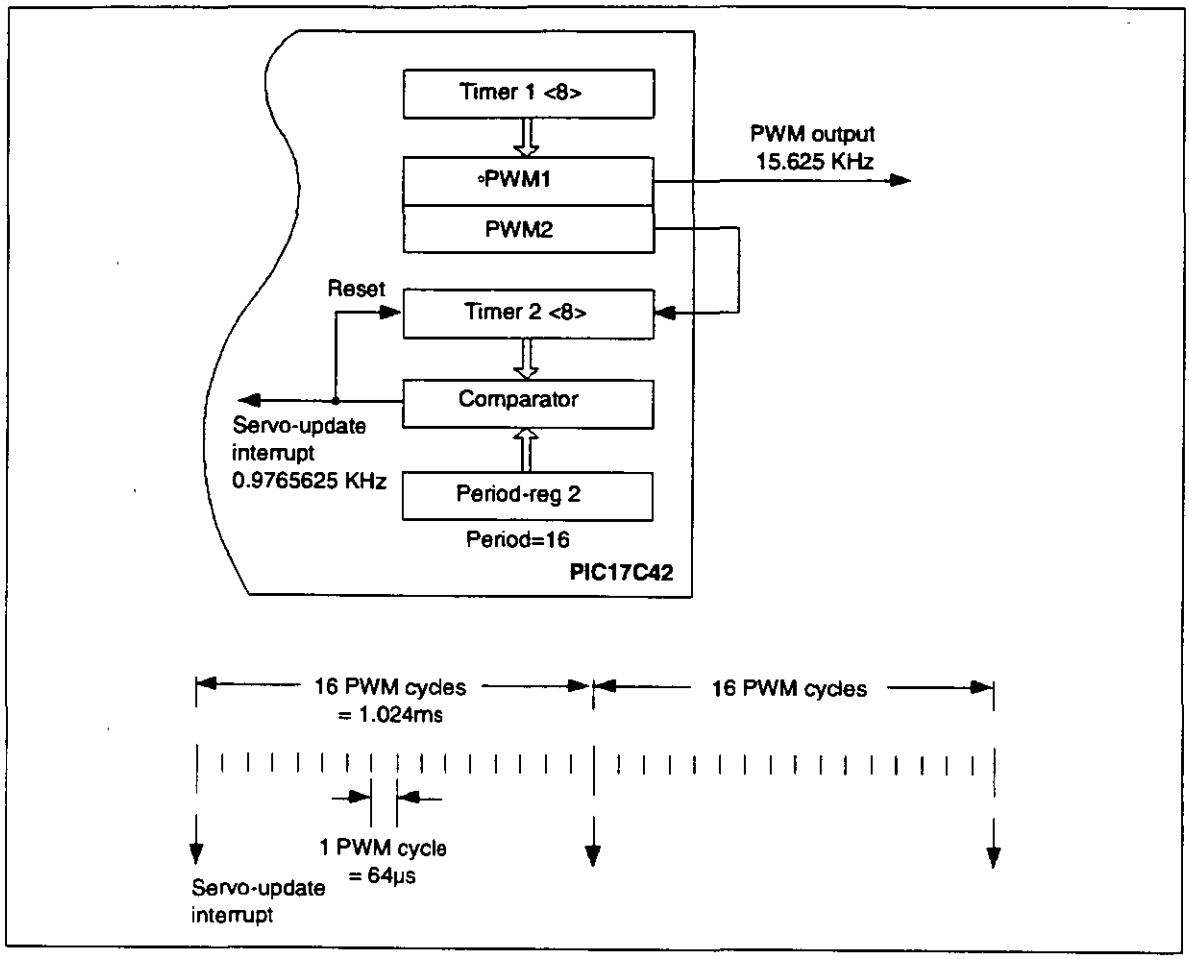


FIGURE 8 - VELOCITY RAMP SEGMENTS FOR POSITION MOVES

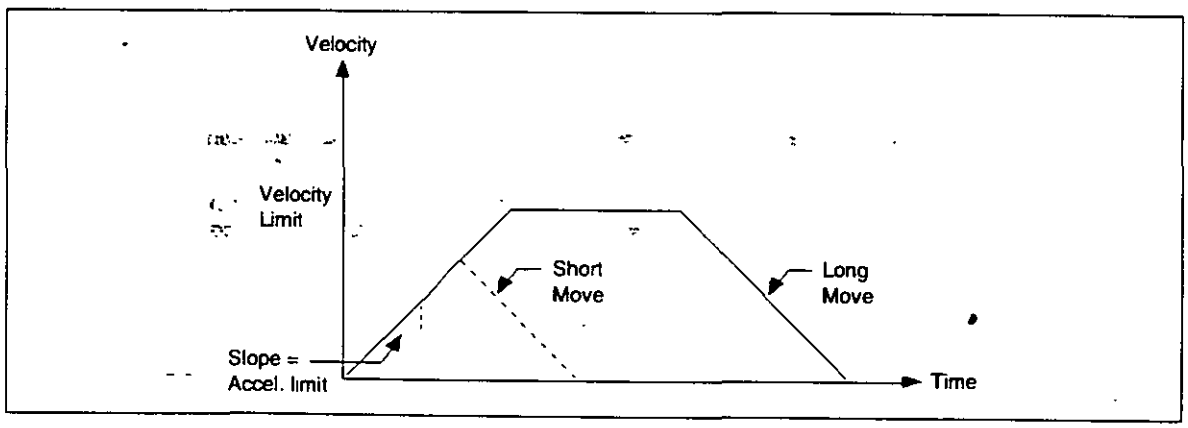


FIGURE 9 - FLOWCHART FOR FORE-GROUND PROCESSING

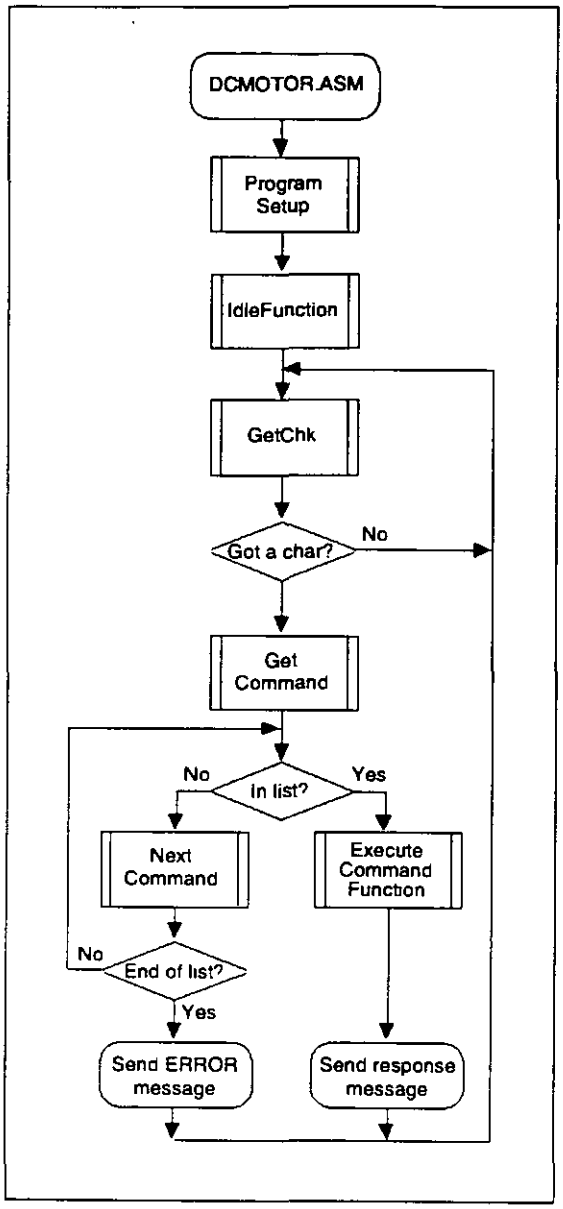
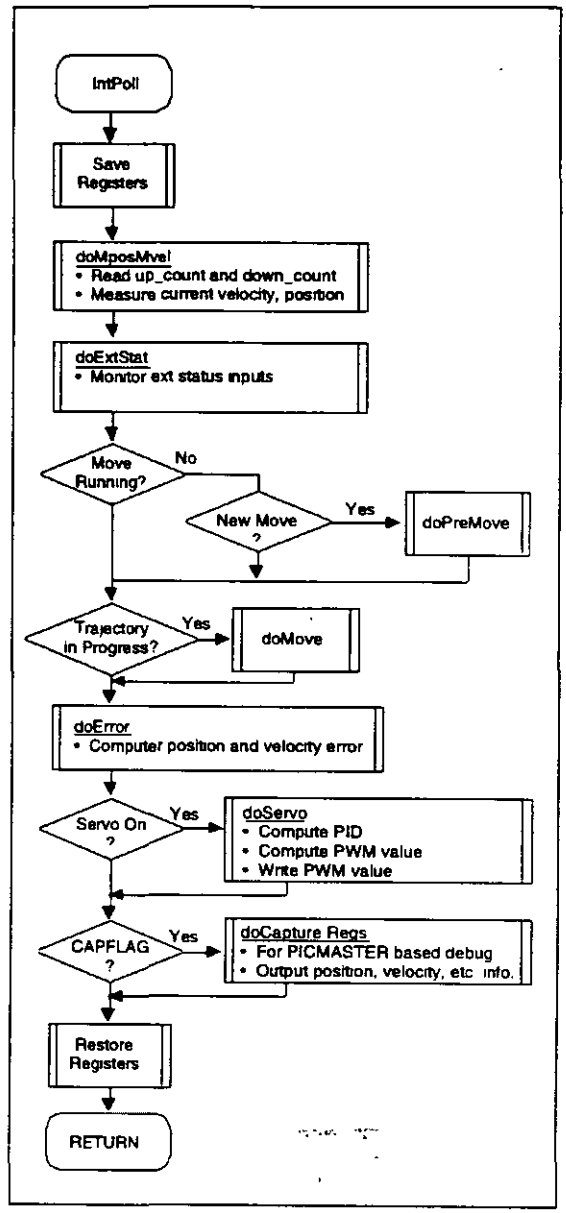


FIGURE 10 - FLOWCHART FOR INTERRUPT SERVICE ROUTINE



The following events must occur in the interrupt service routine:

- Read Timers (RTCC & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error: U(k) = Reference Position - Current Position
- Calculate Y(k) using PID
- Set PWM output
- Manage other housekeeping tasks (i.e. service serial characters)

The entire ISR requires only 0.250mS to execute with 16MHz processor clock frequency.

COMMAND INTERFACE

The following commands are implemented and recognized by the user interface in the foreground loop.

Move (Value): M, [$-8,388,608_{10}$ to $8,388,607_{10}$]

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

Mode: Q, (Type), [P,V, T]

An argument of "P" will cause all subsequent move commands to be incremental position moves. A "V" argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

Set Parameter: S, (#,Value) [00_h to FF_h, $-8,388,608_{10}$ to $8,388,607_{10}$]

Sets controller parameters to the value given. Parameters are shown in Table 1.

TABLE 1 - PARAMETERS

Parameter	#$_h$	Range
Velocity Limit	00	0 to $8,388,607_{10}$ *
Acceleration Limit	01	0 to $8,388,607_{10}$ **
Kp: Proportional Gain	02	-32768_{10} to 32767_{10}
Kd: Differential Gain	03	-32768_{10} to 32767_{10}
Ki: Integral Gain	04	-32768_{10} to 32767_{10}

* (counts per sample time multiplied by 256)

** (counts per sample time per sample time multiplied by 256)

Read Parameter: R, (#) [00_h to FF_h]

Returns the present value of a parameter.

Shutter: C

Returns the time (in sample time counts 0 to $65,536_{10}$) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

Read commanded position: P

Returns the commanded position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read commanded velocity: V

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$

Read actual position: p

Returns the actual position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read actual velocity: v

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

External Status: X

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

TABLE 2 - EXTERNAL STATUS REGISTER BITS

Bit 7	index marker detected
Bit 6	+limit reached
Bit 5	-limit reached
Bit 4	input true
Bit 3-0	n/a

Move Status: Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

TABLE 3 - MOVE STATUS REGISTER BITS

Bit 7	move buffer empty
Bit 6	move complete
Bit 5-0	n/a

Read Index position: I

Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,608$_{10}$ to 8,388,607$_{10}$]

Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.

Reset: Z

Performs a software reset.

Capture Servo-Response: c (#Count)

The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the doCaptureRegs procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see doCaptureRegs procedure). This is useful for debug purposes.

Disable Servo: s

This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.

Examples:

```
Z           ;Reset software (No <CR> required)
OV          ;Set velocity servo mode (No <CR>
                    ;required)
M 1000<CR> ;Set velocity to 1000
M-1000<CR> ;Set velocity to 1000 in reverse
                    ;direction
```

OPTIMIZING THE SYSTEM

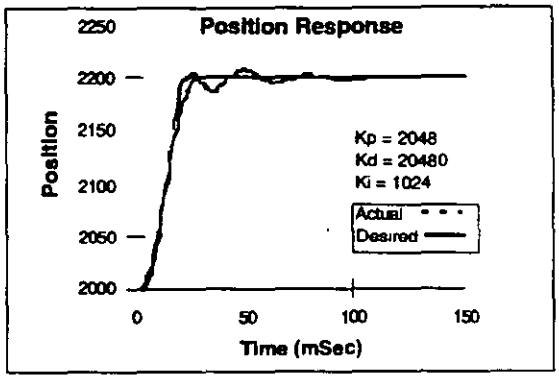
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER™ In-Circuit Emulator for the PIC17C42.

The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, 8K deep trace buffer and external logic probes. It's user interface software runs under Windows™ 3.1 with pull-down menus and on-line help. The PICMASTER software also support dynamic data exchange (DDE) through which it is possible to send its trace buffer information to a spreadsheet, such as EXCEL™, also running under windows.

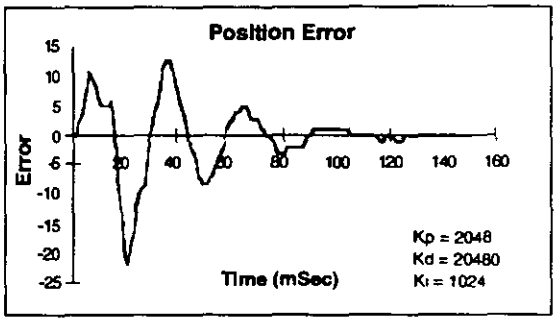
To tune the PID, first a small amount of diagnostics code is added in the servo_routine (doCaptureRegs). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using TABLWT instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle TABLWT instructions. Next, the trace buffer is transferred to EXCEL and the various

FIGURE 11 - TYPICAL SERVO RESPONSE:

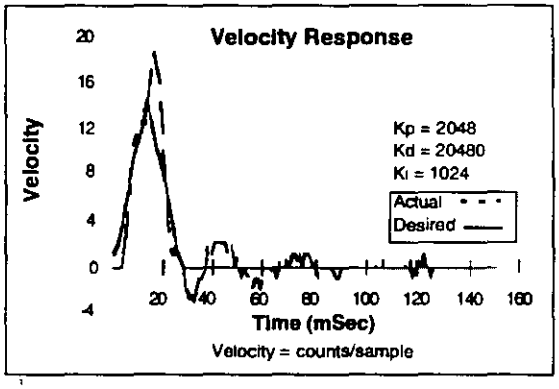
DESIRED/ACTUAL POSITION



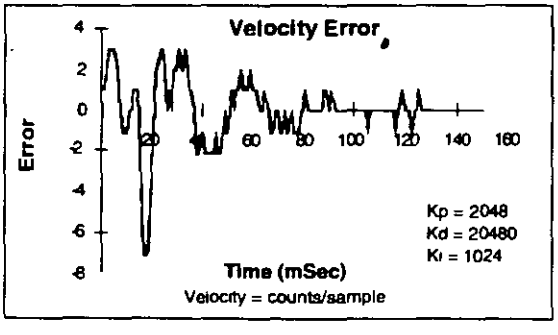
POSITION ERROR



DESIRED/ACTUAL VELOCITY



VELOCITY ERROR



parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering Kp, Ki and Kd, and plotting the results, the system can be fine tuned.

Under windows multitasking environment, using PICMASTER emulator this can be done in real time as described below.

Three sessions are set up under windows:

1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are setup to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setup have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

TABLE 4 - SERVO CONTROL CHIP COMPARISON

	LM629 @8MHz	PIC17C42 @16MHz	PIC17C42 @25MHz
Max Encoder Rate	1MHz	3.3 MHz	4.5 MHz
Servo Update Time	-	0.25 ms	0.16 ms
Max Sampling Frequency	4 KHz	2-3 KHz	4-5 KHz

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.

Credit

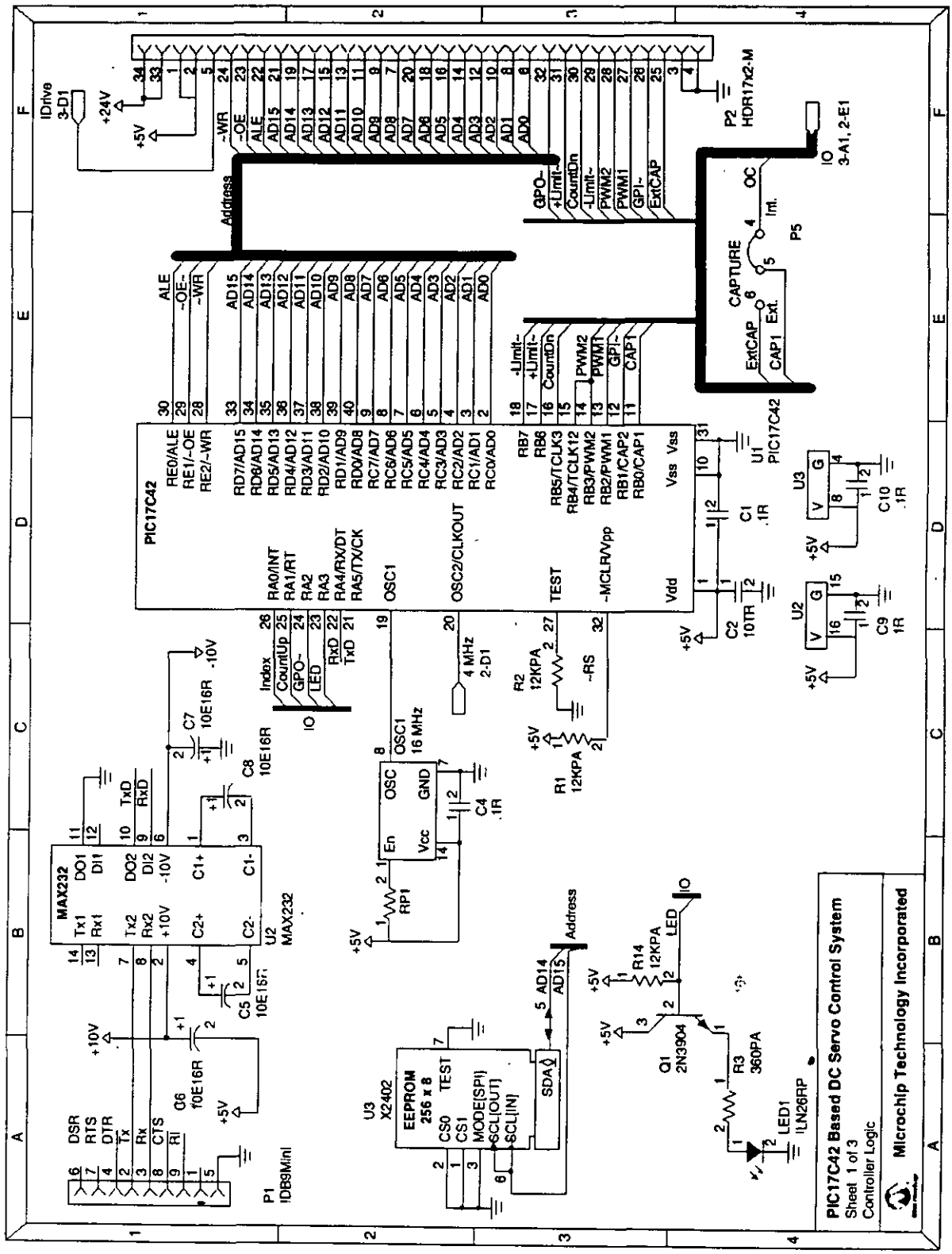
This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

References

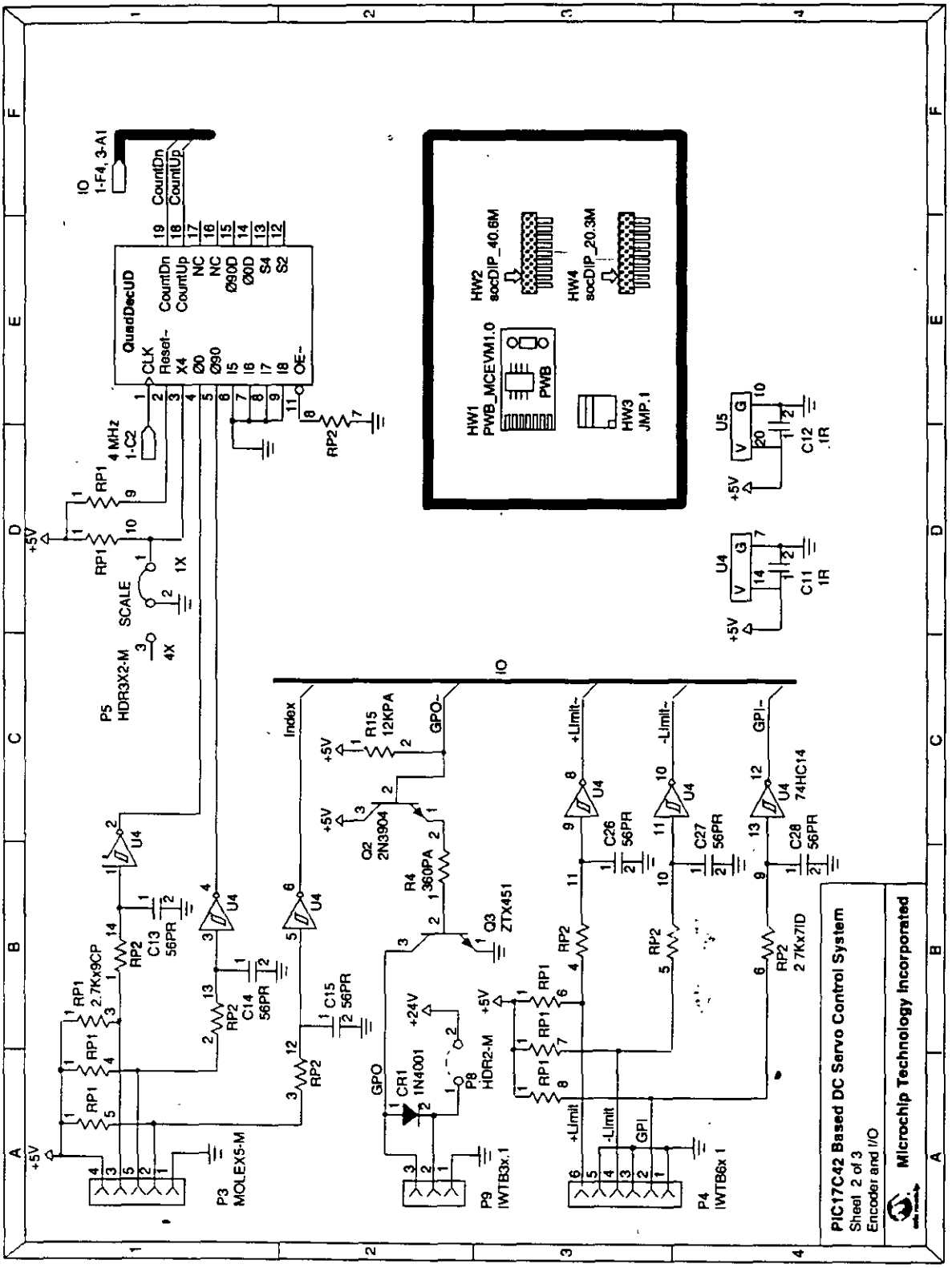
1. Thomas Bucella, "Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data", PCIM conference proceedings © 1990 Intertec Communications, Inc.
2. David M. Auslander, Cheng H. Tham, "Real-Time Software for Control" © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ
3. "DC Motors, Speed Controls, Servo Systems" Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

Windows is a trademark of Microsoft Corporation.

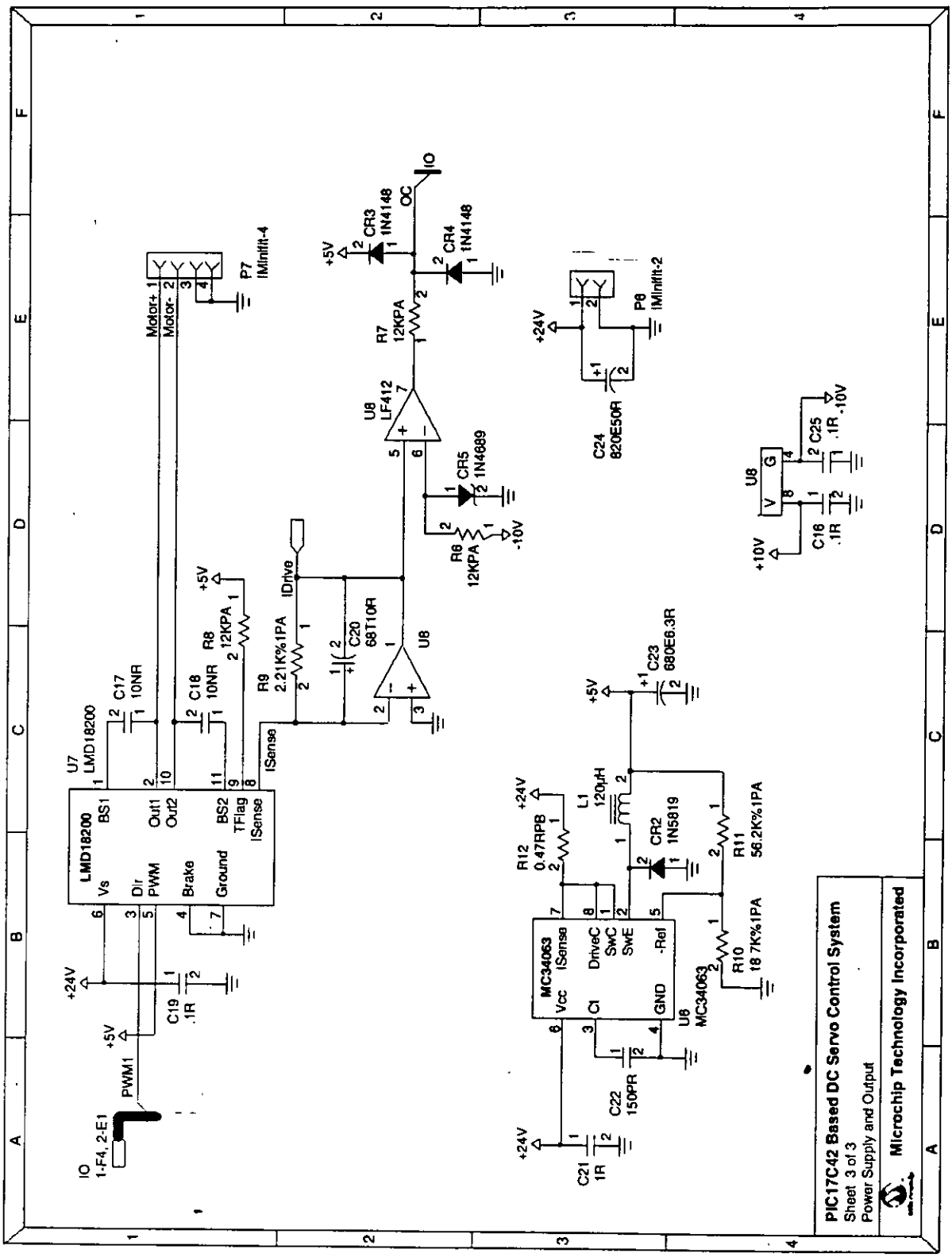
APPENDIX A: SCHEMATIC DIAGRAM



APPENDIX A (CONT.): SCHEMATIC DIAGRAM



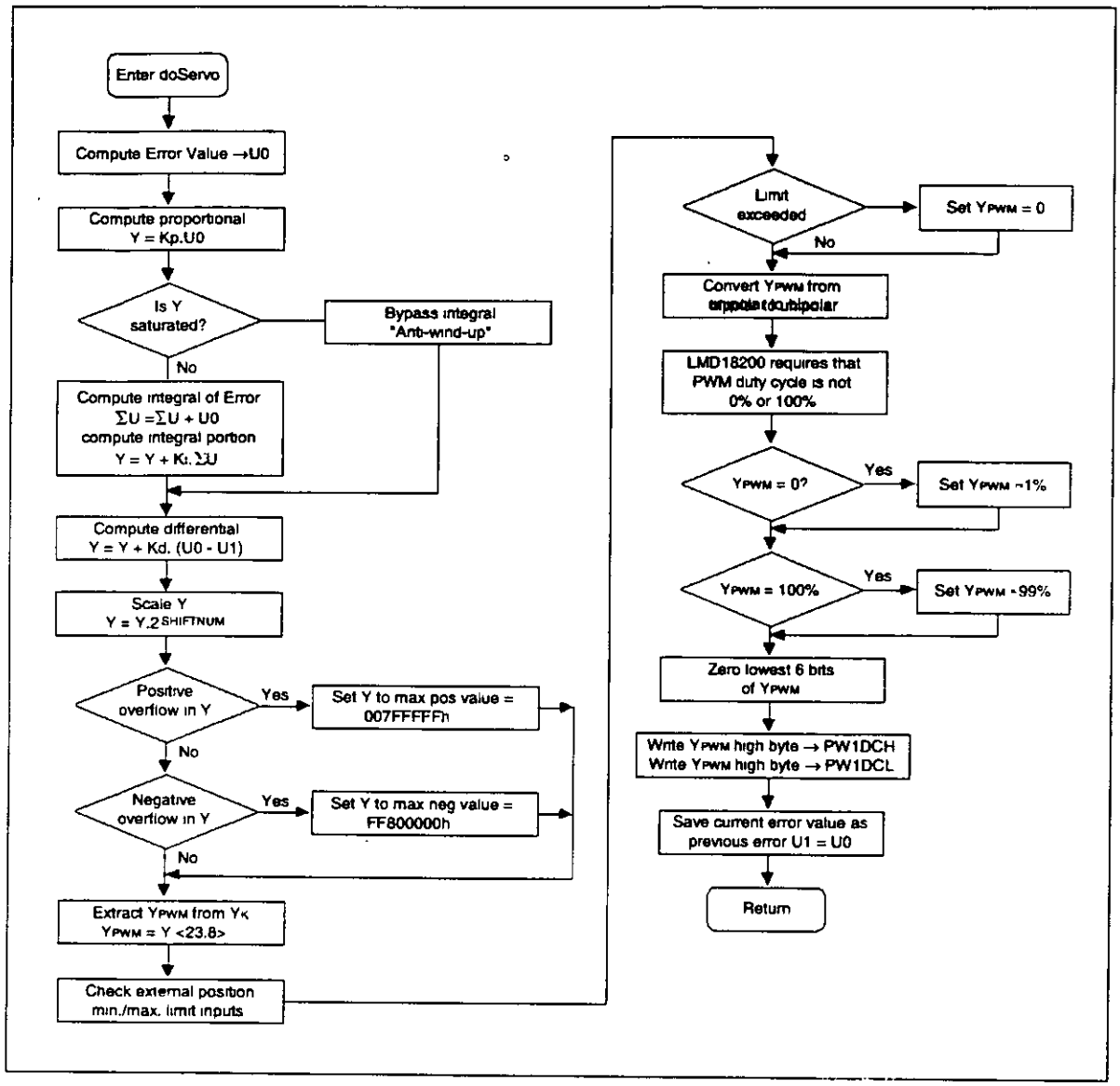
APPENDIX A (CONT.): SCHEMATIC DIAGRAM



APPENDIX B: ENCODER PLD EQUATIONS

```
Combination quadrature decoder and input synchronizer.  This design
allows 1x decoding or 4x decoding based on the X4 pin.

        * Ver 1.0 - November 8, 1991
}
MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

TYPE MMI 16R8;
INPUTS;
        RESET NODE[PIN2] INVERTED;
        X4 NODE[PIN3];
        P0 NODE[PIN4];                  { Phi0 }
        P90 NODE[PIN5];                         { Phi90 }
        INDX NODE[PIN6];
        { Feedback pins }
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
OUTPUTS;
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
TABLE;
        S2 := P0D & !RESET;
        S4 := P90D & !RESET;
        P0D := P0 & !RESET;
        P90D := P90 & !RESET;

        CntUp := COUNT & UP;
        CntDn := COUNT & !UP;

        COUNT :=
              (  P0D &  S2 & !P90D &  S4 & X4 { C1 }
               +!P0D & !S2 &  P90D & !S4      { C2 }
               +!P0D &  S2 & !P90D & !S4 & X4 { C3 }
               + P0D & !S2 &  P90D &  S4 & X4 { C4 }
               + P0D &  S2 &  P90D & !S4 & X4 { C5 }
               +!P0D & !S2 & !P90D &  S4      { C6 }
               +!P0D &  S2 &  P90D &  S4 & X4 { C7 }
               + P0D & !S2 & !P90D & !S4 & X4 { C8 }
               ) & !RESET;
        UP :=
               (
                !P0D &  S2 & !P90D &  S4
               +!P0D &  S2 &  P90D &  S4
               +!P0D &  S2 &  P90D & !S4
               + P0D &  S2 &  P90D & !S4
               + P0D & !S2 &  P90D & !S4
               + P0D & !S2 & !P90D & !S4
               + P0D & !S2 & !P90D &  S4
               +!P0D & !S2 & !P90D &  S4
               ) & !RESET;
END;

END QuadDivider;
```

APPENDIX C: (PART 1): PID ALGORITHM FLOWCHART



APPENDIX C: (PART 2): PID ALGORITHM CODE LISTING

```
;*****************************************************************************
; NAME:         doServo
;
; DESCRIPTION: Performs the servo loop calculations.
;
doServo

        MOV16   POSERROR,U0             ; save new position error in U0

        LOADAB  U0,KP                   ; compute KP*U0                          ┐ Basic PID
        CALL    Dmult                                                            │ calculation
        MVPF32  DPX,Y                   ; Y=KP*U0                                │
                                                                                 │
        CLRF    WREG                    ; if previous output saturated, do ┐ anti-│
        CPFSGT  SATFLAG                 ; not accumulate integrator        ┘ wind-│
        CALL    doIntegral                                                   up  │
                                                                                 │
        LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL                    │
        CALL    Dmult                                                            │
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL                    │
                                                                                 │
        MVFP16  U0,AARG                 ; compute KV*(U0-U1)                     │
        SUB16   U1,AARG                                                          │
        MVFP16  KV,BARG                                                          │
        CALL    Dmult                                                            │
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)         ┘

        CLRF    WREG                                                             ┐
        CPFSGT  SHIFTNUM                    ; scale Y by SHIFTNUM                │ Scale Y
        GOTO    grabok                  ; Y = Y * (2**SHIFTNUM)                  │
        MOVFP   SHIFTNUM,TMP                                                     │
grabloop                                                                         │
        RLC32   Y                                                                │
        DECFSZ  TMP                                                              │
        GOTO    grabloop                                                         ┘

grabok
        CLRF    SATFLAG
        BTFSC   Y+B3,MSB                ; saturate to middle 16 bits,
        GOTO    negs                    ; keeping top 10 bits for PW1DCH
poss                                     ; and PW1DCL
        MOVFP   Y+B2,WREG               ; check if Y >= 2**23                    ┐ If positive
        ANDLW   0x80                                                             │ overflow, saturate
        IORWF   Y+B3                                                             │ y to maximum
        CLRF    WREG                                                             │ positive number
        CPFSGT  Y+B3                                                             │
        GOTO    zero6bits               ; if not, zero 6 bits                    │
                                                                                 │
        INCF    SATFLAG                 ; if so, set Y=0x007FFFFF                │
        CLRF    Y+B3                    ; clear for debug purposes               │
        MOVLW   0x7F                                                             │
        MOVPF   WREG,Y+B2                                                        │
        SETF    Y+B1                                                             │
        SETF    Y+B0                                                             │
        GOTO    zero6bits                                                        ┘
negs
        MOVFP   Y+B2,WREG               ; check if Y <= -2**23                   ┐ If negative
        IORLW   0x7F                                                             │ overflow, saturate
        ANDWF   Y+B3                                                             │ y to maximum
        SETF    WREG                                                             │ negative value
        CPFSLT  Y+B3                                                             │
        GOTO    zero6bits               ; if not, zero 6 bits                    │
                                                                                 │
        SETF    SATFLAG                 ; if so, set Y = 0xFF800000              │
        SETF    Y+B3                                                             │
        CLRF    Y+B2                                                             │
        BSF     Y+B2,MSB                                                         │
        CLRF    Y+B1                                                             │
        CLRF    Y+B0                                                             ┘
```

```
zero6bits
        MOV24   Y+B1,YPWM+B0            ; move Y to YPWM and zero 6 bits
doTorque                                ; entry point for torque mode
        MOVLW   0xC0
        ANDWF   YPWM+B0

        BTFSC   YPWM+B1,MSB
        GOTO    tmlimit
tplimit                                                         If external
        BTFSS   EXTSTAT,BIT6                                    position
        GOTO    mplimitok                                       limits have been
        CLR32   YPWM                                            reached then zero
        GOTO    mplimitok                                       PWM output.
tmlimit
        BTFSS   EXTSTAT,BIT5
        GOTO    mplimitok
        CLR32   YPWM
mplimitok                                                       Convert PWM from
        MOVLW   PW1DCH_INIT             ; adjustment from bipolar to unipolar   unipolar to bipolar
        MOVPF   WREG,TMP+B1             ; for 50% duty cycle
        MOVLW   PW1DCL_INIT
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

        CLRF    TMP+B1                  ; correct by 1 LSB
        MOVLW   0x40                    ; add one to bit5 of PW1DCL
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

testmax                                                         PWM cycle must not
        CLRF    TMP+B2                  ; check pwm maximum limit       be 0% or 100%
        CLRF    YPWM+B2                 ; LMD18200 must have a minimum pulse
        CLRF    YPWM+B3                 ; so duty cycle must not be 0 or 100%
        MVFP16  YPWMAX,TMP
        SUB24   YPWM,TMP
        BTFSS   TMP+B2,MSB
        GOTO    testmin
        MOV16   YPWMAX,YPWM             ; saturate to max
        GOTO    limitok
testmin
        CLRF    TMP+B2                  ; check pwm minimum limit
        CLRF    YPWM+B2
        CLRF    YPWM+B3
        MVFP16  YPWMIN,TMP
        SUB24   YPWM,TMP
        BTFSC   TMP+B2,MSB
        GOTO    limitok
        MOV16   YPWMIN,YPWM             ; saturate to min
limitok
        MOVLB   BANK3                   ; set new duty cycle            Write PWM values
        MOVFP   YPWM+B0,PW1DCL                                          to PWM registers
        MOVFP   YPWM+B1,PW1DCH

        MOV16   U0,U1                   ; push errors into U(k-1)

        RETURN

;*****************************************************************************
```

APPENDIX D: ENCODER INTERFACE ROUTINE

```
;*******************************************************************************
; NAME:                         doMPosMVel
;
; DESCRIPTION:          Calculates current position from UpCount and DownCount
;

doMPosMVel

; Do UpCounter first

                        MVFP16  UPCOUNT,TMP+B0          ; save old upcount
readUp
                        MOVPF   RTCCH,WREG
                        MOVPF   RTCCL,UPCOUNT+B0
                        CPFSEQ  RTCCH                   ; Skip next if HI hasn't changed
                        GOTO    readUp                  ; HI changed, re-read LO
                        MOVPF   WREG,UPCOUNT+B1         ; OK to store HI now

                        CLRF    MVELOCITY+B0            ; clear bits below binary point

                        MOV16   UPCOUNT,MVELOCITY+B1    ; compute upcount increment
                        SUB16   TMP+B0,MVELOCITY+B1

; Now do DownCounter

                        MVFP16  DOWNCOUNT,TMP+B0        ; save old downcount
readDown
                        MOVLB   BANK2                   ; timers in Bank 2
                        MOVPF   TMR3H,WREG
                        MOVPF   TMR3L,DOWNCOUNT+B0
                        CPFSEQ  TMR3H                   ; Skip next if HI hasn't changed
                        GOTO    readDown                ; HI changed, re-read LO
                        MOVPF   WREG,DOWNCOUNT+B1       ; OK to store HI now

                        MVFP16  DOWNCOUNT+B0,TMP+B2     ; compute downcount increment
                        SUB16   TMP+B0,TMP+B2

                        SUB16   TMP+B2,MVELOCITY+B1     ; compute new measured velocity

                        CLRF    MVELOCITY+B3            ; sign extend measured velocity for
                        BTFSC   MVELOCITY+B2,MSB        ; 24 bit addition to measured posi-
tion
                        SETF    MVELOCITY+B3


                        ADD24   MVELOCITY+B1,MPOSITION; compute new measured position
                                                        ; delta position = measured velocity

                        RETURN

;*******************************************************************************
```

APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

doPreMove:

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if abs (MOVVAL) > 2, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and MOVVAL/2 in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at modeready, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

doMove:

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by

$$x(t) = x0 + v0*t + a*(t**2)/2, v(t) = v0 + a*t$$

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by

$$P(k) = P(k-1) + V(k-1) + A/2, \quad V(k) = V(k-1) + A,$$

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by

$$P(k-1) = P(k) - V(K-1) - A/2, \quad V(K-1) = V(k) - A.$$

In position mode, the actual shape of the velocity profile depends on the values of V, A and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speed down sections of the move. Phase 1 is defined as the section of the move where the commanded position is less than half the move, and phase 2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

FIGURE A - SPEED PROFILE FOR TRAPEZOIDAL MOVES

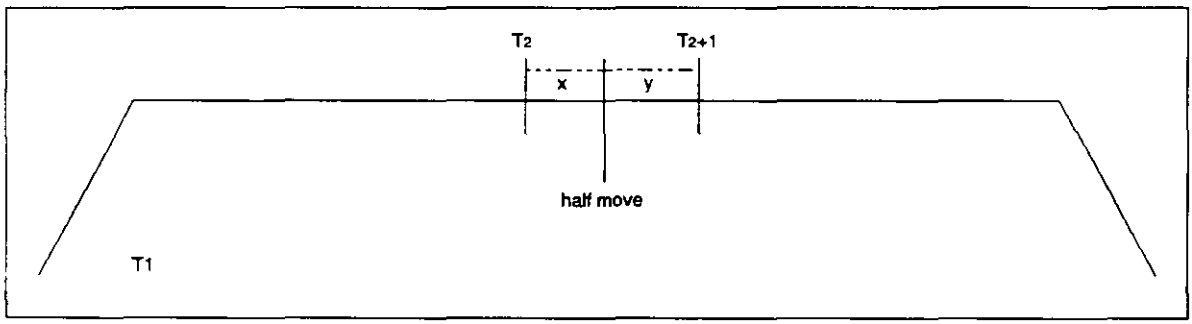


FIGURE B - SPEED PROFILE FOR TRIANGULAR MOVES

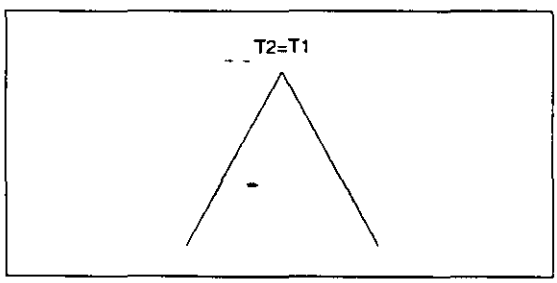
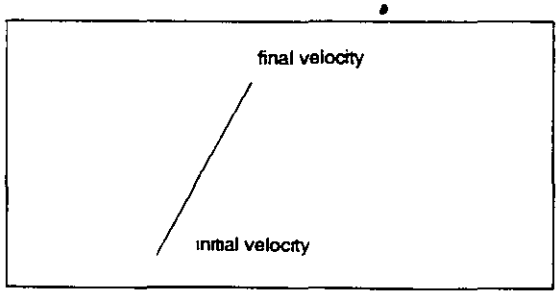


FIGURE C - SPEED PROFILE FOR VELOCITY MOVES



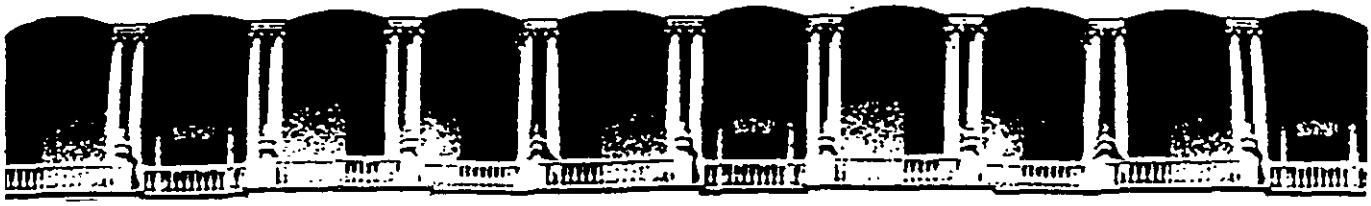
Furthermore, let x be the amount of undershoot and y the amount of overshoot of half the move at T2. Discretization error is minimized by using the values of x and y whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by min. (2x, 2y), resulting in the condition that if 2x > 2y, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by min. (2x, y-x), yielding the condition that if 3x > y, take one more step during the flat section of phase2.

At the beginning of doMove, MOVTIME is incremented and doPosVel is called to evaluate the next proposed values of commanded position and velocity under the current value of A. In position mode, phase1, the original position plus half the move minus the new proposed commanded position is calculated and placed in MOVDEL, with the previous MOVDEL saved in MOVTMP. As half the move would be passed, MOVTMP = -x and MOVDEL = y, with y>0 for the first time indicating that phase1 is about to be completed. Therefore, if y<0, we continue in phase1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, reevaluate the iterative equations with the new acceleration, set T1=MOVTIME-1, and execute the move.

Since T1 is cleared in doPreMove, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that y>0, we drop into code that is executed only one time, with phase2 beginning on the next step. If T1=0, maximum velocity has not yet been reached, so T1=T2 and the velocity profile is triangular. In this case, A is negated for speed down, and if x>y, one more step is needed to minimize the discretization error. So A is negated, the proposed step undone, A is again negated for speed down and the step recalculated and executed, with T2=T1=MOVTIME-1.

If T1 is not zero, indicating that we are in the flat section of phase1, then go to t2net1, where T2=MOVTIME-1, and if 3x>y, then one more phase2 flat step is necessary to minimize the discretization error. PH2FLAT is defined as the number of steps in the flat section of phase2, and is used as a counter during its completion. If 3x>y, then PH2FLAT=T2-T1, otherwise PH2FLAT=T2-T1-1 and phase1 is finally complete. All subsequent steps will proceed through phase2, first deciding if the flat section is finished by checking if PH2FLAT has reached zero. If not, go to flat where PH2FLAT is decremented, and tested if zero. If so, the speed down section is begun by calculating the appropriate signed acceleration limit A, and executing the last of the flat section moves. For all following steps, PH2FLAT=0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speed down.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continued unless the final velocity is zero.



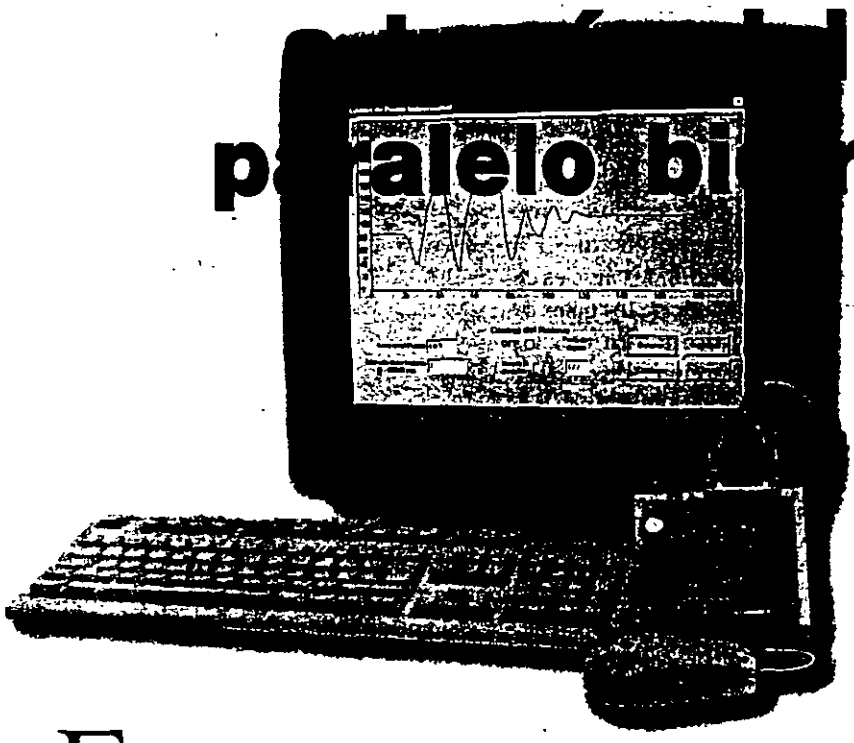
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

ADQUISICIÓN DE DATOS A TRÁVES DEL PUERTO PARALELO BIDIRECCIONAL

PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ
PALACIO DE MINERÍA
ABRIL DE 1999

Adquisición de datos a través del puerto paralelo bidireccional



GUILLERMO RAMOS R.

El puerto paralelo de las nuevas computadoras tiene como propiedad permitir la entrada y salida de datos a través de los pines que sólo servían como salida. Con este proyecto aprenderemos a utilizarlo capturando datos entregados por un conversor análogo digital de alta velocidad.

El puerto paralelo de las computadoras ha sido utilizado ampliamente para diferentes proyectos de electrónica que involucran control a través de salidas y entradas digitales. Uno de los principales inconvenientes tiene que ver con la cantidad de bits de entrada en el puerto, la cual es solamente de 5 en la dirección correspondiente al bus de estado de la impresora (H379). Así entonces, cuando se necesita leer un byte completo, es necesario dividir el byte en dos grupos por medio de la circuitería y hacer dos lecturas desde la computadora, lo que obviamente implica mayor número de componentes y mayor tiempo de lectura por cada byte.

Este problema ha quedado solucionado con las nuevas tarjetas de PC ya que su puerto paralelo permite la configuración de su dirección normal de salidas (H378) también como puerto de entrada, lo que habilita la lectura simultánea de un byte completo. La clasificación y descripción de los puertos paralelos, incluyendo los bidireccionales, puede consultarse en la edición N° 41 de esta publicación.

El circuito, figura 1, y el software descritos en este artículo pueden adquirirse como kit bajo la referencia **K-163** de CEKIT.

Recordemos las principales características de las configuraciones del puerto paralelo:

- **El puerto paralelo SPP**. Puerto paralelo estándar. Corresponde al del original IBM PC. Su bus de datos (H378) es sólo de salida, aunque en las que son configurables, también sirve como entrada. El bus de estado (H379) tiene 5 bits de entrada.

- **El puerto paralelo EPP**. Puerto paralelo mejorado. El puerto de datos es de flujo bidireccional, permitiendo leer o escribir 8 bits a la vez.

- **El puerto paralelo ECP**. Puerto de capacidades extendidas. Esta es la mejor de las configuraciones para puertos paralelos. Además de tener su bus de datos bidireccional, permite altas velocidades de transferencia de información.

Configuración del puerto bidireccional

Si el puerto permite tal configuración, ésta puede hacerse directamente desde el SETUP al momento de arranque de la com-

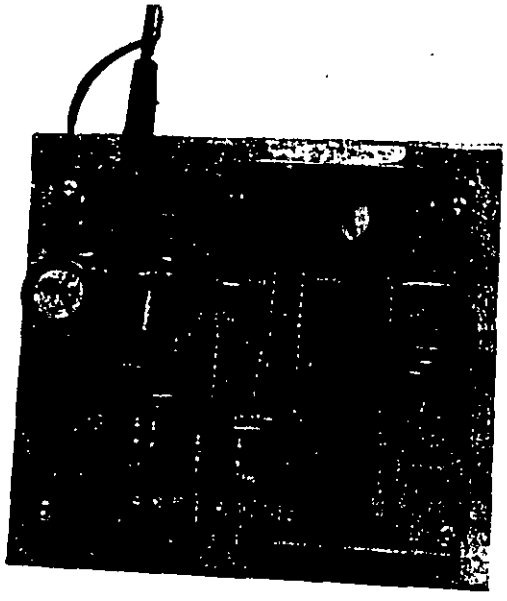


Figura 1. *Kit de adquisición de datos por puerto paralelo bidireccional* ***K-163***

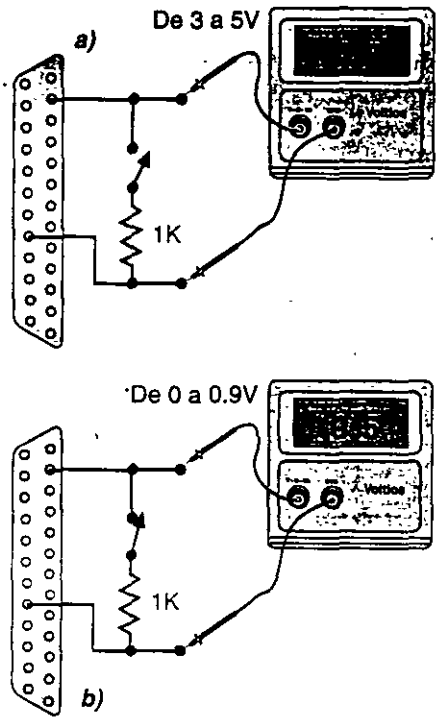


Figura 2. Circuito para comprobar si el puerto está en modo bidireccional

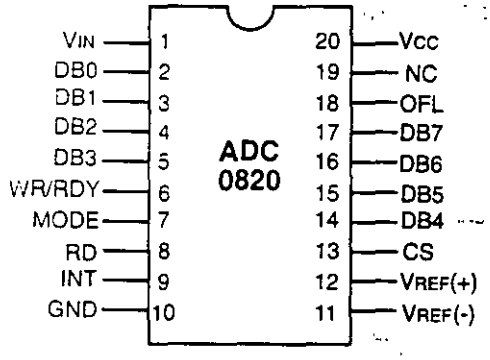


Figura 3. Distribución de pines del ADC 0820

putadora o escribiendo en los bits de control de una de las direcciones del puerto (H37A). En el primer caso, luego de entrar al SETUP, se elige la opción de *INTEGRATED PERIPHERALS* o la opción equivalente y se cambia el modo del puerto paralelo a EPP o a ECP. Si se hace escribiendo directamente en la dirección de control del puerto, bastará con poner un 1 en el bit 5, que es igual a escribir el número 32, en la dirección H37A.

Cómo probar si el puerto es bidireccional

Antes de realizar cualquier tipo de conexiones que requieran escritura de datos sobre el puerto paralelo de la computadora, debemos verificar que su configuración sea la correcta para no correr el riesgo de quemar su circuitería interna. La prueba consiste en la verificación de la impedancia, la cual debe ser un poco alta al permitir la entrada de datos. Previamente debe configurarse el puerto en uno de los modos que permiten comunicación bidireccional.

Para la prueba se pueden utilizar los esquemas que aparecen en la figura 2. La condición inicial es escribir el número 255 (HFF) sobre el puerto H378 para asegurar que tendremos niveles de 5 voltios sobre cada uno de los pines de salida/entrada del puerto. En uno los pines de datos, por ejemplo en el 2, deberá medirse el voltaje con un voltímetro digital utilizando como tierra el pin número 18, ver figura 2a. El nivel de voltaje debe estar entre 3.5 y 5 voltios. Posteriormente aplicamos una carga al puerto, en este caso una resistencia de 1K, llevándola a tierra. Si el puerto permite la entrada de datos, el voltaje deberá variar sustancialmente hasta quedar por debajo de 1 voltio, figura 2b, de lo contrario, el puerto será sólo de salida y no se podrá conectar ningún tipo de señal como entrada.

Si se llega el caso de aplicar señal de entrada a un puerto que sólo es de salida, y si los niveles de voltaje son diferentes, por ejemplo un 1 de salida y un cero como entrada, ocurrirá una colisión, lo que puede destruir la circuitería de salida del puerto de la computadora.

Adquisición de datos

La adquisición de datos a través de una computadora es de vital importancia ya que se pueden generar gráficos, cálculos, estadísticas, etc., de muchos procesos que constantemente estén emitiendo datos de los resultados obtenidos (por ejemplo un contador de botellas) o del estado de la variable a medir (temperatura, voltaje, corriente, etc.).

Igualmente, gracias a esto, es posible generar un archivo de datos y tiempo que pueda ser llevado a una hoja electrónica (Excel, Lotus, etc.) para su debido proceso.

Ya que la naturaleza de las variables que se encuentran con mayor frecuencia son de carácter análogo, es necesario un circuito de interface que convierta estas señales provenientes de los procesos reales en un código binario que pueda ser manejado por circuitos digitales como lo es la computadora.

Conversión análogo digital

La manera más eficaz para que un circuito digital, por ejemplo una computadora, pueda leer el valor de variables análogas, es tomar muestras sucesivas a determinados intervalos de tiempo y convertirlas al formato digital. Los dispositivos electrónicos que se encargan de realizar esta tarea son los conversores análogo a digital o conversores A/D.

Los conversores A/D requieren un intervalo de tiempo para poder realizar una conversión. Este tiempo depende del tipo de conversor empleado, y en últimas determina la máxima velocidad con la que se pueden realizar las muestras

Otro parámetro determinante en este tipo de dispositivos es la resolución, que indica el número de bits de código empleados para representar los valores análogos que se están convirtiendo y establece la precisión del conversor, es decir, si el conversor es de 8 bits, podrá entregar 2^8=256 valores diferentes, en cambio, si el conversor es de 12 bits entregará 2^{12}=4096 valores diferentes (mayor precisión).

Cada aplicación tiene requerimientos específicos de tiempo de conversión y de resolución de los conversores que se vayan a utilizar. Por ejemplo, los sistemas de monitoreo de temperatura no requieren tanta rapidez como la medición de corriente o de voltaje.

El circuito integrado ADC 0820

Este circuito integrado tiene la propiedad de ser un conversor rápido, con un tiempo de conversión del orden de uS, mucho más veloz que el ADC 08. Esto quiere decir que es posible efectuar hasta 500.000 muestras de la señal

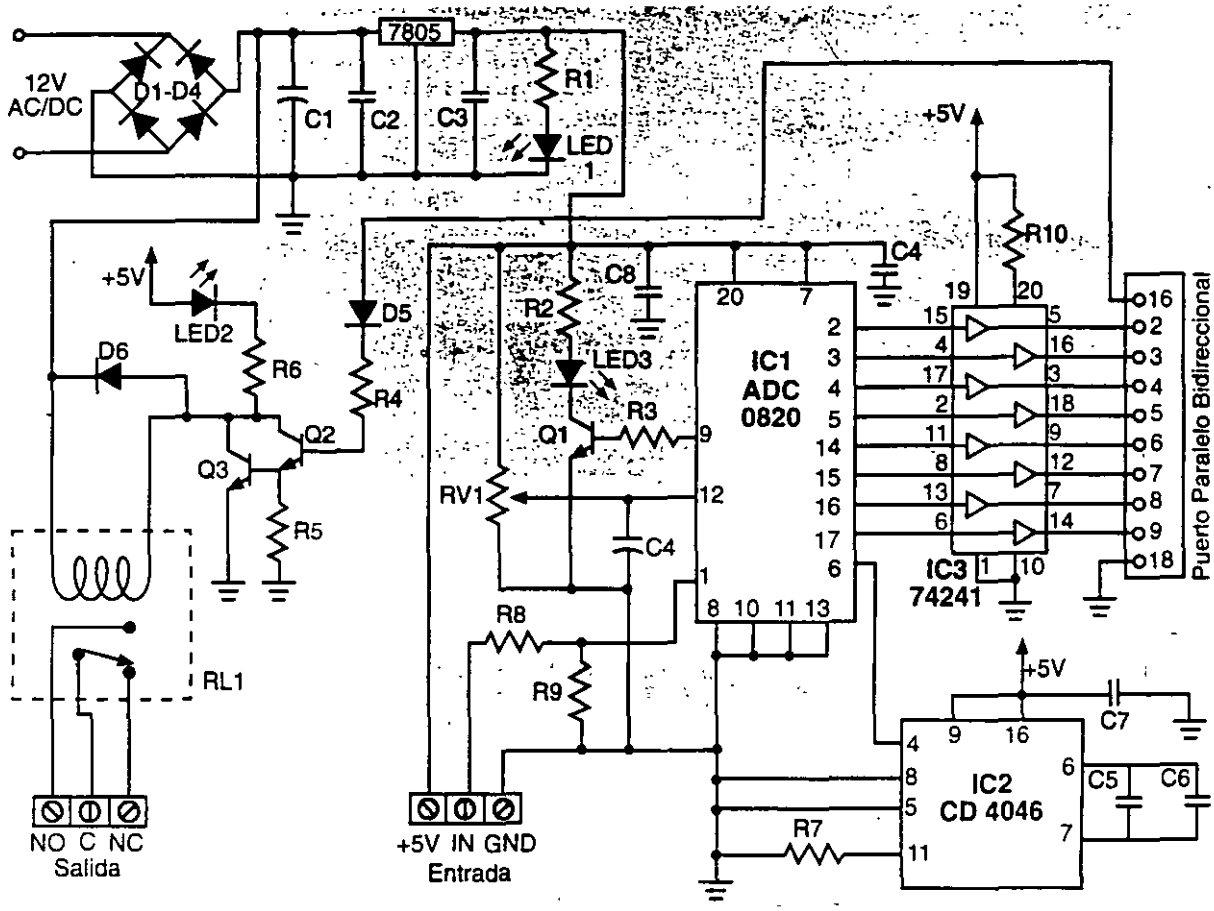


Figura 4. Diagrama esquemático del circuito

por segundo. Observe la distribución de sus pines en la figura 3. Su alta velocidad se debe al método de conversión utilizado: dos grupos de 4 bits en modo Flash. Primero se hace una conversión de sólo 4 bits y el valor obtenido es restado de la señal original para poder hacer la conversión de los bits restantes. A pesar de que el proceso se hace en dos fases, resulta mucho más rápido que los métodos utilizados por los conversores tradicionales.

Hemos elegido este conversor ya que con él podemos leer señales de hasta 250 KHz, mucho más que el ADC 0804 que solamente permite 5 KHz. Esta velocidad de conversión la aprovecharemos para leer desde la computadora a una alta rata de transferencia de datos.

Funcionamiento del circuito del proyecto

En la figura 4 aparece el plano esquemático del circuito diseñado para lectura de señales análogas a través de un puerto paralelo bidireccional. De otra parte, en la figura 5 se puede observar la lista de los componentes correspondientes a dicho esquema.

Normalmente, el conversor necesita un pulso externo para indicarle que debe iniciar una conversión (pin 6). Aunque este pulso puede darse desde la misma computadora, se eligió su aplicación por medio de un circuito de reloj externo con el fin de no aumentar el tiempo de cada lectura desde la computadora. Mediante este reloj, en el programa que se elabore para la captura de datos, nos evitaremos una instrucción adicional que implicaría el envío del pulso de inicio. El reloj fue diseñado a partir del circuito integrado IC2, capaz de manejar señales cuadradas a altas frecuencias. El valor de los condensadores C5 y C6, que están en paralelo para generar capacitancias más exactas (100 pF + 22 pF) , y la resistencia R7 son los encargados de configurar la frecuencia de oscilación, en este caso aproximadamente 500 KHz. Así entonces, cada segundo se realizará tal número de conversiones, logrando muestrear señales análogas de hasta 250 KHz. El circuito integrado entrega la señal de reloj a través del pin 4.

De otro lado, el conversor tiene una salida que indica el final de cada conversión (INT). Este pin está diseñado para generar interrupciones cuando es conectado a algún microprocesador o sistema de proceso. En este proyecto sólo lo utilizamos como indicador de conversión, excitando un LED (LED 3) a través del transistor Q1. Si este LED no enciende, quiere decir que el conversor no está funcionando correctamente.

El circuito integrado de compuertas Buffer (IC3), cuya única función es la de mejorar la corriente de cada uno de los bits entregados por el conversor, se alimenta a través de R10 para eliminar los problemas ocasionados por la frecuencia del reloj del circuito. Si al efectuar las lecturas aparecen datos erróneos, cambie el valor de esta resistencia a 160 ohmios.

Lista de componentes

Resistencias 1/4W
R1, R2, R5, R6 - 330Ω
R3, R4 - 2KΩ
R7- 10KΩ
R8 - 20 Ω
R9 - 1 MΩ
R10 - 68Ω
RV1 - Trimmer de 10KΩ
Condensadores electrolíticos
C1 - 2200µF/35v
Condensadores de tantalio
C4 - 1 µF/16v
Condensadores cerámicos
C2, C3, - 0.1µF/50
C7, C8,C10 - 0.1µF/50
C5 - 100 pF
C6 - 22 pF
Semiconductores

IC1 - Circuito integrado	ADC 0820
IC2 - Circuito integrado	CD 4046
IC3 - Circuito integrado	74LS241
REG1 - Regulador	LM7805
D1-D5 - Diodo rectificador	1N4004
D6 - Diodo rápido	1N4148
Q1-Q3 - Transistor	2N3904

LED1-LED3 - Diodo LED 5 mm
Otros
1 Relevo de 5 pines 12 voltios
2 Bornes de 3 tornillos
1 Conector DB25 para impreso
1 Circuito impreso K-163
1 Conector AC/DC
1 Disquete con programas

Figura 5. Lista de componentes

El control del relevo, que es opcional, se hace a través de una de las salidas del puerto paralelo, por medio de la dirección correspondiente al control de la impresora (H37A), el mismo donde se escribe el bit 5 para la configuración a puerto bidireccional. El pin de control es el número 16, que corresponde al bit número 2 de dicha dirección. Para excitar la bobina del relevo se utilizaron dos transistores en configuración darlington (Q2 y Q3), que reciben la señal del puerto y elevan su corriente para poder suministrar la potencia suficiente a dicha bobina. El diodo D6 tiene como función eliminar los picos de voltaje generados por el relevo en el instante de conmutación.

En el circuito se han puesto dos juegos de tres bornes cada uno. El primero, marcado como "relevo", permite el acceso a los contactos del relevo, el común (COM), el normalmente_abierto (NO) y el normalmente cerrado (NC). El otro juego de bornes permite la conexión de la señal de entrada (IN) y adicionalmente tiene el voltaje de alimentación (+5V y GND) con el fin de suministrar voltaje a algún tipo de sensor que requiera de éste. La señal de entrada siempre deberá estar referida a tierra, es decir, como mínimo deben estar conectados IN y GND.

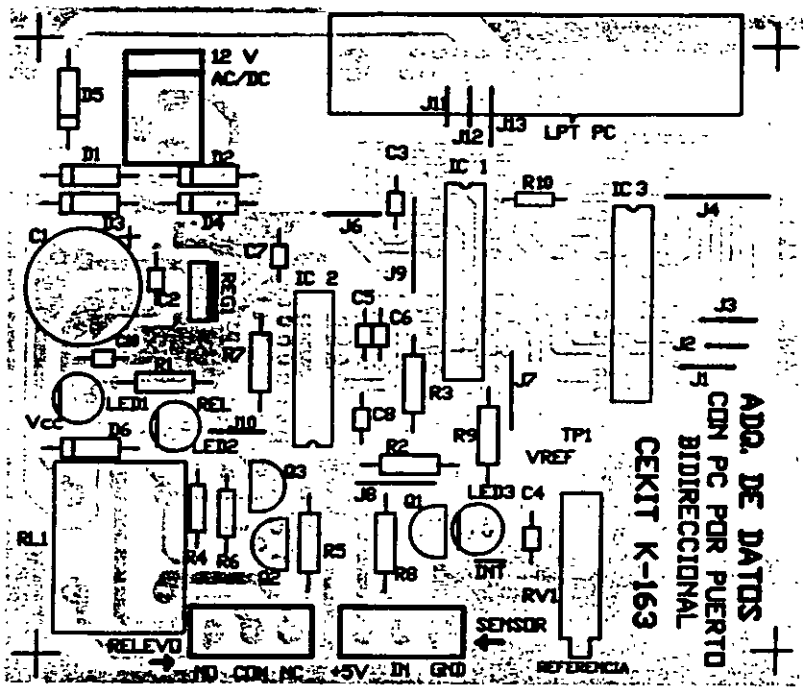


Figura 6. Guía de montaje

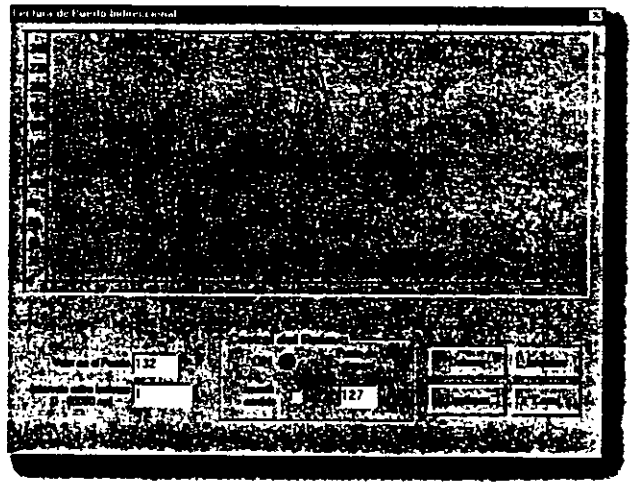


Figura 7. Programa ejecutado en Visual Basic

Montaje del circuito

Para el montaje del circuito, igual que como hemos indicado en los anteriores proyectos, ensamble primero los puentes de alambre y los componentes de menor tamaño, dejando de último los conectores, el condensador C1 y el relevo. Utilice como guía de montaje el gráfico que aparece en la figura 6. Verifique que las soldaduras hayan quedado bien firmes y que no hagan contacto con los puntos contiguos. Debido a que el reloj funciona a una velocidad relativamente alta, asegúrese de limpiar la placa de circuito impreso para que los residuos de resina y de soldadura no queden ejerciendo falsos contactos, lo que alteraría el funcionamiento general del circuito.

Rutinas de programación

Programa en Visual Basic. En este lenguaje hemos elaborado un programa que muestra el valor leído en el puerto y adicionalmente entrega un gráfico en dos dimensiones que indica la evolución de la señal con el transcurso del tiempo. Observe en la figura 7 la ejecución del programa y el listado respectivo en la figura 8. Para la lectura del puerto se hizo uso de la librería *INPOUT. DLL* bajada de Internet.

```
'Declaración de variables
General Declarations
Dim dato(500) As Integer
Dim i As Integer
Dim DatoAnterior As Integer
Dim Puerto As Integer

'Inicialización de variables
Private Sub Form_Load()
   'Escala que utilizará el gráfico
   Picture1.Scale (-8, 256)-(200, -15)
   DatoAnterior = 127
   Puerto=&H378      'Dirección del puerto
   Out Puerto+2, 32 'Habilita puerto bidireccional
   Inicializar_Click 'Prepara para lectura
End Sub

'Rutina para borrar el gráfico actual y empezar de nuevo
Private Sub Inicializar_Click()
        Picture1.Cls
        'Genera los ejes X y Y en el gráfico
        Picture1.Line (-1, -2)-(200, -2)
        Picture1.Line (0, -2)-(0, 256)

       'Escribe los valores del eje X
      For j = 0 To 200 Step 20
        Picture1.Line (j - 4, -4)-(j - 3, -4), &HC0C0C0
        Picture1.Print j
    Next j

     'Escribe los valores del eje Y
     For j = 0 To 255 Step 20
        Picture1.Line (-8, j)-(-8, j + 5), &HC0C0C0
        Picture1.Print j
     Next j

    i = 0
    Continuar_Click 'Inicia la lectura
End Sub

'Rutina que se ejecuta en forma periódica encargada de
'ejecutar las lecturas y dibujar el gráfico
Private Sub Timer1_Timer()
  dato(i) = Inp(Puerto)    'Lee el puerto
  'Dibuja una linea según el valor leído:
  Picture1.Line (i - 1, DatoAnterior)-(i, dato(i)), &HFF
  DatoAnterior = dato(i)
  Dato.Text = dato(i) 'Escribe dato en la caja de texto
  'Escribe intervalo en ms en la caja de texto:
  If Timer1.Interval <> 0 Then Timer1.Interval =
                                        Intervalo.Text
  If i = 200 Then    'Si finaliza el gráfico..
    Inicializar_Click
  End If

  'Manejo del relevo
  If i <> 0 And i <> 200 Then
  If dato(i) > Val(VrDisparo.Text) Then
    If Invertido.Value = 0 Then
        Out Puerto+2, 36          'Enciende relevo
        Luz.FillStyle = 0
        Label3.Caption = "ON"
    Else: Out Puerto+2, 32        'Apaga relevo
        Luz.FillStyle = 1
        Label3.Caption = "OFF"
    End If
  Else:
    If Invertido.Value = 0 Then
        Out Puerto+2, 32          'Apaga relevo
        Luz.FillStyle = 1
        Label3.Caption = "OFF"
    Else: Out Puerto+2, 36  'Enciende relevo
        Luz.FillStyle = 0
        Label3.Caption = "ON"
    End If
  End If
  End If
   i = i + 1
End Sub

'Botón que deshabilita la rutina Timer1
Private Sub Detener_Click()
Timer1.Interval = 0
End Sub

'Botón que habilita la rutina Timer1
Private Sub Continuar_Click()
    Timer1.Interval = Intervalo.Text
End Sub

'Botón para cerrar la aplicación
Private Sub Cerrar_Click()
    Unload Me
End Sub
```

Figura 8. Lista de programa en Visual Basic

```
//Declaración de librerías y variables
#include <dos.h>
#include <stdio.h>
#include <conio.h>

#define puerto 0x378
int dato,disparo;

//Rutina principal
void main()
{
     clrscr();
     //Referencia para el encendido del relevo
     disparo=120;
     //Habilitar el puerto bidireccional
     outportb(puerto+2,32);
     printf("\n\nLectura obtenida en el puerto:  ");

     //Ciclo para realizar las lecturas
     do{
       dato=inportb(puerto);       //Lee el puerto
       //Enciende o apaga el relevo según el dato
       if (dato>disparo) outportb(puerto+2,36);//Enciende
            else outportb(puerto+2,32);       //Apaga
       gotoxy(33,3);
       printf("%d ",dato);
       delay(500);          //Retardo en ms entre lecturas
     }while (!kbhit());//Repite rutina
}
```

Figura 9. Listado del programa en lenguaje C

En la casilla de intervalo entre lecturas se escribe el tiempo en ms que transcurrirá entre lectura y lectura. En la sección de control del relevo, se escribe el valor límite en el cual el relevo se encenderá o se apagará de acuerdo al valor leído en el puerto. La casilla de verificación se utiliza para cambiar la lógica de operación de dicho relevo es decir, si se debe apagar o encender por debajo del valor límite y viceversa.

Rutina en lenguaje C. La rutina que hemos elaborado en lenguaje C, figura 9, sirve sólo para comprobar el funcionamiento del circuito. Este programa configura inicialmente el puerto como bidireccional y luego entra en un ciclo en el cual hace lecturas consecutivas del puerto y muestra el valor obtenido en la pantalla. Para el control del relevo se ha puesto una línea que hace la comparación con un valor indicado previamente en el encabezado del programa. Dependiendo del resultado, se energiza o apaga dicho relevo.

En un próximo artículo haremos una pequeña aplicación de control a través de este mismo circuito. Dudas y sugerencias, escriba a la dirección **gui.ramos@usa.net** del correo electrónico.

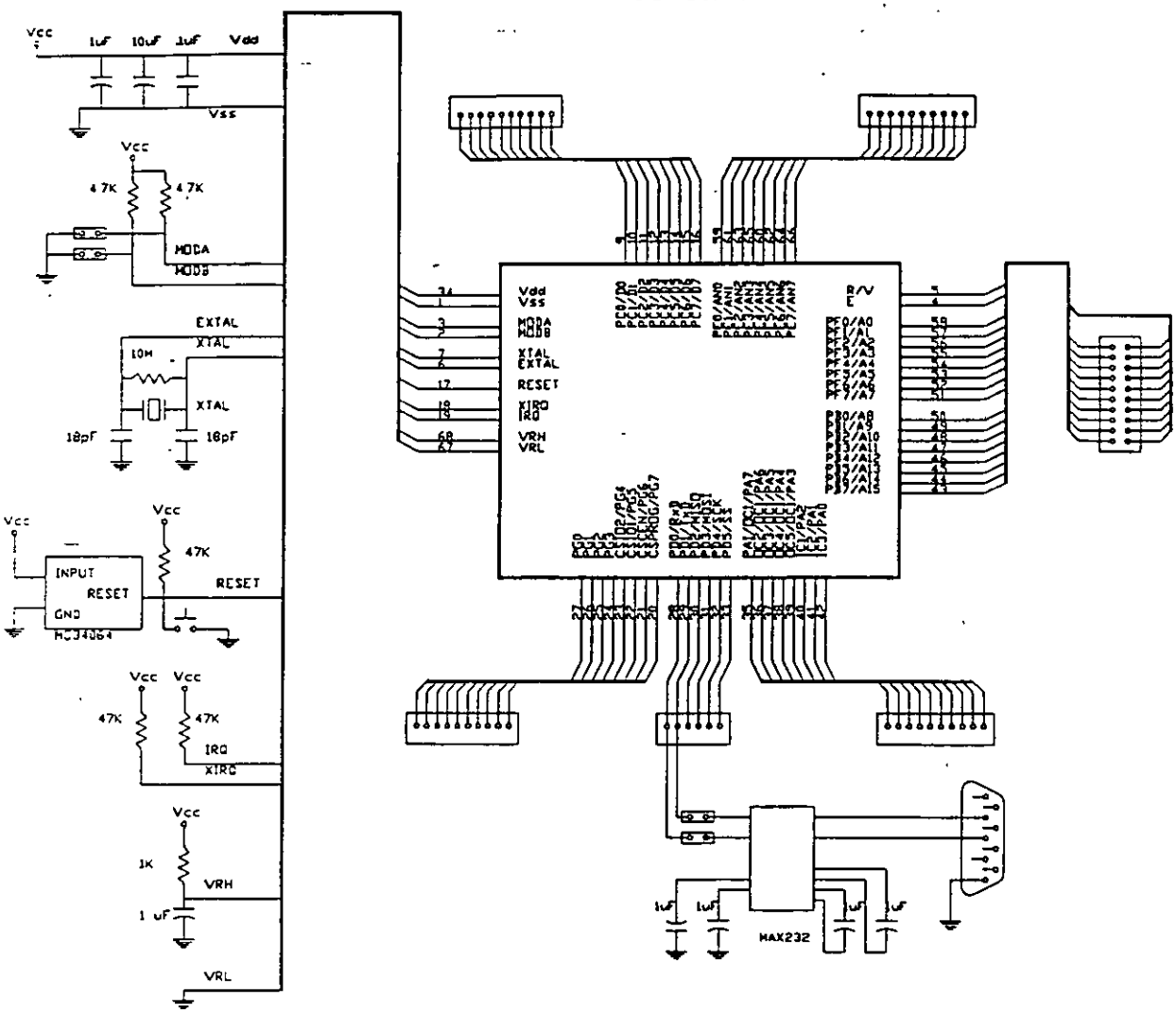
FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

SISTEMA MÍNIMO 68 HC11F1

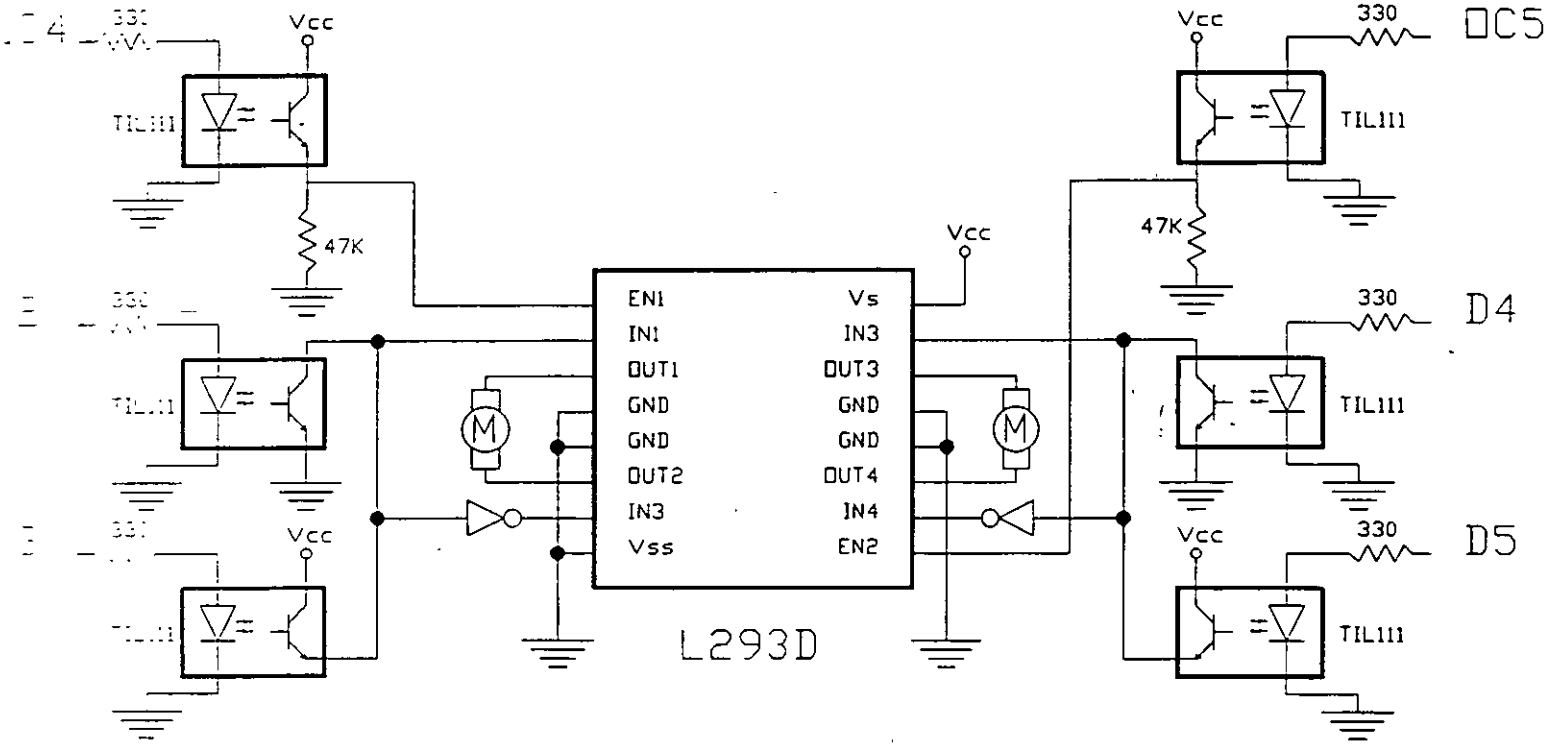
PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ
PALACIO DE MINERÍA
ABRIL DE 1999

SISTEMA MINIMO
68HC11F1



1 BASE PARA MICROCONTROLADOR HC11F1 (68 pines)
1 MICROCONTROLADOR HC11F1
1 CRISTAL DE 8 MHz
1 C. I. Max 232
1 CONECTOR DB9
5 RES DE 47 KΩ
1 RES DE 10 MΩ
2 CAP. DE 18pf
2 CAP DE 1μf
1 cap de 10μf
1 CAP DE 0.1μf
1 BOTON RESET N/A
1 C. MC34064

CONTROL DE MOTORES DE CORRIENTE DIRECTA (1 Amp. max.)



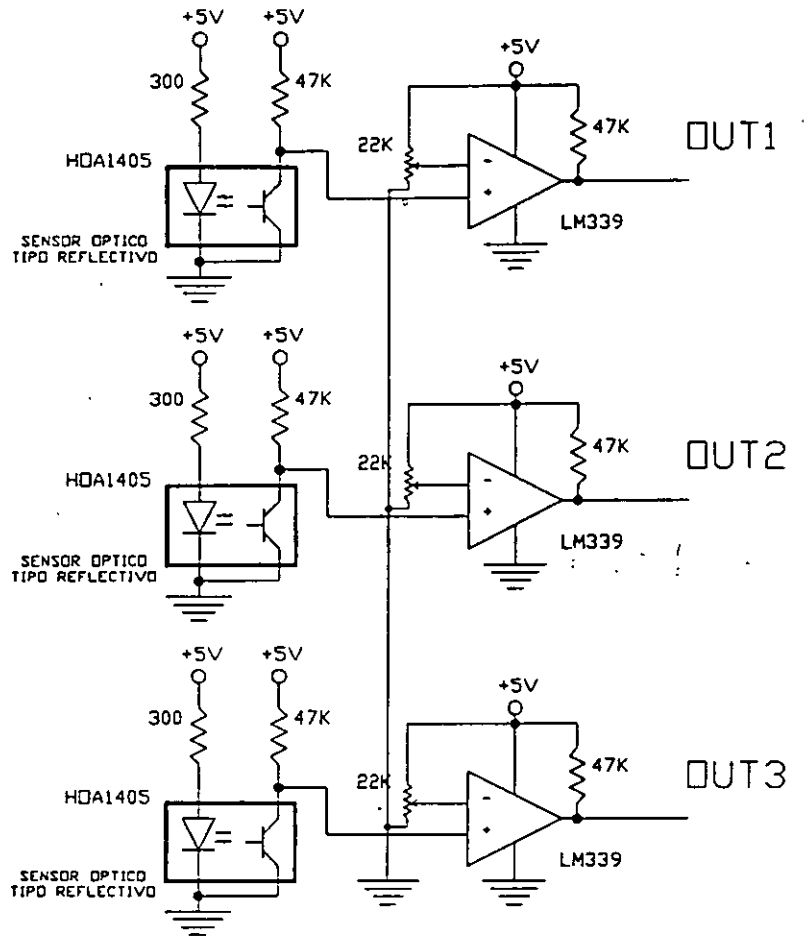
6 OPTO ACOPLADORES TIL111

1 CIRCUITO INTEGRADO L293D

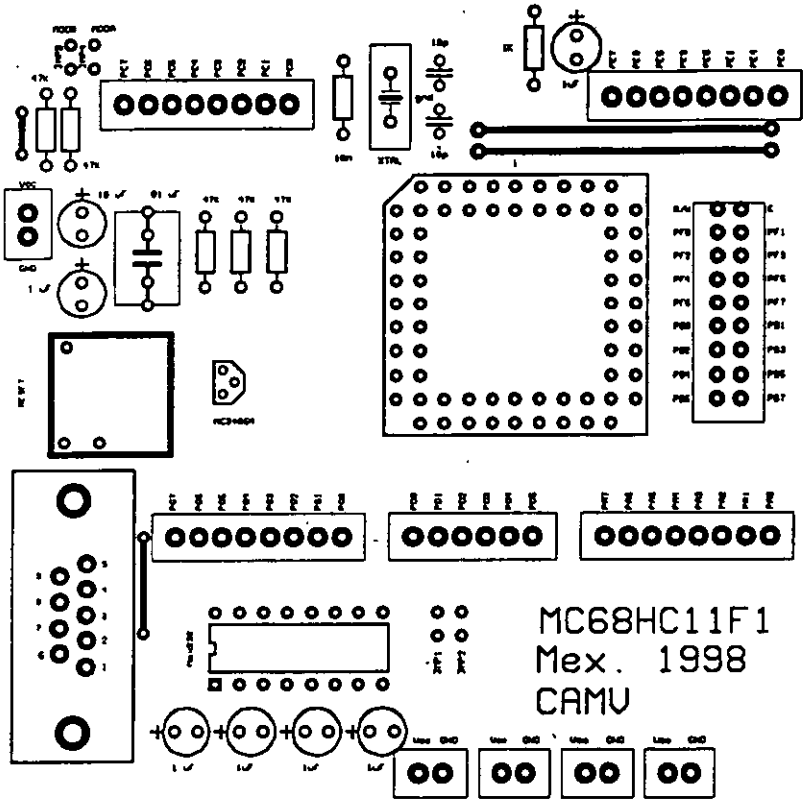
6 RESISTENCIAS DE 330 Ω

62 RESISTENCIAS DE 47 KΩ

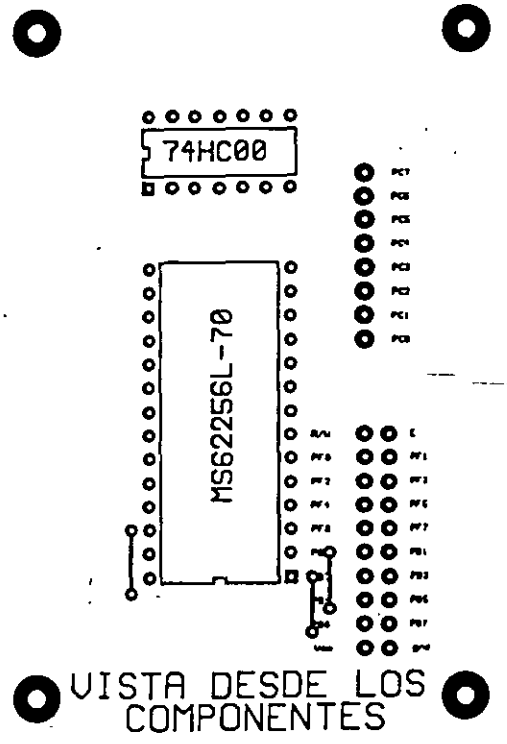
ETAPA DE SENSADO DE NIVEL DE INTENSIDAD DE LUZ



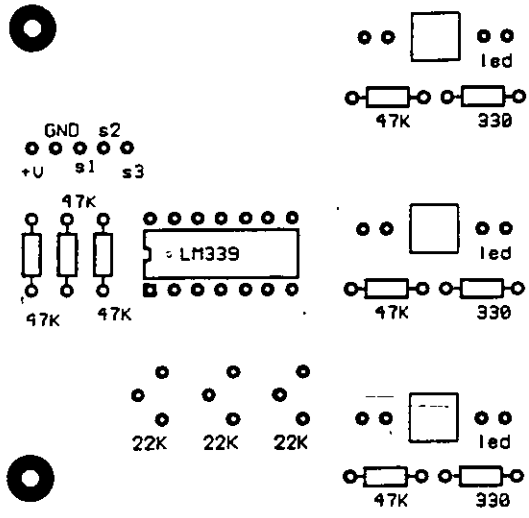
3 SENSORES DE EFECTO REFLECTIVO HOA1405
1 COMPARADOR DE VOLTAJE LM339
3 POTENCIÓMETROS MINI-22 K
3 RESISTENCIAS DE 300 Ω
6 RESISTENCIAS DE 47 KΩ



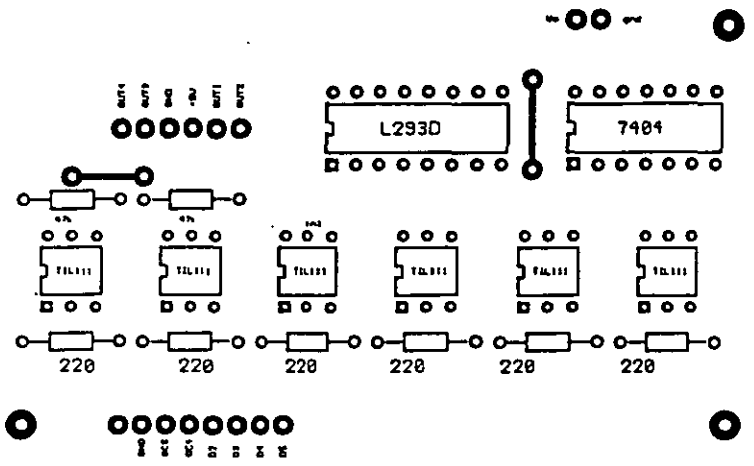
VISTA DESDE LOS
COMPONENTES

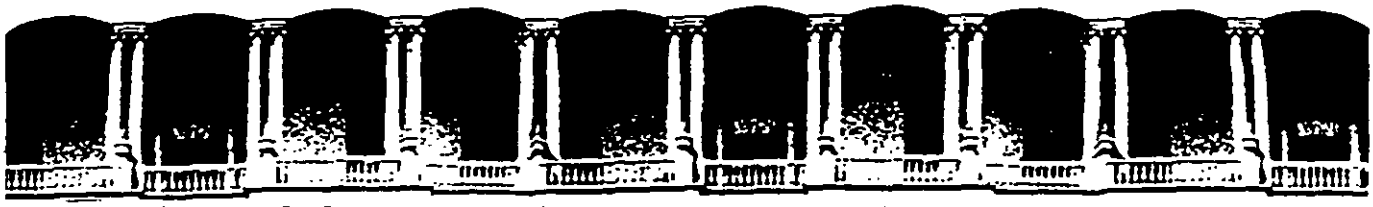


VISTA DESDE LOS COMPONENTES



vista desde los componentes


L293D
7404
220
220
220
220
220
220



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA
CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

ACS ROBOTICS SIG.

LEGO WORKSHOP 1 PHOTOS

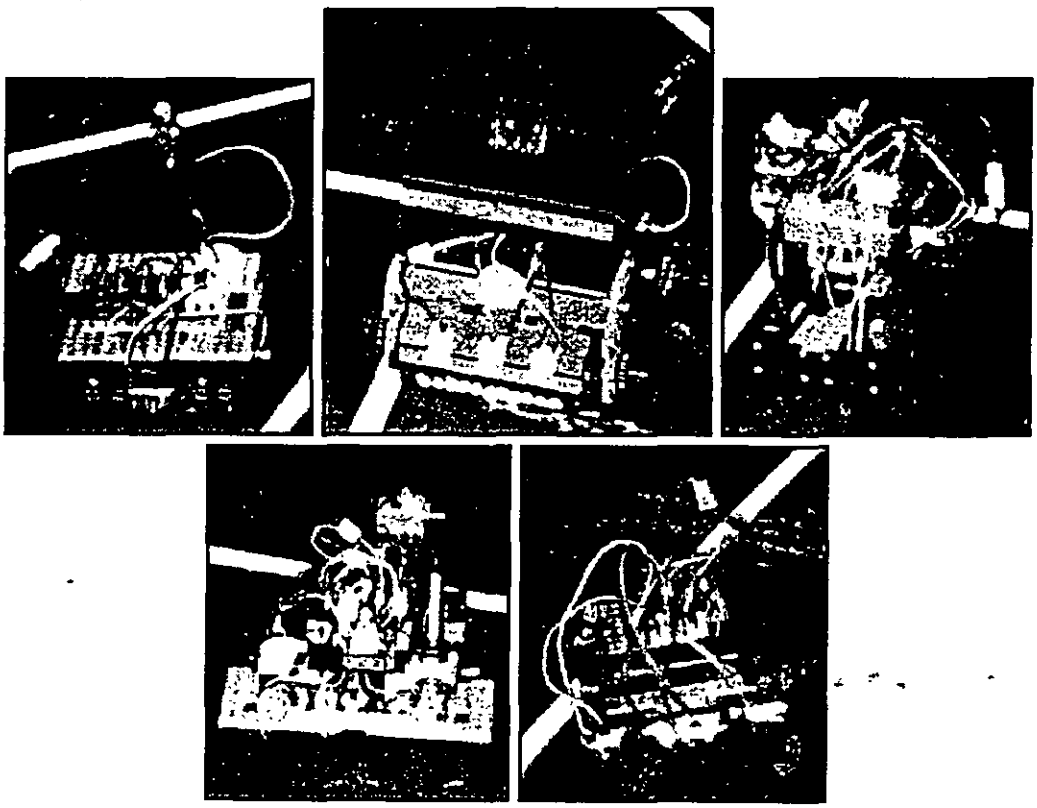
PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALES AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ

ACS ROBOTICS SIG

LEGO Workshop 1 Photos

Each picture here is a link to the original photo.

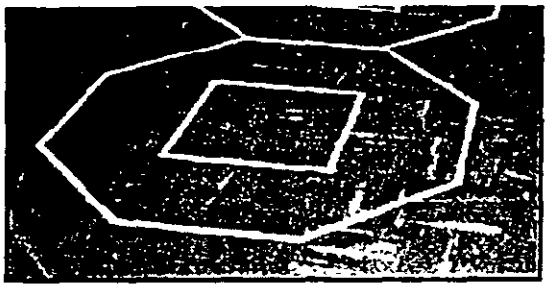
The Robots



The Event

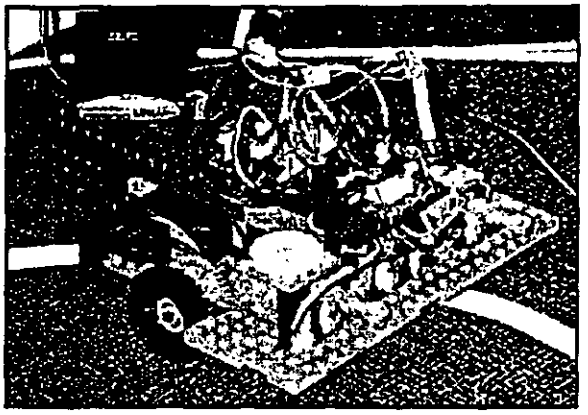
The track

The robots had to navigate the track in a figure 8, following the outer line. If they



drifted off into a center square, they were disqualified.

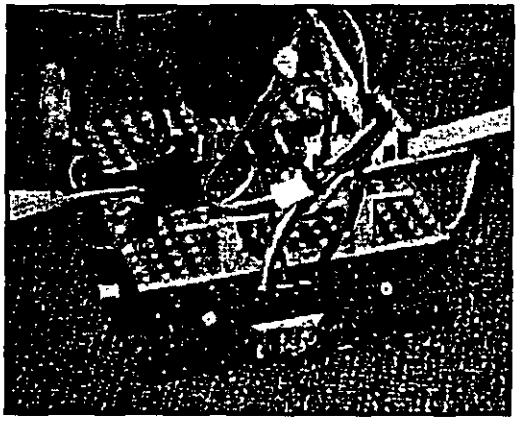
Two robots were placed on the track at a time, starting at opposite ends. Collision detection and handling were thus added to the problems of following the white line in the correct pattern.



The equipment

The robot's "brain" was a 68HC11 chip in a basic Rugwarrior-like board. The robot was powered by rechargable Nicad batteries.

Inputs to the board were a serial cable with modular plug (from the computer) and a power input (to recharge the battery). The chip was programmed with Interactive C.



The sensors

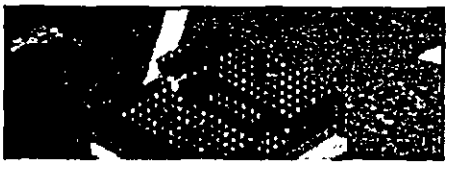
The robot sensed the white line through a combination of IR emitters and detectors. In addition there was a larger IR detector for detecting the start signal. There was also a bump switch, which could be used for collision detection.

Outputs from the board included some LEDs for diagnosis and a speaker.

Navigating the track

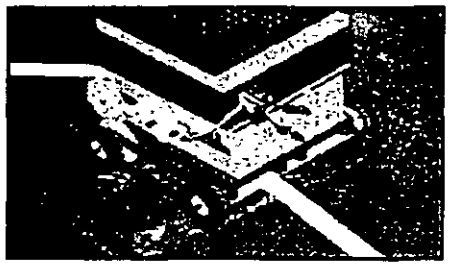
The robots had two LEGO motors each, which could be driven in forward or reverse independently. Information gathered by the sensors was processed to determine motor actions, thus producing appropriate behaviour.

If a robot lost the white line, it might be able to pick it up again (but not always facing the right direction).



Handling a junction

The junctions posed two problems - detecting a fork in the line, and correctly



deciding which fork to take. A robot which took the wrong fork was deemed disqualified.

The robots were to do three laps of the track, which meant 6 junction decision points had to be passed correctly.

Last Change: 9th July 1996
Author: Tracy Lightfoot <T.Lightfoot@mailbox.gu.edu.au>

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

APENDICE A

MICROCONTROLADOR MC68C11E9

PROFESORES: DR. JESUS SAVAGE CARMONA
M. en I. MARCO ANTONIO MORALÈS AGUIRRE
ING. RUBEN ANAYA GARCÍA
ING. CARLOS MUNIVE VÁZQUEZ
PALACIO DE MINERÍA
ABRIL DE 1999

APENDICE A
MICROCONTROLADOR MC68HC11E9

El MC68HC11E9 es un semiconductor de alta densidad de óxido de metal complementario (HCMOS), unidad microcontroladora (MCU). Esta MCU tiene un voltaje de operación bajo, de alta velocidad, con un bus multiplexado de una velocidad nominal de 2MHZ.

Las características especiales de este microcontrolador son:

- Cinco puertos paralelos, con doble función cada uno de ellos
- Expansión de sistemas de tiempo, con cuatro etapas preescalables
- Se resalta el no retorno a cero (NRZ)
- Interface de Comunicación Serial (SCI)
- Convertidor Analógico Digital de ocho canales, con ocho bits de resolución
- Bloque protector de mecanismo para EEPROM y CONFIG
- Bus multiplexado
- Empaquetado de 58 pines
- Tope para el ahorro de energía y modos de espera
- 64 Kbytes de memoria direccionable
- Interface de Comunicación con Periféricos (SPI)
- 512 Bytes de EEPROM
- 512 Bytes de RAM
- 12 Kbytes de ROM, con el Buffalo programado
- Circuito de Interrupción real
- Acumulador de pulsos
- Captura de entrada
- Comparación de salida

Modos de operación de HC11

Emplea dos pines, para seleccionar el modo de operación, el MODA y el MODB, básicamente existen dos modos de operación normales, que son el single chip y el expandido, y dos modos de operación especiales, el bootstrap y el test. La selección del modo de operación es de acuerdo a los valores codificados en estos pines.

MODA	MODB	Modo seleccionado
0	1	Normal Single Chip
1	1	Normal Expandido Multiplexado
0	0	Especial Bootstrap
1	0	Especial Test

Single Chip

Solo se dispone de la memoria interna del microcontrolador. Los puertos B y C, así como las terminales STRA y STRB están disponibles como entrada y salida paralelas de propósito general; todo el software necesario para controlar el MCU está contenido en los recursos internos.

Expandido

En el modo de operación expandido, permite accesar a la memoria externa dentro de la totalidad de los 64 Kbytes de espacio direccionable, el espacio incluye al espacio de memoria interna del modo single chip, los puertos B y C se emplean para realizar la expansión de memoria, el puerto B formará la parte alta del bus de direcciones y el puerto C tendrá la función multiplexada para la parte baja del bus de direcciones y el bus de datos, además incluye las señales AS, E y R/W.

Test

Permite el acceso privilegiado a recursos internos del MCU, este es una variación del modo expandido y es normalmente empleado para pruebas internas de producción por el fabricante.

Bootstrap

Es una variación especial del modo single chip, este permite dar entrada a programas en la RAM interna. Al seleccionarse este modo de operación, durante el restablecimiento una pequeña ROM de boostrap se hace presente en el mapa de memoria. Esta contiene un pequeño programa el cuál inicializa la interface serial asíncrona de comunicación (SCI), lo cuál permite al usuario cargar programas dentro de la RAM interna, finalizando esto, el control lo tendrá el programa recién cargado. Este es el modo de operación en el cual se trabaja con mayor frecuencia.

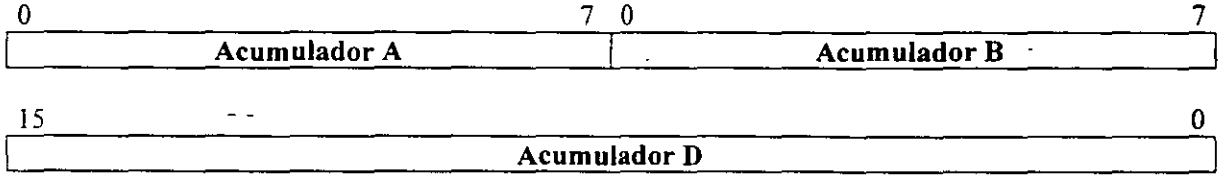
Modelo del Programador

La familia del MC68HC11 tiene ocho registros de unidades centrales de proceso, disponibles para el programador. Cada registro es explicado en los siguientes párrafos.

Acumuladores (A, B y D)

Los acumuladores A y B son registros de 8 bits, de propósito general, utilizados para almacenar los operandos y los resultados de las operaciones lógicas o aritméticas o para la manipulación de información.

Estos dos acumuladores concatenados forman el doble acumulador que será de 16 bits de longitud, donde el acumulador B es la parte baja y el acumulador A es la parte alta de este doble acumulador.



Registro de índice (IX)

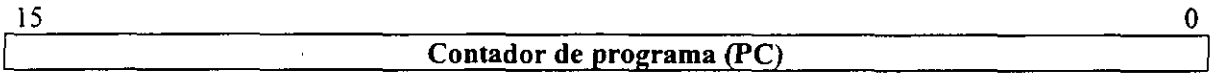
El registro IX es un registro de 16 bits, donde su utilidad se encuentra principalmente para el modo de direccionamiento indexado. Provee unos 16 valores que pueden ser sumados a un offset de ocho bits para accesar una dirección efectiva. Este registro puede ser también utilizado para realizar tiempos de retardo o como un área de almacenamiento temporal.

Registro de índice Y (IY)

Es un registro de 16 bits, utilizado para el modo de direccionamiento indexado, similar al del registro IX. Sin embargo la mayoría de las instrucciones donde se emplea el registro IY, tiene dos bytes y requiere un byte extra de código de máquina y un ciclo extra al momento de su ejecución.

Contador de programa (PC)

El contador de programa PC es un registro de 16 bits, contiene la dirección de la siguiente instrucción que será ejecutada.



Apuntador a la pila (SP)

El stack pointer SP es un registro de 16 bits que contiene la dirección de la siguiente ubicación libre en el stack. El stack es configurado como una secuencia de últimos en entrar, primeros en salir (LIFO), lee registros de escritura que permiten datos importantes para ser almacenados durante interrupciones y llamados a subrutinas.

Registro de índice (IX)

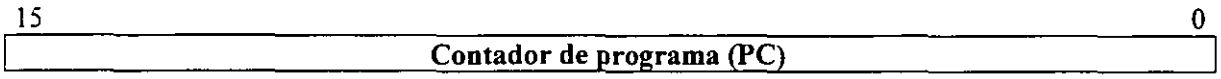
El registro IX es un registro de 16 bits, donde su utilidad se encuentra principalmente para el modo de direccionamiento indexado. Provee unos 16 valores que pueden ser sumados a un offset de ocho bits para accesar una dirección efectiva. Este registro puede ser también utilizado para realizar tiempos de retardo o como un área de almacenamiento temporal.

Registro de índice Y (IY)

Es un registro de 16 bits, utilizado para el modo de direccionamiento indexado, similar al del registro IX. Sin embargo la mayoría de las instrucciones donde se emplea el registro IY, tiene dos bytes y requiere un byte extra de código de máquina y un ciclo extra al momento de su ejecución.

Contador de programa (PC)

El contador de programa PC es un registro de 16 bits, contiene la dirección de la siguiente instrucción que será ejecutada.



Apuntador a la pila (SP)

El stack pointer SP es un registro de 16 bits que contiene la dirección de la siguiente ubicación libre en el stack. El stack es configurado como una secuencia de últimos en entrar, primeros en salir (LIFO), lee registros de escritura que permiten datos importantes para ser almacenados durante interrupciones y llamadas a subrutinas.

Registro de banderas (CCR)

El registro de banderas es un registro de 8 bits, en el cuál cinco bits son utilizados para indicar los resultados de la instrucción recién ejecutada, dos bits mascarables de interrupciones y un bit de paro. Estas banderas pueden ser checadas individualmente por el programa y realizar una tarea específica como resultado de su estado; cada bit se explica a continuación.

7	6	5	4	3	2	1	0
S	**X**	**H**	**I**	**N**	**Z**	**V**	**C**

(C) Acarreo (Carry/Borrow)

El bit C indica si en la última operación realizada, ya sea lógica o aritmética existió un acarreo, esta bandera actúa también como una bandera de error para las operaciones de multiplicación y división, así mismo se afecta durante las operaciones de rotación de los registros.

(V) Desbordamiento (Overflow)

El bit V indica si ocurrió un sobreflujo como resultado de una operación.

(Z) Cero (Zero)

Se fija cuando el resultado de la última operación lógica, aritmética o manipulación de datos es cero.

(N) Negativo (Negative)

El bit N indica si el resultado de la última operación aritmética, lógica o manipulación de datos es negativo, es decir, refleja el estado del bit más significativo (MSB) de un resultado. Un número es positivo cuando el MSB es cero y es negativo cuando MSB es uno.

(H) Medio Acarreo (Halft Carry)

Indica el acarreo existente en una operación aritmética entre los bits tercero y cuarto; es de utilidad cuando se desea hacer el ajuste a decimal del resultado de una suma.

(I) Interrupción de Máscara

Este bit puede ser activado por software y hardware para habilitar todas las fuentes de interrupción de máscara.

(X) Interrupción de Máscara

El bit X se activa solamente por hardware (RESET o XIRQ) y limpiado solamente por el programa de instrucciones de transferencia de A hasta CCR o retornando de interrupciones (RTI).

(S) Inhabilitador de Paro

Este bit permite habilitar o deshabilitar la instrucción de stop.

Modos de Direccionamiento

El MCU cuenta con seis modos de direccionamiento, inmediato, directo, extendido, indexado, inherente y relativo, que son utilizados para obtener o almacenar un dato, de o hacia alguna localidad de memoria.

Modo de Direccionamiento Inmediato (INM)

En el modo de direccionamiento inmediato, el argumento actual está contenido en el byte(s) que siguen a la instrucción.

Formato: ***Instrucción #Dato***

Ejemplo:

```
LDAA #$A0          ; Carga el dato $A0 en hexadecimal al acumulador A
LDAB #%01100011    ;Carga el dato binario indicado al acumulador B
```

Modo de Direccionamiento Directo (DIR)

En el modo de direccionamiento directo, el byte menos significativo de la dirección efectiva de la instrucción aparecerá en el byte siguiente al opcode. El byte más significativo de la dirección efectiva es tomado como $00. Este modo de direccionamiento hace referencia al espacio de memoria comprendido entre $0000 y $00FF, es conocido como direccionamiento página cero.

Formato: ***Instrucción $00Dirección***

Ejemplo:

```
LDAA #$A0    ;Carga el dato indicado al acumulador A
STAA $50     ;Envía el contenido de A a la dirección $50
```

Modo de Direccionamiento Extendido (EXT)

La dirección efectiva de la instrucción aparece explícitamente en los dos bytes siguientes al opcode. Por lo tanto se puede hacer referencia al área total del microcontrolador es decir de la dirección $0000 a la $FFFF.

Formato: ***Instrucción $Dirección***

Ejemplo:

```
LDAA #$01FF ;Carga el dato indicado al acumulador A
STAA $1004  :Envía el contenido del acumulador A en la dirección $1004
```

Modo de Direccionamiento Indexado (INDX, INDY)

Los registro de índice X o Y son empleados para hacer referencia a la dirección efectiva donde se obtendrá o almacenará el dato. La dirección efectiva es variable y depende del contenido actuál del registro de índice a emplear y de un OFFSET sin signo, contenido en la instrucción.

Formato: ***Instrucción*** ***$Offset,Reg_índice***

Ejemplo:

```
LDX #$1000   ;Inicializa el registro de índice X
LDAB $31,X   ;Carga el contenido de la dirección $1031 en B
```

Modo de Direccionamiento Inherente (INH)

En este modo de direccionamiento, toda la información necesaria para ejecutar una instrucción, está contenida en el código de operación.

Formato: ***Instrucción***

Ejemplo:

```
INX          ;Incrementa el contenido del registro X
ABA          ;Suma el contenido de los acumuladores A y B
```

Modo de Direccionamiento Relativo (REL)

Su utilidad se encuentra en los saltos o brincos relativos, comúnmente ejecutados después de una comparación o lectura del CCR; para lo cuál es posible la toma de decisiones de bifurcación. Las instrucciones de bifurcación generan dos bytes, uno para el opcode y el otro para el offset relativo. Si la condición de bifurcación es verdadera, el contenido de los 8 bits del byte siguiente al opcode (offset) son sumados al contenido del contador de programa para formar la dirección efectiva de la bifurcación, de otra manera, el control pasa a la instrucción siguiente a esta.

Formato: ***Instrucción Etiqueta***

Ejemplo:

```
LDAA $100A   ;Carga en A el contenido de la dirección $100A
CMPA #$F0    ;Compara A con el dato #$F0
BEQ igual    ;Si Z=1 brinca a igual
BNE diferente ;Si no es igual brinca a diferente
```

Puertos Paralelos

En la serie E del microcontrolador MC68HC11, se disponen de 5 puertos paralelos, y un total de 38 pines de entrada y salida; a continuación se presentas estos:

Puerto	Pines de entrada	Pines de salida	Pines Bidireccionales	Función alterna
Puerto A	3	3	2	Temporizador
Puerto B	-	8	-	Parte alta del bus de direcciones
Puerto C	-	-	8	Parte baja del bu de direcciones/ bus de datos
Puerto D	-	-	6	SCI y SPI
Puerto E	8	-	-	Convertidor A/D

Puerto A

Al puerto A no le afecta el modo de operación, viene configurado para esta versión del microcontrolador de la siguiente manera, tres pines de salida, tres de entrada y dos que se pueden seleccionar como entrada o salida.

PORTA Datos de Puerto A $1000

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAI	OC2	OC3	OC4	IC4/0C5	IC1	IC2	IC3

Puerto B

En single chip y bootstrap todos los pines del puerto B son salidas de propósito general; en el modo expandido y test, forman la parte alta del bus de direcciones.

PORTB Datos del Puerto B $1004

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
ADR15	ADR14	ADR13	ADR12	ADR11	ADR10	ADR9	ADR8

Puerto C

En single chip y bootstrap, el puerto C se puede configurar para que trabajen sus pines como salida o entrada, dependiendo de la selección realizada en el registro DDRC. En el modo expandido y test, el puerto C tiene la función multiplexada para la parte baja del bus de direcciones y el bus de datos.

DDRC Registro de Dirección de Datos del Puerto C $1007

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0

DDC(7:0) Dirección de datos para el puerto C

0 = Bit seleccionado como entrada
1= Bit seleccionado como salida

PORTC Datos del puerto C $1003

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
ADR7 DATA7	ADR6 DATA6	ADR5 DATA5	ADR4 DATA4	ADR3 DATA3	ADR2 DATA2	ADR1 DATA1	ADR0 DATA0

Puerto D

Al puerto D o le afecta el modo de operación, pero únicamente se dispone de 6 bits de propósito general de entrada o salida, seleccionados por la configuración del registro DDRD; tiene su doble función con los subsistemas SCI y SPI.

DDRD Registro de Dirección de Datos del puerto D $1009

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0

PORTD Datos del puerto D $1008

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		PD5	PD4	PD3	PD2	PD1	PD0
		PD5	PD4	PD3	PD2	PD1	PD0
		SS	SCK	MOSI	MISO	TxD	RxD

Puerto E

El puerto E es de 8 bits de entrada general, no le afecta el modo de operación, comparte su función con el convertidor analógico digital.

PORTE Datos del Puerto E $100A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0

Convertidor Analógico Digital (A/D)

El convertidor analógico digital utilizado en el HC11 es un convertidor de aproximaciones sucesivas, usa la técnica de restribución de carga capacitiva para convertir señales analógicas a valores digitales. El sistema A/D es un convertidor de ocho canales, 8 bits y entradas multiplexadas está compuesto por cuatro bloques funcionales multiplexor, convertidor analógico, control digital y almacenamiento de resultados.

Multiplexor. El multiplexor selecciona una de las entradas para conversión. La selección de entrada es controlada por el valor de los bits CD, CC, CB y CA en el registro ADCTL.

Convertidor Analógico. La conversión de una entrada analógica seleccionada por el multiplexor ocurre en este bloque. Contiene el arreglo de un Capacitor Digital Analógico (DAC), un comparador y un registro de aproximaciones sucesivas (SAR). Cada conversión es una secuencia de 8 operaciones de comparación, comenzando con el bit menos significativo (MSB). Cada comparación determina el valor de un bit en el registro de aproximaciones sucesivas.

Control Digital. Todas las operaciones del convertidor A/D son controladas por bits en el registro ADCTL. Los bits de este registro indican el estado de la conversión, el control de la conversión, el control de las conversiones simples o continuas, y si las conversiones se harán sobre canales simples o múltiples.

Registro de resultados. Existen cuatro registros de ocho bits cada uno para el almacenamiento de resultados (ADR1 - ADR4). Cada uno de estos registros debe ser accesado por el procesador de la CPU. La bandera de conversión completa (CCF) indica cuando un dato válido está presente en el registro de resultados.

Secuencia de Conversión

Las operaciones del convertidor A/D son ejecutadas en secuencias de cuatro conversiones cada una; una secuencia de conversión puede ser repetida continuamente o parar después de una iteracción. La bandera de CCF cambia después de las cuatro conversiones en una secuencia que muestra la disponibilidad de datos en el registro de resultados.

Modos de Operación

Existen dos modos de operación del convertidor Analógico Digital, el de canal sencillo y el de canales múltiples.

Operación sobre un canal simple.

Es la selección y operación de un solo canal del convertidor A/D, pudiéndose seleccionar cualesquiera de los ocho canales disponibles, esto es de acuerdo a la combinación de bits en el registro ADCTL, existen dos tipos de operaciones sobre un canal simple. Cuando SCAN=0, se realiza una conversión en 4 tiempos consecutivos. El primer resultado se almacena en el registro ADR1 y el cuarto resultado es almacenado en ADR4; después que las cuatro conversiones son completadas, el sistema se detiene hasta que un nuevo comando de conversión es escrito en el registro ADCTL. En el segundo tipo, cuando SCAN=1, las conversiones son efectuadas continuamente almacenándose en los registros ADR1 - ADR4 y sobre escribiéndose en los cuatro tiempos.

Operación sobre canales múltiples.

Es posible la selección y operación de cuatro canales del convertidor A/D al mismo tipo, configurando el primer bloque o el segundo bloque de cuatro entradas. Existen dos tipos de operaciones sobre canales múltiples. Cuando SCAN=0 , un grupo de cuatro canales son convertidos uno a la vez. El resultado es almacenado en ADR1 y el cuarto resultado es almacenado en ADR4 . Después de 4 conversiones, el sistema se detiene hasta que un nuevo comando de conversión es escrito en el registro ADCTL. En el segundo tipo, cuando SCAN=1 , las conversiones son efectuadas continuamente sobre el grupo de canales seleccionados, almacenándose en los registros ADR1-ADR4 y sobre escribiéndose cada tiempo.

Registros involucrados

OPTION - **$1039**

Bit 7							Bit 0
ADPU	CSEL		DLY				

ADPU: Habilita el Convertidor Analógico Digital
0 = Deshabilitado
1 = Habilitado

CSEL: Selección del Reloj
0= El A/D y la EEPROM usan el sistema de reloj E.
1= El A/D y la EEPROM usan el reloj interno RC.

DLY: Habilita el oscilador iniciando retardos.

Registro de Estado/Control del A/D

ADCTL Control del Analógico Digital $1030

Todos los bits de este registro pueden ser leídos o escritos, excepto CCF (bit 7), el cuál es solamente de lectura como indicador de estado.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit1	Bit 0
CCF	--	SCAN	MULT	CD	CC	CB	CA

CCC Bandera de conversión completa.
Este bit cambia cuando se tiene una conversión válida.
1 = Conversión completa

SCAN Control de rastreo continuo.
0 = SCAN sencillo
1 = SCAN continuo

MULT Control de canal simple/múltiple canal.
0 = canal simple
1 = múltiple canal

CD - CA Selección de los canales del convertidor A/D.

Si MULT= 0 (Selección de canal simple)

CD	CC	CB	CA	Canal seleccionado
0	0	0	0	AN0
0	0	0	1	AN1
0	0	1	0	AN2
0	0	1	1	AN3
0	1	0	0	AN4
0	1	0	1	AN5
0	1	1	0	AN6
0	1	1	1	AN7

Si MULT=1 (Selección de un bloque se cuatro canales)

CD	CC	Resultado
0	0	AN0 resultado en ADR1
		AN1 resultado en ADR2
		AN2 resultado en ADR3
		AN3 resultado en ADR4
0	1	AN4 resultado en ADR1
		AN5 resultado en ADR2
		AN6 resultado en ADR3
		AN7 resultado en ADR4

Registros de resultados (ADRx)

Estos registros son solamente de lectura que mantienen el resultado de la conversión en 8 bits. Los datos en los registros de resultados del convertidor A/D son válidos cuando ,la bandera de CCF se activa.

ADR1 Registro de Resultados 1 $1031

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

ADR2 Registro de Resultados 2 $1032

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

ADR3 Registro de Resultados 3 $1033

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

ADR4 Registro de Resultados 4 $1034

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Interface de Comunicación Serial (SCI)

El SCI permite al MCU ser eficientemente una interface con los dispositivos periféricos que permiten un formato de datos seriales en forma asíncrona. El SCI usa un estandar de no retorno a cero (NRZ) con una variedad de rangos de baudios derivados del circuito de reloj de cristal. Se utilizan los pines PD0 para recibir datos y PD1 para trasmitir; el circuito de generación de velocidades en bauds contiene un reloj programable, preescalador y divisor.

Formato de datos

El formato de los datos seriales requiere lo siguiente:

- Una línea desocupada en un estado de prioridad alta para la transmisión recepción de un mensaje.
- Un bit de comienzo (cero lógico) que es transmitido-recibido, indicando el comienzo de cada caracter.

- Datos que son transmitidos y recibidos primero con el bit menos significativo.
- Un bit de parada (uno lógico) usado para indicar el final de un frame) Un frame consiste de un bit de inicio, un caracter de 8 o 9 bits de datos y un bit de parada).
- Un break, el cual está definido como la transmisión o recepción de un cero lógico para algunos múltiples números de frames.

La selección de la longitud de la palabra es controlada por el bit M del registro SCCR1.

Operación de transmisión

El transmisor incluye un registro de transmisión de datos paralelos, llamados al SCDR y transmitidos a un buffer que pone los datos en forma serial. Los contenidos del registro serial pueden ser solo escritos a través del SCDR. Este doble sistema de buffer permite que un caracter sea cambiado serialmente mientras otro caracter está esperando en el SCDR para ser transferido. La salida del registro es aplicado al PD1 tan largo como la transmisión en progreso o el bit de habilitación de transmisión del registro de control de comunicaciones SCCR2 es almacenado.

Operación de recepción

En operaciones de recepción, la secuencia de transmisión es invertida. El dato es recibido en el registro serial y es transferido al registro receptor de datos paralelos SCDR como una palabra completa. Este sistema de doble buffer permite a un caracter ser enviado serialmente mientras otro caracter es preparado en el SCDR.

Características de Wakeup

Las características del wakeup reduce los servicios SCI superiores en sistemas múltiples de recepción. El sofrware para cada receptor evalúa el primer caracter de cada mensaje. Si el mensaje es interpretado por un receptor deferente, el SCI puede colocarse en modo de pausa, parando el resto del mensaje para generar requerimientos para el servicio. En cualquier lugar que comience un nuevo mensaje, las causas lógicas de la pausa de los receptores es para reiniciar y evaluar el caracter inicial del nuevo mensaje.

Dos métodos de wakeup son aceptables, línea lenta de wakeup o dirección de marca; en la línea lenta de wakeup, un receptor en pausa se reanuda así es como pronto la línea R/D llega a ser lenta. En la dirección de marca del wakeup, uno de los MSB de un caracter es usado para indicar que el mensaje es una dirección que reanuda todos los receptores en pausa.

Registro Involucrados en el SCI

SCDR Registro de Comunicación de Datos Serial $102F

Es el registro donde se almacena el dato, con la doble función, ya sea como transmisor o receptor de información.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

SCCR1 Registro de Control 1 de Comunicación Serial $102C

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R8	T8		M	WAKE			

R8: Bit 9 de recepción

T8: Bit 9 de transmisión

M: Longitud del carácter SCI
0: Un bit de inicio, ocho bits de datos y un bit de paro
1: Un bit de inicio, nueve bits de datos y un bit de paro

WAKE: Selector de método de wakeup

1: Marca de dirección
0: Línea lenta

SCCR2 Registro de Control 2 de Comunicación Serial $102D

El registro SCCR2 provee los bits de control que habilita o deshabilita las funciones de interrupción.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

TIE Habilita la interrupción de transmisión
0 = Deshabilita interrupciones TDRE
1 = Solicitud de interrupciones de SCI cuando el estado de la bandera TDRE cambia

TCIE Habilita interrupciones para completar transmisión
0 = Deshabilita interrupciones TC
1 = Solicitud de interrupción de SCI cuando el estado de bandera TC cambia

RIE Habilita interrupciones de recepción
0 = Deshabilita interrupciones RDRF y OR
1 = Solicitud de interrupción para RDRF u OR

ILIE Habilita interrupciones de línea desocupada
0 = Deshabilita interrupciones IDLI
1 = Solicitud de interrupción de SCI cuando el estado de bandera IDLE cambia

TE Habilita transmisor
0 = Deshabilita transmisor
1 = Habilita transmisión

RE Habilita receptor
0 = Deshabilita recepción
1 = Habilita recepción

SCSR Registro de Estado de Comunicación Serial $102E

El SCSR indica el estado del puerto serial.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

TDRE Registro de transmisión de datos desocupados
0 = SCDR activo
1 = SCDR desocupado

TC Bandera de transmisión completa
0 = Transmisor activo
1 = Transmisor desocupado

RDRF Registro de recepción de datos llenos
0 = Registro SCDR vacio
1 = Registro SCDR lleno, indica que se puede obtener el dato

BAUD Registro de Velocidad $102B

Este registro es empleado para seleccionar diferentes rangos de velocidad, que pueden ser usados como control para la recepción o la transmisión.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0

Interface de comunicación serial con Periféricos

El SPI es un sistema serial de entradas y salidas de alta velocidad. El SPI (Interface Serial Periférica) puede ser usado para una expansión simple de entrada/salida o para permitir que muchos MCUs sean interconectados en una configuración multimaestra. La fase de reloj, polaridad es programable por software para permitir una directa compatibilidad con un gran número de dispositivos periféricos.

Cuatro líneas de señales básicas están asociadas con el sistema SPI: El Maestro Salida – Esclavo Entrada (MOSI), el Maestro Entrada – Esclavo Salida (MISO), el reloj serial (SCK) y la Selección de Esclavo (SS). Las señales del SPI son asignadas a los bits de los puertos D (5-2).

El SPI es un doble buffer para lectura, pero no para escritura. Si una escritura es intentada mientras un dato es transmitido, la transmisión es interrumpida. Esta condición causará en el bit de colisión de escritura del SPSR al ser almacenado.

En el modo maestro, el pin SCK es una salida. Dependiendo del bit CPOL del registro SCP, el SCK es lento o bajo, hasta que el dato es escrito al registro de cambio. Una vez escrito el dato, ocho pulsos son generados para cambiar los ocho bits del dato.

En el modo esclavo, los receptores lógicos comienzan una lógica delta en el pin SS y en la entrada del reloj. Así, el esclavo es sincronizado con el maestro. Los datos del maestro son recibidos serialmente en la línea MOSI de esclavo y carga los ocho bits del cambio del registro.

Registros involucrados

Registro de Datos de Entrada/Salida Periférica SPDR $102A

Registro de Control Serial Periférica SPCR $1028

Registro de Estado Serial Periférica SPSR $1029

Sistema Temporizador (TIMER)

El temporizador incluye los siguientes subsistemas:

- Temporizador, preescalador y contador
- Función de comparación de salida
- Función de captura de entrada
- Acumulador de pulsos

Temporizador, prescalador y contador

Es la base de todas las funciones de tiempo, así como controla la velocidad de operación del microcontrolador. Contiene dentro de los registros asociados a esta función, un preescador divisible entre 1, 4, 8 y 16 el tiempo base.

Cuando el contador TCNT cambia del valor $FFFF a $0000, entonces el temporizador activa la bandera de sobreflujo, con la cuál puede habilitar una interrupción.
Registros asociados:

TCNT Cuenta de temporizador $100E, $100F

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit2	Bit 1	Bit 0

TFLG2 Bandera de Interrupción del temporizador 2 $1025

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TOF							

TOF Bandera de sobreflujo del temporizador
Se activa cuando TCNT cambia de $FFFF a $0000

TMSK2 Máscara de Interrupciones del Temporizador 2 $1024

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TOI						PR1	PR0

TOI Habilitación de Interrupción cuando ocurra un sobreflujo

PR1:PR2 Selección del preescalador

PR1	PR0	Divide E por
0	0	1
0	1	4
1	0	8
1	1	16

Función de Salidas de Comparación

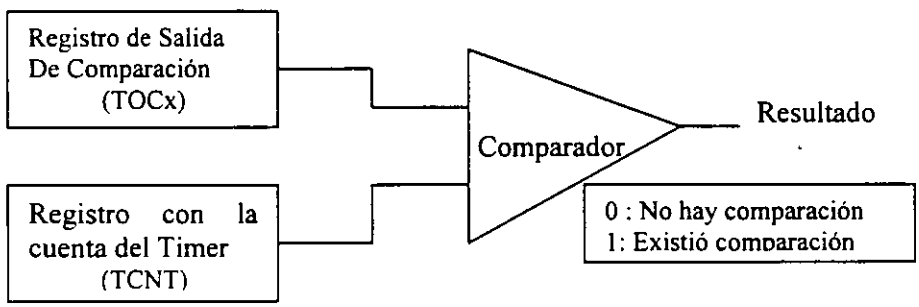
Se utilizan como indicadores de tiempo y para controlar la generación de ondas.

Durante cada ciclo de reloj, las salidas de los registros de comparación son comparados con el contador que corre libremente, si el valor de los registros es igual al del contador, entonces:

1.- El bit de bandera de interrupciones es prendido, además puede ocurrir lo siguiente:

a) El estado del pin de salida asociado puede cambiar de estado
b) Una interrupción puede ocurrir

Función de la salida de comparación



Registros asociados

Registros de salida de comparación	TOC1	$1016, $1017
	TOC2	$1018, $1019
	TOC3	$101A, $101B
	TOC4	$101C, $101d

Registro donde se almacena el valor con el cuál se compara el contador, para que al ser iguales, se genere la acción seleccionada.

TMSK1 Bandera de Interrupción del Temporizador 1 $1022

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OC1F	OC2F	0C3F	OC4F	I4/OC5F			

OC1F – OC5F Se activa la bandera correspondiente cuando existió comparación

TMSK1 Máscara de Interrupciones del Temporizado $1022

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OC1I	OC2I	OC3I	OC4I	I4/OC5I			

OC1I – OC4I Habilitación de interrupciones cuando exista comparación el pin seleccionado

TCTL1 Control del Temporizador $1020

OMx	OLx	Configuración
0	0	Sin acción
0	1	Cambia la salida de OCx
1	0	OCx se va a cero
1	1	OCx se va a uno

Función de Captura de las Entradas

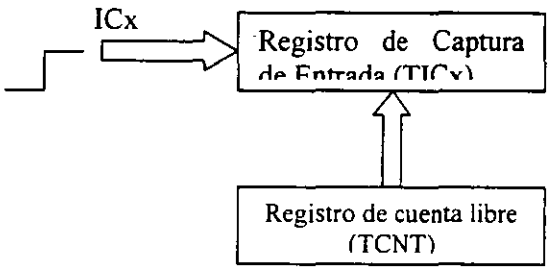
Se captura el valor del timer de 16 bits cuando una acción ocurre en el pin de entrada IC1 – Ic3.

Se utiliza para:

1.- Medir tiempos entre intervalos (ocurridos en hardware)
2.- Relacionar a eventos en tiempo real

Cuando una transición ocurre en el pin seleccionado, entonces se pueden dar los siguientes casos:

a) El valor del contador libre va al registro de captura de entrada, entonces una bandera de estado es prendida.
b) Una interrupción puede ocurrir.



Registros asociados:

Registros de Captura de entrada del Timer	TIC1	$1010, $1011
	TIC2	$1012, $1013
	TIC3	$1014, $1015

TFLG1 Registro de Bandera de Interrupción del Timer $1023

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					IC1F	IC2F	IC3F

IC1F – IC3F Bandera del registro de la entrada de captura seleccionada

TMSK1 Registro de Mascara de Interrupción del Timer $1022

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					IC1I	IC2I	IC3I

IC1I – IC3I Habilitación de las interrupciones para la entrada de captura seleccionada

TCTL2 Registro de Control del Timer 2 $1021

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A

Registro de control de la captura de entrada, según la siguiente configuración:

EDGxB	EDGxA	Configuración
0	0	Captura deshabilitada
0	1	Captura en flanco de subida
1	0	Captura en flanco de bajada
1	1	Captura en cualquier flanco

Acumulador de pulsos

Es útil como sistema contador de eventos y para acumulación de tiempos.

Responde a transiciones en el pin PAI, incrementando el contador/acumulador de pulsos de 8 bits; en el modo gated el contador de 8 bits es incrementado por E/64, después de ocurrir una transición en PAI.

Si una transición ocurre, entonces:

1.- La bandera de acumulación de pulsos es prendida
2.- El contador puede ser incrementado
3.- El bit de sobreflujo del acumulador de pulsos puede ser prendida
4.- Una interrupción puede ocurrir.

Registros asociados:

PACNT Cuenta del Acumulador de Pulsos $1027

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Contiene el valor del acumulador de pulsos.

TFLG2 Registro de bandera de interrupción del Timer 2$1025

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		PAOVF	PAIF				

PAOVF Bandea de sobreflujo del acumulador de pulsos; se activa cuando en el registro PACNT cambia la cuenta de $FF a $00

PAIF Bandera de Interrupción del acumulador de pulsos

Registro de Mascara de Interrupciones del Timer TMSK2 $1024

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		PAOVI	PAII				

PAOVI Habilita interrupciones de sobreflujo del acumulador de pulsos
0 = Deshabilitado
1 = Se requiere cuando PAIF del registro TFLG2 es activado

PAII Habilita interrupciones del acumulador de pulsos

0 = Interrupciones del acumulador de pulsos deshabilitado
1 = Interrupción requerida cuando el bit PSIF de TFLG" es activado.

PACTL Control del Acumulador de Pulsos $1026

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DDRA7	PAEN	PAMOD	PEDGE				

DDRA7 Dirección de datos del pin PA7
0 = Entrada
1 = Salida

PAEN Habilitación del acumulador de pulsos
0 = Acumulador de pulsos deshabilitado
1 = Acumulador de pulsos habilitado

PAMOD Modo de operación del acumulador de pulsos
0 = Operación de cuenta de pulsos
1 = Acumulación de tiempos

PEDGE Control del flanco del acumulador de pulsos
0 = Incrementa el contador con flanco de bajada
1 = Incrementa el contador con flanco de subida

Interrupciones en Tiempo Real

Se utiliza para procesar a intervalos periódicos a intervalos especificados por el usuario, usando el contador de cuenta libre, pudiendo ocurrir lo siguiente:

1.- La bandera de interrupción de tiempo real es prendida, además una interrupción puede ocurrir.

Registros asociados:

TCNT Cuenta de temporizador $100E, $100f

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit2	Bit 1	Bit 0

TFLG2 Registro de bandera de interrupción del Timer 2$1025

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	RTIF						

RTIF Bandera de Interrupción de tiempo real

Registro de Mascara de Interrupciones del Timer TMSK2 $1024

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	RTII						

RTII Habilitación de interrupciones en tiempo real

PACTL .. Control del Acumulador de Pulsos PACTL $1026

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
						RTR1	RTR0

RTR1:RTR0 Selección de la razón de las interrupciones

RTR1	RTR0	Divide E por
0	0	2^{13}
0	1	2^{14}
1	0	2^{15}
1	1	2^{16}

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS ABIERTOS

DISEÑO DE ROBOTS MÓVILES

TEMA :

INTRODUCCIÓN

EXPOSITOR: Dr. JESÚS SAVAGE CARMONA
PALACIO DE MINERÍA
MAYO DE 1999

1 Introducción

Una de las partes centrales de un robot móvil son los programas que lo controlan, ya que son estos los que conducen al hardware para que el robot tenga el comportamiento deseado por el usuario.

Hay varias formas de programar un robot, dependiendo de si se desea que interactúe de manera inteligente con su entorno o no. En el caso de los robots móviles es prácticamente imposible lograr que el robot tenga un desempeño adecuado si no se programa utilizando técnicas de inteligencia artificial (IA).

Existen diversas arquitecturas de control de robots móviles, en los primeros años de su desarrollo se utilizó una arquitectura que se ha dado en llamar "tradicional". En la arquitectura tradicional, como se puede ver en la figura 1 existen varios componentes que tienen las siguientes tareas:

1. Obtener datos de los sensores.
2. Interpretar esos datos.
3. Realizar un modelo del mundo en función de la respuesta de los sensores.
4. Planificar los movimientos necesarios en función de la meta y del modelo del mundo.
5. Enviar instrucciones a los actuadores.

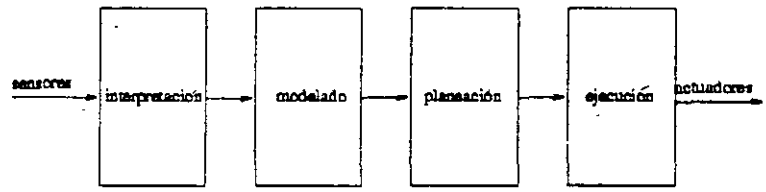


Figura 1: Enfoque tradicional

Este enfoque fue aplicado con relativo éxito en muchos casos, pero desgraciadamente en el momento en que los sistemas de prueba se han sacado de los ambientes fabricados expresamente para su operación, han fallado. Ahora es claro que es muy difícil obtener suficientes datos de los sensores que permitan realizar un modelo exacto del mundo. Además el ciclo sensado, modelado, planificación, actuación es muy lento y puede hacer que un

robot tarde varios minutos en tomar una decisión simple si no cuenta con suficiente potencia de cómputo (y en ocasiones aún teniéndola).

Por otro lado tenemos la arquitectura de control basada en comportamientos desarrollado por Rodney Brooks en el MIT. Esta arquitectura está compuesta por una red de máquinas de estado finito aumentadas (AFSM's por sus siglas en inglés) generadoras de comportamientos, cada una de las AFSM's envía y recibe mensajes y almacena información respecto a los mensajes que se recibieron más recientemente. Dependiendo de los datos que cada AFSM recibe, puede enviar mensajes de salida o mensajes a otros AFSM's. Los mensajes de salida generados por las diversas AFSM's son recibidos por otro módulo diseñado para manejar los conflictos que se pueden producir. Por ejemplo un comportamiento puede ordenar un giro a la derecha, mientras la orden de otro comportamiento puede ser girar hacia la izquierda. La manera en que el conflicto se resuelve es dando prioridad a las órdenes dadas por el comportamiento más importante "sometiendo" a las demás momentaneamente. En la figura 2 se puede ver la distribución de los módulos generadores de comportamientos.

En el enfoque de comportamientos, se aprovecha la capacidad de generación de comportamientos de cada módulo, de manera que cuando se cuenta con varios de éstos, surgen comportamientos complejos a partir de otros que son muy simples, de esta forma se puede generar un comportamiento aparentemente inteligente.

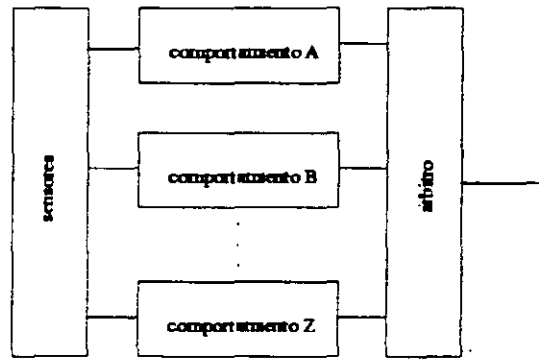


Figura 2: Arquitectura de comportamientos

2 Arquitectura de control de nuestro robot móvil

Una véz revisadas algunas propuestas de arquitecturas para controlar robots, nos podemos avocar a definir una arquitectura de control para el robot que se está desarrollando durante el curso.

Se propone una arquitectura que aproveche las ventajas de la arquitectura tradicional y de la de comportamientos. Como se aprecia en la figura 3 los componentes principales son: Un planificador, un navegador, un piloto y un controlador.

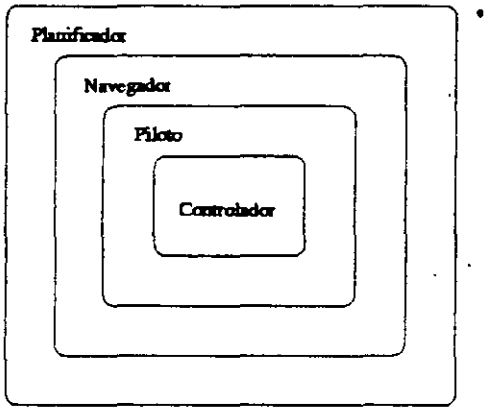


Figura 3: Organización lógica de un robot móvil

El *planificador* genera una ruta que permite que el robot llegue a su meta sin chocar con los obstáculos predefinidos en un mapa. Debe generar rutas globales en un entorno compuesto por áreas grandes y rutas locales.

El *navegador* divide la ruta en una secuencia de configuraciones que le envía al piloto.

El *piloto* recibe del navegador la siguiente configuración y trata de llegar a ella sin chocar. Ésto se logró construyendo al piloto de manera que tiene dos comportamientos: uno que trata de posicionar al robot en la configuración deseada, y otro que evita colisiones con los obstáculos.

El *controlador* controla directamente a los motores del robot y toma lecturas de los sensores.

En este curso nos enfocamos en el controlador y al piloto, que son los componentes más cercanos al robot.

3 Piloto

Hay varias formas de conducir a un robot a sus metas, y dado que se usa una arquitectura con comportamientos, cada una de estas formas se puede reflejar en un comportamiento que presenta el robot. En esta sección tratamos dos mecanismos muy diferentes que le pueden aportar al robot un comportamiento también distinto. Por un lado tenemos dos algoritmos que le permiten al robot evadir obstáculos: los algoritmos Bug1 y Bug2. Por otra parte se aborda un mecanismo que le permite al robot seguir una línea blanca sobre un piso negro y que está basado en una técnica de control conocida como control difuso.

3.1 Algoritmos Bug1 y Bug2

La forma mas sencilla de sensado en un robot móvil es mediante sensores de tacto. El robot hace el recorrido de su posición inicial S a su posición objetivo T a través de una linea recta hasta que llega a T o interactúa con el *i-esimo* obstáculo en el punto de contacto H_i. Si se dió el contacto el robot sigue el perímetro del obstáculo hasta que llega al punto de salida L_i y continua en línea recta hacia T, como se puede ver en la figura 4. Dado que no se conocen las características del objeto con el que se hizo contacto, la dirección en que se le rodea se establece de manera arbitraria, por ejemplo a la izquierda. El problema de seguir este algoritmo es que es probable que por la forma de los obstáculos o por la distribución de éstos en el entorno del robot, nunca se llegue al objetivo a pesar de haber un camino.

Para resolver este problema se han propuesto los algoritmos Bug1 y Bug2 [?] que asumen que el robot puede conocer en todo momento sus coordenadas y las de su objetivo.

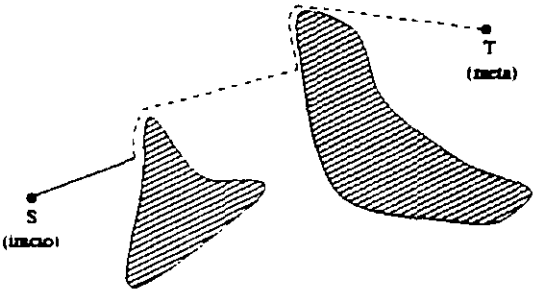


Figura 4: Navegación con sensores de tacto

3.1.1 El algoritmo Bug1

El algoritmo Bug1 consta de los siguientes pasos:

1. Del punto L_{i-1} (con $L_0 = S$, mueve al robot hacia T a lo largo de la recta (S,T) hasta que ocurra alguna de las siguientes opciones:

   (a) T es alcanzado; el proceso termina.

   (b) Se encuentra un obstáculo en el punto de contacto H_i; ve al paso 2.

2. Usando la dirección de movimiento predefinida, sigue la orilla del obstáculo. Si se llega a T, para. Mientras se avanza, se almacenan las coordenadas del punto actual en el registro R_1, el punto visitado mas cercano a T se almacena en Q_m, la distancia recorrida desde H_j en el registro R_2 y la distancia recorrida desde Q_m se guardan en el registro R_3. Después de haber rodeado completamente al obstáculo (se llegó de nuevo a H_i) se define el punto de salida $L_i = Q_m$ y se continúa con el paso 3.

3. Si hay un segmento de recta entre L_i y T que cruze el obstáculo en el punto L_i, termina el proceso, porque el objetivo no es alcanzable. En otro caso utiliza R_2 y R_3 para determinar el camino más corto a L_i, se sigue el camino elegido y se establece $i = i + 1$ y se regresa a 1.

En la figura 5 se observa el ejemplo de un robot rodeando los obstáculos con el algoritmo Bug1.

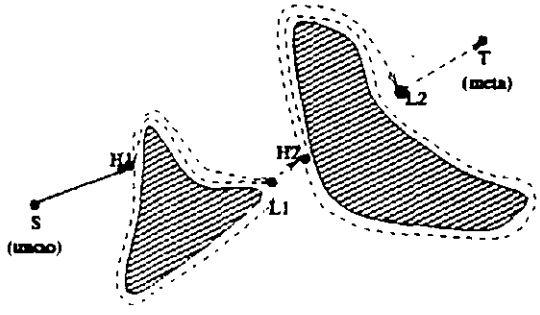


Figura 5: Ejemplo del algoritmo Bug1

3.1.2 El algoritmo Bug2

El algoritmo Bug2 es una variante de Bug1. En la figura 6 se observa un ejemplo de aplicar los pasos que se detallan a continuación:

1. Del punto L_{j-1} (con $L_0 = S$), mueve al robot hacia T a lo largo de la recta (S,T) hasta que ocurra una de las siguientes opciones:
   (a) T es alcanzado; termina el proceso.
   (b) Se encuentra un obstáculo en el punto de contacto H_j; se sigue en el paso 2.
2. Usando la dirección de movimiento predefinida, se sigue el contorno del obstáculo hasta que:
   (a) T es alcanzado; termina el proceso.
   (b) La recta (S,T) se encuentra en un punto Q tal que la distancia $D(Q,T) < D(H_j,T)$ y la recta (Q,T) no cruza al obstáculo en el punto Q. Se define el punto de salida $L_j = Q_m$, se establece $j = j+1$ y se va al paso 1.
   (c) El robot recorre totalmente el contorno del obstáculo y regresa a H_j sin haber definido el siguiente punto de contacto H_{j+1}, en este caso termina el proceso, ya que el objetivo está "atrapado" y es inalcanzable.

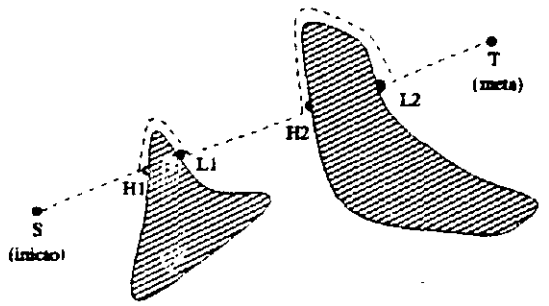


Figura 6: Ejemplo del algoritmo Bug2

En la práctica la eficiencia de estos dos algoritmos es variable, pero se puede decir que Bug1 es más conservador, ya que garantiza que el punto de salida es el más cercano a la meta. Bug2 es más agresivo y muchas veces más eficiente, ya que no requiere rodear todo el obstáculo.

3.2 Seguir la línea usando lógica difusa

3.2.1 lógica difusa

El control difuso es una técnica de control fundamentada en la lógica difusa o lógica de lo borroso propuesta por Lofti Zadeh. La lógica difusa esun superconjunto de la lógica booleana en la que se redefinen las operaciones lógicas tradicionales AND, OR y NOT de forma que admitan valores en el rango de [0,1] y no solamente los valores 0 y 1 o Verdadero y Falso.

3.2.2 Definición de las características funcionales del controlador

El comportamiento que estamos tratando consiste en que el coche siga una línea blanca sobre un piso negro. Para esto utilizamos los sensores fotoreflectivos que tiene el robot en su parte inferior. La salida de cada uno de estos se introduce en el puerto E del robot en el que se encuentra un convertidor analógico-digital. Solamente se utilizan los sensores de los extremos de manera que el robot debe asegurarse de que la intensidad de luz reflejada por estos sea baja, o lo que es lo mismo que la línea blanca se encuentre entre los dos.

3.2.3 Definición de las superficies de control

En el momento en que uno de los sensores empieze a salir del camino se deberá tomar acción correctiva. Para poder hacer esto con lógica difusa es necesario leer cada sensor y fusificar las entradas (definir el grado de membresía con los conjuntos). Se decidió dividir los conjuntos difusos de cada entrada en MUY OSCURO, OSCURO, INTERMEDIO, CLARO Y MUY CLARO, y para definir los tamaños de cada conjuntos difuso correspondiente a cada sensor fué necesario hacer pruebas de las cuales fué posible obtener los siguientes conjuntos:

SENSOR DERECHO:

SENSOR IZQUIERDO:

Las variables de salida difusas son tiempo de movimiento y dirección del mismo. A la variable de tiempo se le pueden asignar los valores de MUY POCO, POCO, MEDIO, LARGO y MUY LARGO, mientras a la variable de dirección se le asignan los valores de ADELANTE, DERECHA y IZQUIERDA. Si bien en nuestro caso por tener un control muy limitado de dirección, esta variable no cambiará tan suavemente de un valor a otro como la variable de tiempo. Los conjuntos difusos de salidas quedan como sigue:

TIEMPO

DIRECCIÓN

3.2.4 Definición de reglas

Ahora continuamos con la definición de las reglas:

Si SENSOR DERECHO es MUY OSCURO y SENSOR IZQUIERDO es MUY OSCURO

entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY LARGO.

Si SENSOR DERECHO es MUY OSCURO y SENSOR IZQUIERDO es OSCURO

entonces DIRECCIÖN es ADELANTE y TIEMPO es LARGO

Si SENSOR DERECHO es MUY OSCURO y SENSOR IZQUIERDO es INTERMEDIO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MEDIO
Si SENSOR DERECHO es MUY OSCURO y SENSOR IZQUIERDO es CLARO
entonces DIRECCIÖN es IZQUIERDA y TIEMPO es MEDIO
Si SENSOR DERECHO es MUY OSCURO y SENSOR IZQUIERDO es MUY CLARO
entonces DIRECCIÖN es IZQUIERDA y TIEMPO es MUY LARGO
Si SENSOR DERECHO es OSCURO y SENSOR IZQUIERDO es MUY OSCURO
entonces DIRECCIÖN es ADELANTE y TIEMPO es LARGO.
Si SENSOR DERECHO es OSCURO y SENSOR IZQUIERDO es OSCURO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MEDIO
Si SENSOR DERECHO es OSCURO y SENSOR IZQUIERDO es INTERMEDIO
entonces DIRECCIÖN es ADELANTE y TIEMPO es POCO
Si SENSOR DERECHO es OSCURO y SENSOR IZQUIERDO es CLARO
entonces DIRECCIÖN es IZQUIERDA y TIEMPO es MEDIO
Si SENSOR DERECHO es OSCURO y SENSOR IZQUIERDO es MUY CLARO
entonces DIRECCIÖN es IZQUIERDA y TIEMPO es MUY LARGO
Si SENSOR DERECHO es INTERMEDIO y SENSOR IZQUIERDO es MUY OSCURO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MEDIO.
Si SENSOR DERECHO es INTERMEDIO y SENSOR IZQUIERDO es OSCURO
entonces DIRECCIÖN es ADELANTE y TIEMPO es POCO
Si SENSOR DERECHO es INTERMEDIO y SENSOR IZQUIERDO es INTERMEDIO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es INTERMEDIO y SENSOR IZQUIERDO es CLARO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es INTERMEDIO y SENSOR IZQUIERDO es MUY CLARO
entonces DIRECCIÖN es IZQUIERDA y TIEMPO es MUY POCO

Si SENSOR DERECHO es CLARO y SENSOR IZQUIERDO es MUY OSCURO
entonces DIRECCIÖN es DERECHA y TIEMPO es MEDIO.
Si SENSOR DERECHO es CLARO y SENSOR IZQUIERDO es OSCURO
entonces DIRECCIÖN es DERECHA y TIEMPO es MEDIO
Si SENSOR DERECHO es CLARO y SENSOR IZQUIERDO es INTERMEDIO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es CLARO y SENSOR IZQUIERDO es CLARO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es CLARO y SENSOR IZQUIERDO es MUY CLARO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es MUY CLARO y SENSOR IZQUIERDO es MUY OSCURO
entonces DIRECCIÖN es DERECHA y TIEMPO es MUY LARGO.
Si SENSOR DERECHO es MUY CLARO y SENSOR IZQUIERDO es OSCURO
entonces DIRECCIÖN es DERECHA y TIEMPO es MUY LARGO
Si SENSOR DERECHO es MUY CLARO y SENSOR IZQUIERDO es INTERMEDIO
entonces DIRECCIÖN es DERECHA y TIEMPO es MUY POCO
Si SENSOR DERECHO es MUY CLARO y SENSOR IZQUIERDO es CLARO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO
Si SENSOR DERECHO es MUY CLARO y SENSORIZQUIERDO es MUY CLARO
entonces DIRECCIÖN es ADELANTE y TIEMPO es MUY POCO

3.2.5 Métodos de defusificación

Ahora queda la selección del método de defusificación. Como tenemos dos salidas de naturaleza distinta, tendremos dos métodos de defusificación: Por centroide para el TIEMPO y por conjunto de valor máximo para la DIRECCIÓN.

3.2.6 Programa para el MC68HC11

```
#define vel_rectas 0x09ff
```

```
#define vel_curvas 0x09ff

#define veces 500
#define punto_muy_claro_izq      0
#define pendiente_muy_claro_izq  100/100
#define punto_claro_izq       100
#define pendiente_claro_izq   100/30
#define punto_intermedio_izq       130
#define pendiente_intermedio_izq   100/10
#define punto_oscuro_izq       160
#define pendiente_oscuro_izq   100/30
#define punto_muy_oscuro_izq      255
#define pendiente_muy_oscuro_izq  100/95
#define punto_muy_claro_der      0
#define pendiente_muy_claro_der  100/100
#define punto_claro_der       100
#define pendiente_claro_der   100/30
#define punto_intermedio_der       130
#define pendiente_intermedio_der   100/10
#define punto_oscuro_der       160
#define pendiente_oscuro_der   100/30
#define punto_muy_oscuro_der      255
#define pendiente_muy_oscuro_der  100/95
#define punto_muy_poco      0
#define pendiente_muy_poco  100/2
#define punto_poco       2
#define pendiente_poco   100/2
#define punto_medio       5
#define pendiente_medio   100/2
#define punto_largo       7
#define pendiente_largo   100/2
#define punto_muy_largo      10
#define pendiente_muy_largo  100/2
#define t_maximo      10

main(void)
{
   int sens_izq,sens_der,mayor;
   long tiempo;
   int izq_muy_oscuro, izq_oscuro, izq_intermedio, izq_claro, izq_muy_clar
```

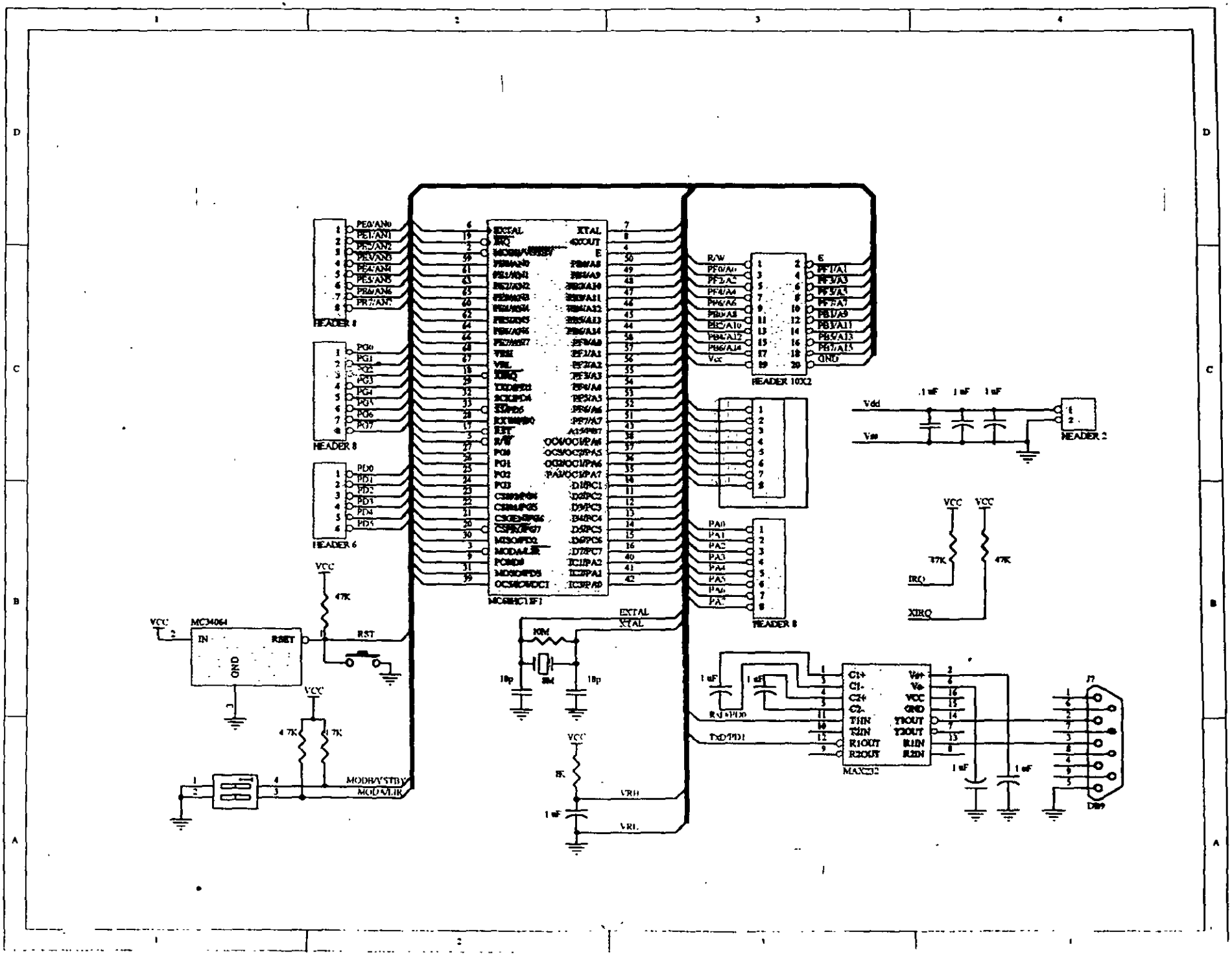
```
   int der_muy_oscuro, der_oscuro, der_intermedio, der_claro, der_muy_claro;
   int adelante, derecha, izquierda, muy_largo, largo, medio, poco, muy_poco;
   char direccion;
   inicializa();
   init_communications();
   do
   {  lee_ad(&sens_izq,&sens_der);


      /* fusificaci'on de las entradas*/
      izq_muy_oscuro =
        pertenece (punto_muy_oscuro_izq,pendiente_muy_oscuro_izq,sens_izq);
      izq_oscuro =
        pertenece (punto_oscuro_izq,pendiente_oscuro_izq,sens_izq);
      izq_intermedio =
        pertenece (punto_intermedio_izq,pendiente_intermedio_izq,sens_izq);
      izq_claro =
pertenece (punto_claro_izq,pendiente_claro_izq,sens_izq);
      izq_muy_claro =
pertenece (punto_muy_claro_izq,pendiente_muy_claro_izq,sens_izq);
      der_muy_oscuro =
pertenece (punto_muy_oscuro_der,pendiente_muy_oscuro_der,sens_der);
      der_oscuro =
pertenece (punto_oscuro_der,pendiente_oscuro_der,sens_der);
      der_intermedio =
pertenece (punto_intermedio_der,pendiente_intermedio_der,sens_der);
      der_claro =
pertenece (punto_claro_der,pendiente_claro_der,sens_der);
      der_muy_claro =
pertenece (punto_muy_claro_der,pendiente_muy_claro_der,sens_der);
      /*c'alculo de conjuntos de salida */
      adelante = 0;
      derecha = 0;
      izquierda = 0;
      muy_largo = 0;
      largo = 0;
      medio = 0;
      poco = 0;
      muy_poco = 0;
```

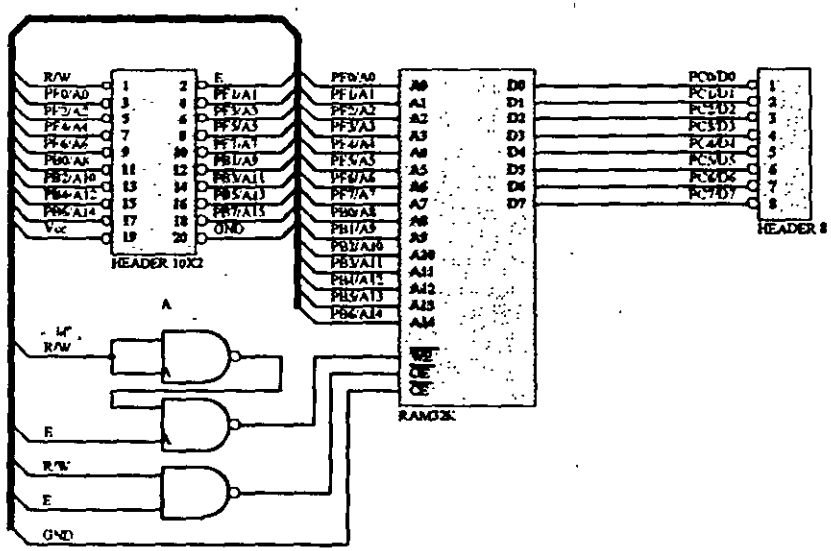
```
regla (izq_muy_oscuro,der_muy_oscuro, &adelante,&muy_largo);
regla (izq_oscuro,der_muy_oscuro, &adelante,&largo);
regla (izq_intermedio,der_muy_oscuro, &adelante,&medio);
regla (izq_claro,der_muy_oscuro, &izquierda,&medio);
regla (izq_muy_claro,der_muy_oscuro, &izquierda,&muy_largo);

regla (izq_muy_oscuro,der_oscuro, &adelante,&largo);
regla (izq_oscuro,der_oscuro, &adelante,&medio);
regla (izq_intermedio,der_oscuro, &adelante,&poco);
regla (izq_claro,der_oscuro, &izquierda,&medio);
regla (izq_muy_claro,der_oscuro, &izquierda,&muy_largo);

regla (izq_muy_oscuro,der_intermedio, &adelante,&medio);
regla (izq_oscuro,der_intermedio, &adelante,&poco);
regla (izq_intermedio,der_intermedio, &adelante,&muy_poco);
regla (izq_claro,der_intermedio, &adelante,&muy_poco);
regla (izq_muy_claro,der_intermedio, &izquierda,&muy_poco);

regla (izq_muy_oscuro,der_claro, &derecha,&medio);
regla (izq_oscuro,der_claro, &derecha,&medio);
regla (izq_intermedio,der_claro, &adelante,&muy_poco);
regla (izq_claro,der_claro, &adelante,&muy_poco);
regla (izq_muy_claro,der_claro, &adelante,&muy_poco);

regla (izq_muy_oscuro,der_muy_claro, &derecha,&muy_largo);
regla (izq_oscuro,der_muy_claro, &derecha,&muy_largo);
regla (izq_intermedio,der_muy_claro, &derecha,&muy_poco);
regla (izq_claro,der_muy_claro, &adelante,&muy_poco);
regla (izq_muy_claro,der_muy_claro, &adelante,&muy_poco);
```

```
/* defusificacin de la direccin */
mayor = adelante;
direccion = 'a';
if (derecha > mayor)
```

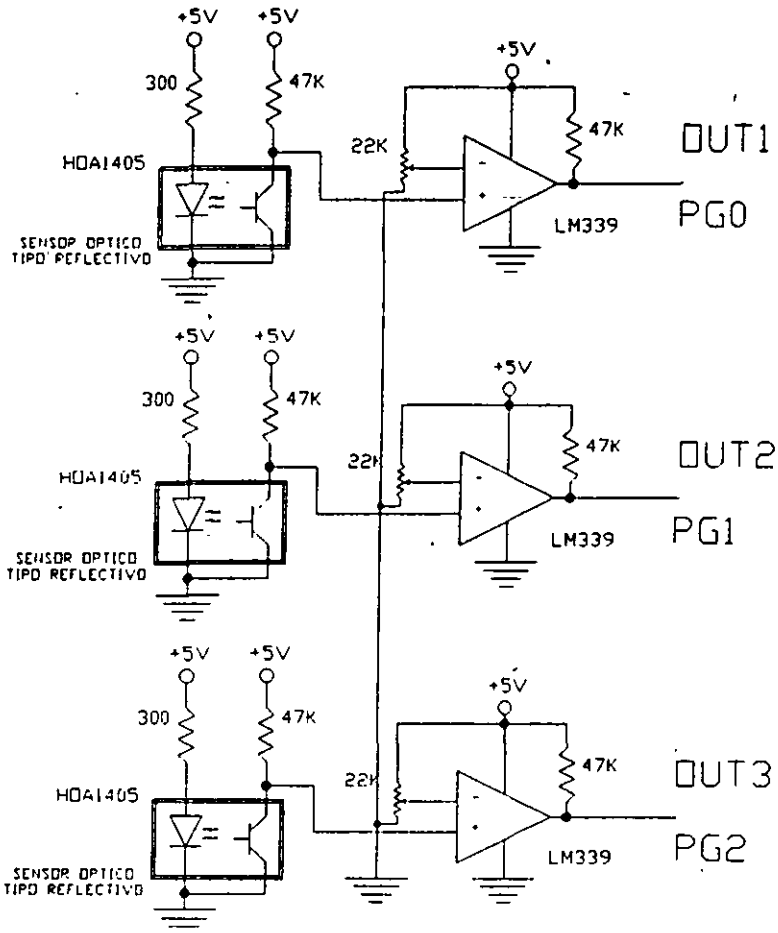
```
      {  mayor = derecha;
 direccion = 'd';
      }
      if (izquierda > mayor)
      {  direccion = 'i';
      }
      /* defusificacin del tiempo */
      tiempo = muy_poco*punto_muy_poco + poco*punto_poco + medio*punto_medio +
largo*punto_largo + muy_largo*punto_muy_largo;
      tiempo = tiempo / (muy_poco + poco + medio + largo + muy_largo);
      mueve(direccion,tiempo);
   }while (1);
}
inicializa()
{          poke(0x1020, 0x28);/*timers 3 y 4 se limpian en comparaci'on exitosa
poke(0x100C, 0x30);/*oc1 afectar'a al 3 y al 4 */
poke(0x100D, 0);/* timers 3 y 4 adquirir'an el valor de cero en cuenta exitosa*/
pokeword(0x101C, vel_curvas);
pokeword(0x101A, vel_rectas);
poke(0x1039, 0x90);/* enciende el A/D con retardo*/
bit_clr(0x1028, 0x20);/* desactiva el open drain del puerto d*/
poke(0x1009, 0x0C);/* activa como salidas D2 y D3*/
}
lee_ad(int *izq,int *der)
{       int status;
poke(0x1030, 0x10);
while (((status=peek(0x1030))& 0x80)==0);
*izq=peek(0x1032);
*der=peek(0x1033);
}
mueve(dir, tiem)
char dir;long tiem;
{  long i,j;
   if (dir=='a')
   {  poke(0x100D, 0x20);
      poke(0x1008, 0x04);
      putchar('a'); }
   else if (dir=='d')
   {  poke(0x100D, 0x10);
      poke(0X1008, 0x08);
```

```
      putchar('d'); }
   else if (dir=='i')
   {  poke(0x100D, 0x10);
      poke(0x1008, 0x00);
      putchar('i');}
   putchar('-');
   puthex(tiem);
   putchar('*');
   putchar('*');
   for (j=0;j<veces;j++)
     for (i=tiem;i>=0; i--);
}

int pertenece(int punto, int pendiente, int sensor)
{  int tmp, derecho, izquierdo;
   derecho = -pendiente*(sensor-punto) + 100;
   izquierdo = pendiente*(sensor-punto) + 100;
   if ((izquierdo > 0) && (izquierdo <= 100))
      tmp = izquierdo;
   else
   {  if ((derecho > 0) && (derecho <=100))
 tmp = derecho;
      else
 tmp = 0;
   }
   return (tmp);
}

regla(int premisa1, int premisa2, int *accion1, int *accion2)
{  int and;
   and = premisa2;
   if (premisa1 <= premisa2)
      and = premisa1;
   if (and >= *accion1)
      *accion1 = and;
   if (and >= *accion2)
      *accion2 = and;
}
```

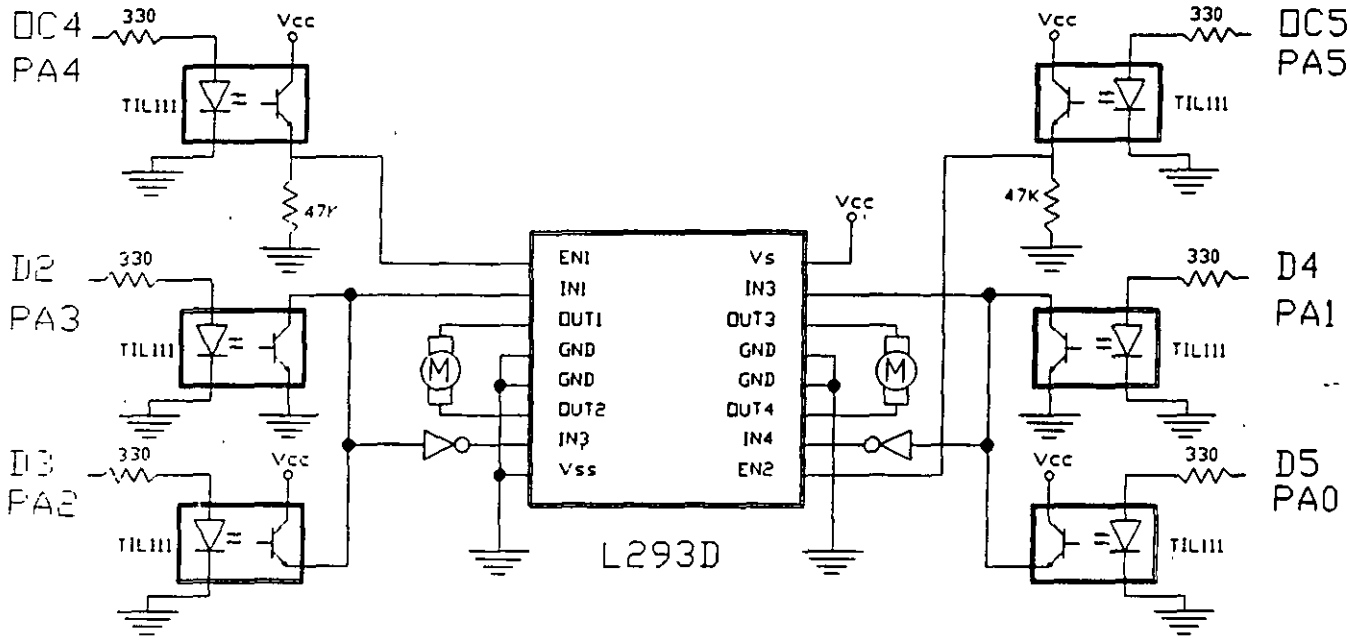

HEADER 8
HEADER 8
HEADER 6
HEADER 10X2
HEADER 8
HEADER 2
MC34064
IN
RSET
GND
RST
VCC
47K
4.7K
1K
MODB/VSTBY
MODA/LIR
XTAL
EXTAL
4M
18p
VRH
VRL
1uF
MAX232
C1+
C1-
C2+
C2-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
Vdd
Vss
.1uF
IRQ
XIRQ
RxD/PD0
TxD/PD1
DB9

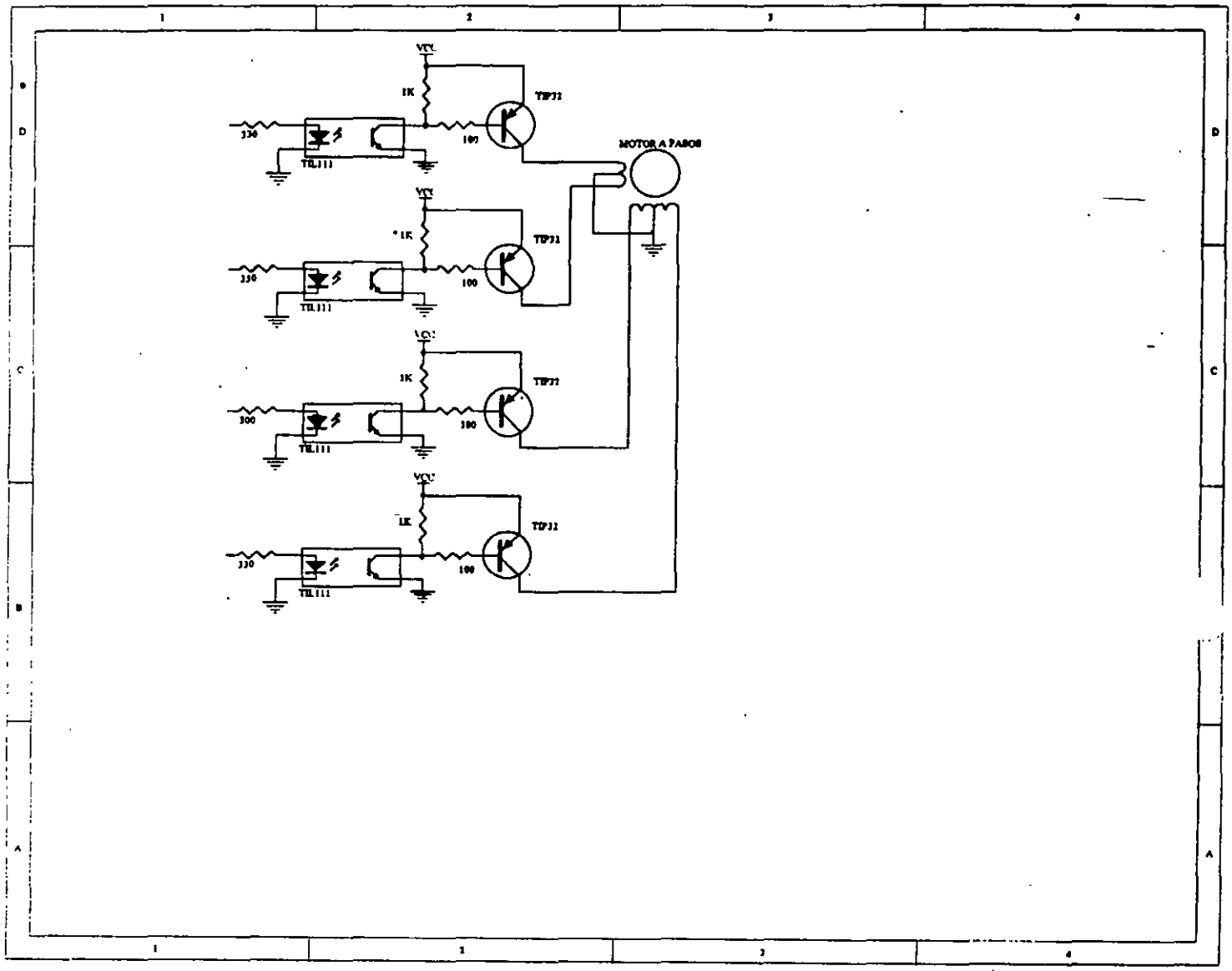

R/W
E
GND
HEADER 10X2
HEADER 8
RAM32K
A

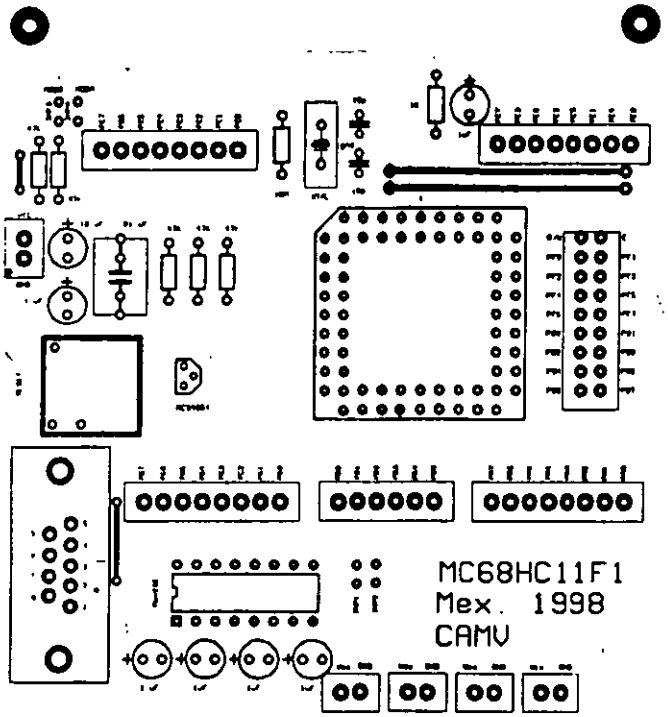
ETAPA DE SENSADO DE NIVEL DE INTENSIDAD DE LUZ

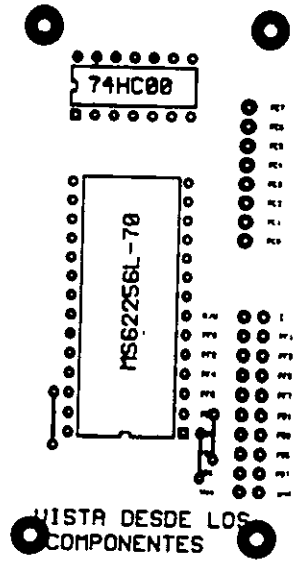


CONTROL DE MOTORES DE CORRIENTE DIRECTA (1 Amp. max.)



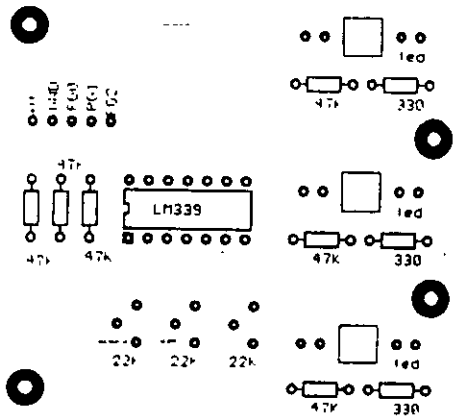

VCC
1K
TIP32
330
100
TIL111
MOTOR A PASOS
VCC
1K
TIP32
330
100
TIL111
VCC
1K
TIP32
330
100
TIL111
VCC
1K
TIP32
330
100
TIL111

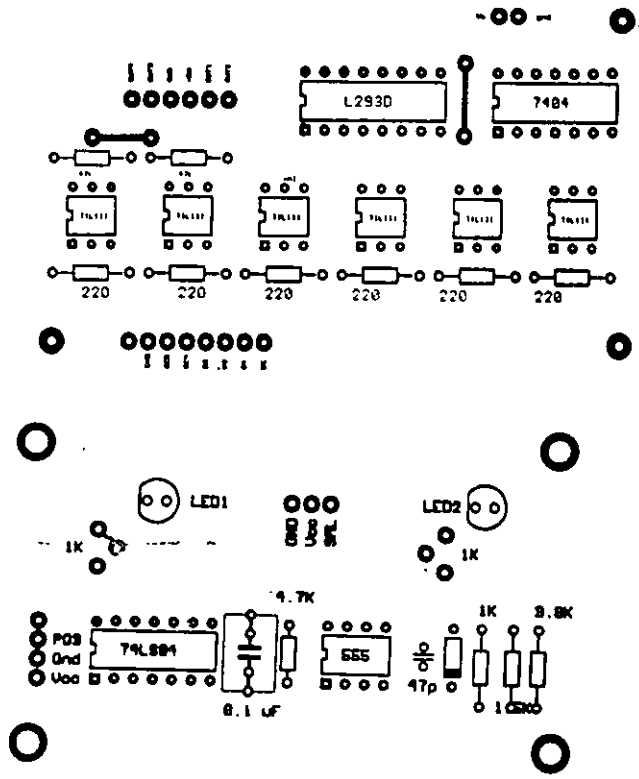


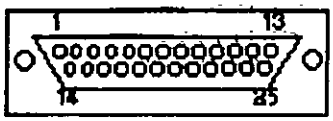


VISTA DESDE LOS COMPONENTES

VISTA DESDE LOS COMPONENTES







View is looking at
Connector side of
DB-25 Male Connector.

Pin	Description	
1	$\overline{\text{Strobe}}$	PC Output
2	Data 0	PC Output
3	Data 1	PC Output
4	Data 2	PC Output
5	Data 3	PC Output
6	Data 4	PC Output
7	Data 5	PC Output
8	Data 6	PC Output
9	Data 7	PC Output
10	$\overline{\text{ACK}}$	PC Input
11	Busy	PC Input
12	Paper Empty	PC Input
13	Select	PC Input
14	$\overline{\text{Auto Feed}}$	PC Output
15	$\overline{\text{Error}}$	PC Input
16	Initialize Printer	PC Output
17	$\overline{\text{Select Input}}$	PC Output

Pin Assignments

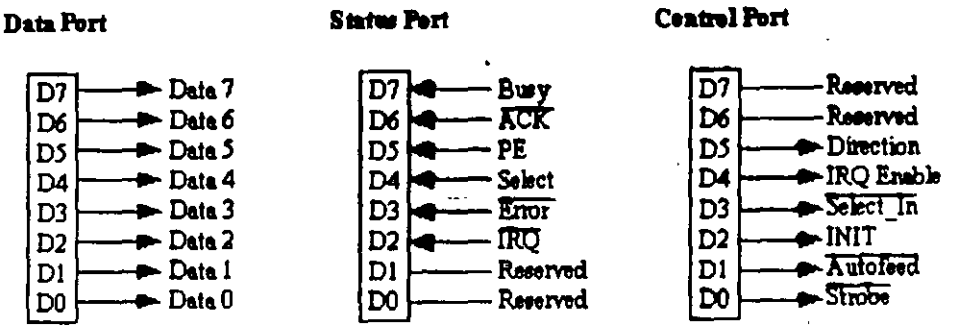
Note: 8 Data Outputs
4 Misc Other Outputs

5 Data Inputs

Note: Pins 18-25 are
Ground

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 / Paper-End	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		

Table 1. Pin Assignments of the D-Type 25 pin Parallel Port Connector.



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

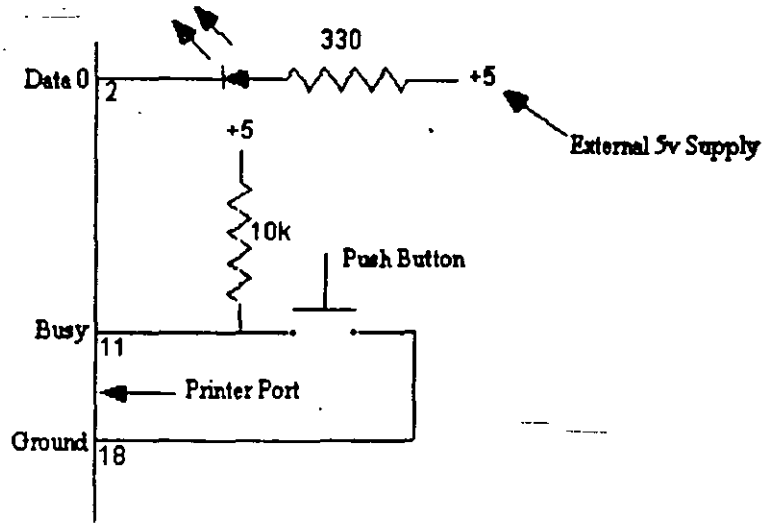
Table 4 Data Port

Offset	Name	Read/Write	Bit No.	Properties
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

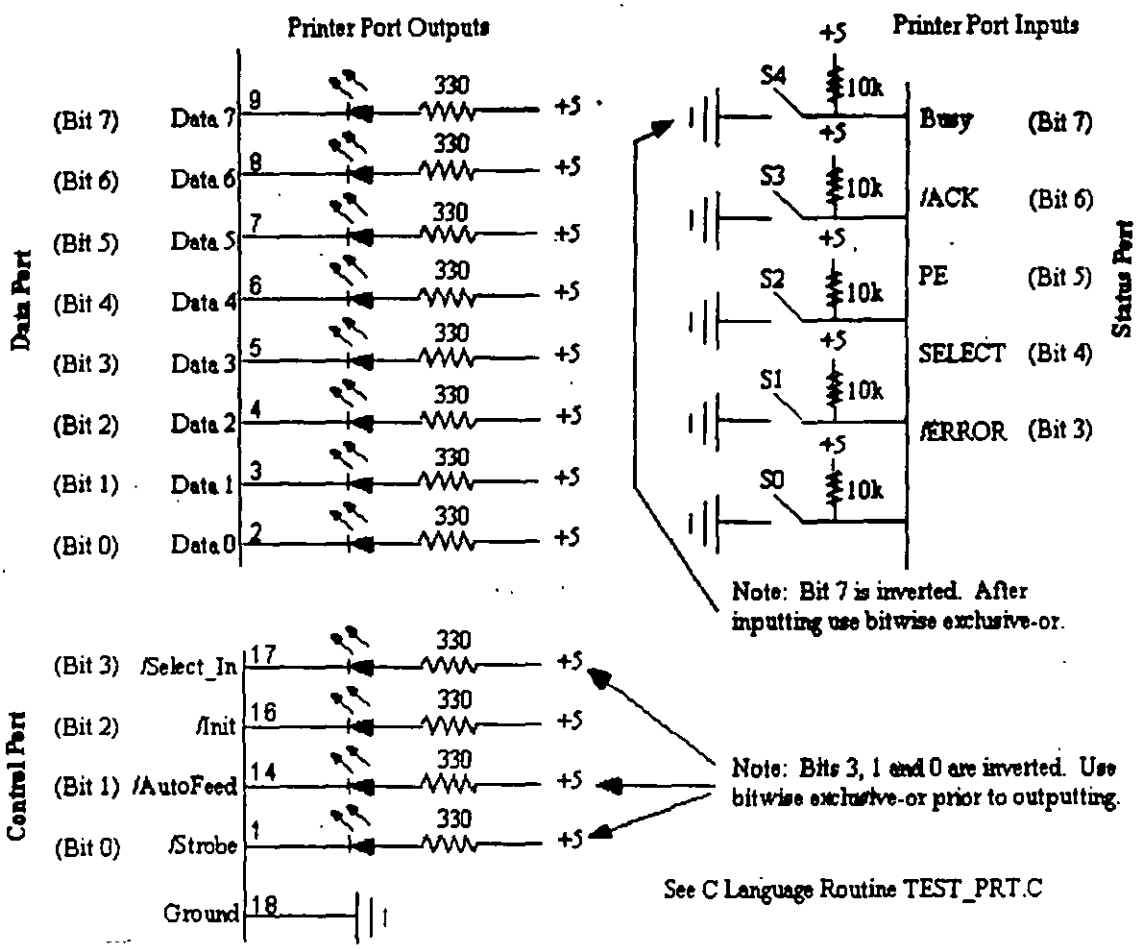
Table 5 Status Port

Offset	Name	Read/Write	Bit No.	Properties
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

Table 6 Control Port




```
/* Prende el led, si se presiona el interruptor */

#include <stdio.h>
#include <dos.h>

#define DATA 0x378
#define STATUS DATA+1
#define CONTROL DATA+2

void main(void)
{
   int in;
   while(1)
   {
      in = inportb(STATUS);
      if (((in^0x80)&0x80)==0)
      /* if BUSY bit is at 0 (sw closed) */
      {
         outportb(DATA,0x00);   /* Prende LED */
         delay(100);
         outportb(DATA, 0x01);   /* Apaga LED */
         delay(100);
      }
      else
      {
         outportb(DATA,0x01);
      /* if PB not depressed, turn LED off */
      }
   }
}
```




```
/* File TEST_PRT.C
**
** Program to exercise 12 outputs and five inputs.
**
** Program sequentially turns off LEDs on Bits 7, 6, 5, ... 0 on the
** Data Port, and then Bits #, 2, 1 and 0 on the Control Port:  Each
** LED is held off for nominally 1 second.  Note that an LED is turned
** off with a logic one.  This process is executed once.
**
** Program then loops, scanning the highest five bits on the Status Port
** and continuously displays the content in hexadecimal.
**
** P.H. Anderson, Dec 25, '95
*/

#include <stdio.h>
#include <dos.h>        /* required for delay function */

#define DATA 0x0378     /* for the PC I used */
#define STATUS DATA+1
#define CONTROL DATA+2
```

```
void main(void)
{
   int in, n;

   outportb(DATA,0x00); /* turn on all LEDs on Data Port */
   outportb(CONTROL, 0x00^0x0b); /* same with Control Port */

   /* now turn off each LED on Data Port in turn by positioning a logic
   one in each bit position and outputing.
   */
   for (n=7; n=0; n++)
   {
      outportb(DATA, 0x01 << n);
      delay(1000);
   }
   outportb(DATA, 0x00);

   /* now turnoff each LED on control port in turn
   ** note exclusive-or to compensate for hardware inversions
   */

   outportb(CONTROL, 0x08^0x0b);        /* bit 3 */
   delay(1000);
   outportb(CONTROL, 0x04^0x0b);        /* bit 2 */
   delay(1000);
   outportb(CONTROL, 0x02^0x0b);        /* bit 1 */
   delay(1000);
   outportb(CONTROL, 0x01^0x0b);        /* bit 0 */
   delay(1000);

   outportb(CONTROL, 0x00);

   /* Continuously scan switches and print result in hexadecimal */

   while(1)
   {
      in = (inportb(STATUS)^0x80)&0xf8;
      /* Note that BUSY (msbit) is inverted and only the
      ** five most significant bits on the Status Port are displayed.
      */
      printf("%x\n", in);
   }
}
```

```
***** CONFIGURACION DE CONSTANTES *****
EPROM    EQU     $100
STACK    EQU     $03FF
DELAY1   EQU     $00FFF  * CARGAR A REGISTRO X
DELAY2   EQU     $0001   * CARGAR A REGISTRO Y
OUT      EQU     $FF
IN       EQU     $00

LINEA    EQU     $07

***** CONFIGURACION DE REGISTROS *****
PORTA    EQU     $1000
DDRA     EQU     $1001
PORTG    EQU     $1002
DDRG     EQU     $1003

***** INICIO DEL PROGRAMA PRINCIPAL *****
         ORG EPROM      ; INICIO DE LA RAM INTERNA
         LDS #STACK
         LDAA #OUT
         STAA DDRA
         LDAA #$0F8
         STAA DDRG

REPITE   NOP
         LDAA PORTG
         ANDA #LINEA
         STAA PORTA
         JSR TIME

         JMP REPITE

***** SUBRUTINA DE TIEMPO *****
TIME     NOP
         LDX #DELAY1
RET2     NOP
         DEX
         BNE RET2
         RTS

         END EPROM
```

```
/*****************************************************
*                                                    *
*       REFLE.c                                      *
*                        8/Abril/99                  *
*                        C. Munive                   *
*****************************************************/


#include "stdio.h"

#define DDRA                    0x1001
#define PORTA                   0X1000
#define DDRG                    0x1003
#define PORTG                   0x1002

main(void)
{
   int SENSOR;

   inicializa();
   init_communications();
   puts("\n\n Rabochnick Version 0.40 ");

   while(1){
        SENSOR=peek(PORTG);
        poke(PORTA,SENSOR);
        puts_string_hex("Sensor -> ",SENSOR);
        wait(200);


   }     /*  Fin del while */
}       /* fin del main */


inicializa()
{       poke(DDRA, 0xFF);
        poke(DDRG, 0xF8);

}


dummy(){
int i;
i=1;
 asm("_END_PRG  equ *");
}
```

```
/* CONTROL DE UN MOTOR A PASOS
   ELABORADO POR: CARLOS A. MUNIVE VAZQUEZ
   STEP.C    =>   STEP.EXE
   ABRIL 1999
*/

#include <stdio.h>
#include <dos.h>

#define DATA 0x0378
#define STATUS DATA+1
#define CONTROL DATA+2
#define TIEMPO 100

void CW (int);
void CCW (int);

void main (void)
{
int pasos=10;
   CW(pasos);
   CCW(pasos);

}

void CW (int paso)
{
   int i;
   for (i=1;i<paso;i++)
   {
   outportb(DATA,0x0a);
   delay(TIEMPO);
   outportb(DATA,0x09);
   delay(TIEMPO);
   outportb(DATA,0x05);
   delay(TIEMPO);
   outportb(DATA,0x06);
   delay(TIEMPO);
   }
}

void CCW (int paso)
{
   int i;
   for (i=1;i<paso;i++)
   {
   outportb(DATA,0x06);
   delay(TIEMPO);
   outportb(DATA,0x05);
   delay(TIEMPO);
   outportb(DATA,0x09);
   delay(TIEMPO);
   outportb(DATA,0x0a);
   delay(TIEMPO);
   }
}
```