UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"MICROCONTROLADORES DE 8 Y 16 BITS PARA UN SISTEMA DE CONTROL VÍA INTERNET"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO Y ELECTRÓNICO

P R E S E N T A :

JÁUREGUI VILLANUEVA JOSÉ FERNANDO

DIRECTOR DE TESIS: ING. EDUARDO RAMÍREZ SÁNCHEZ

CIUDAD UNIVERSITARIA

MÉXICO, D.F., JUNIO 2003

A Fernando Jáuregui Zavala.
A Virginia Villanueva de Jáuregui,
A Raymundo Jáuregui.
A Virginia Jáuregui.
A la familia Martínez Orozco.

A Cecilia Martínez Orozco.

A mis compañeros y maestros.
A la Universidad Nacional Autónoma de México.

CONTENIDO.

INTRODUCCIÓN

Los ingenieros en sistemas embebidos se encuentran frente al dilema de implementar conectividad utilizando Internet en prácticamente todos sus diseños debido, entre otras cosas, al número creciente de usuarios, a la arquitectura abierta o bien, en un aspecto más técnico, su gran adaptación a una diversidad de medios de transmisión. La gran gama de aplicaciones que puede abarcar esta conectividad incluye el control remoto de procesos y es este punto el que nos ocupa en el presente trabajo ya que tiene relación directa con el control de dispositivos electrónicos y con alguna de sus tareas como pueden ser:

- Administración y mantenimiento remoto de equipos: detección de fallas, actualizaciones de software, etc..
- Sistemas de recolección de datos: comportamiento estadístico, tarificación, etc..
- Monitoreo remoto: detección de alarmas, analizadores de estado del equipo, etc..
- Medición remota de variables y señales: video, audio, clima, etc..

El objetivo de este trabajo es lograr controlar hardware a distancia utilizando un microprocesador de 8 bits. El hardware propuesto es una aplicación basada en motores de pasos y su control a distancia será desde cualquier lugar en donde se encuentre un dispositivo que soporte TCP/IP y que se encuentre en la misma red que la tarjeta del presente proyecto. Esta tarjeta alojará una página HTML y atenderá requisiciones HTTP, protocolo incluido en la pila TCP/IP.

La metodología empleada para alcanzar nuestro objetivo va desde un panorama general hasta llegar a nuestra aplicación específica. En el primer capítulo con el nombre de EL MODELO OSI, se toca el primer panorama general de nuestra metodología. En éste capitulo nos enfocaremos en el punto de comunicaciones entre redes para lograr la interoperabilidad entre los diversos vendedores de sistemas que adopten el estándar en sus equipos.

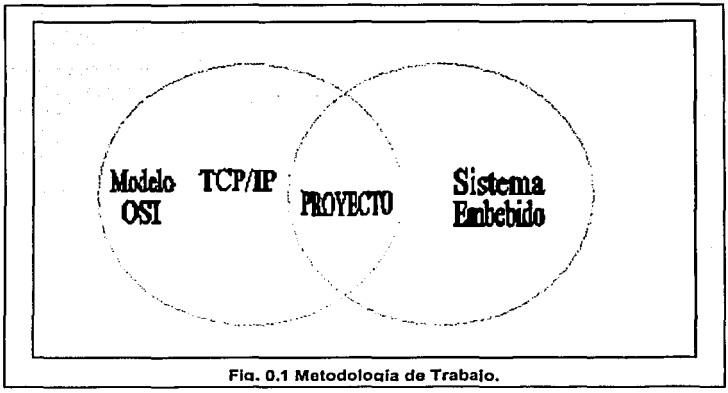


Fig. 0.1 Metodología de Trabajo.

En segunda instancia, el capítulo con el nombre LA PILA TCP/IP se adentra en un concepto mas específico sin salirnos del panorama general del modelo OSI. Este capítulo se centra en el punto de interconectividad con Internet, por lo que nos enfocaremos en sus diferentes capas, en la relación de éstas con el modelo OSI, el flujo y manejo de los datos en cada una así como los protocolos que manejan y su función en la pila TCP/IP.

A continuación, en el tercer capítulo denominado SISTEMA EMBEBIDO, tocaremos un segundo panorama general de nuestra metodología. Nos enfocaremos en un aspecto teórico de un sistema embebido haciendo hincapié en conceptos relacionados con el proyecto aquí presentado para su debida clasificación y entendimiento. Cabe mencionar que un sistema embebido puede basarse en microprocesadores de 8, 16 o 32 bits.

El cuarto capítulo, IMPLEMENTACION DE LOS PROTOCOLOS TCP/IP EN UN SISTEMA EMBEBIDO, enlaza los 2 aspectos generales ya presentados de nuestra metodología, el modelo OSI y el sistema embebido. Básicamente el presente capítulo se enfoca en el como se incorpora el código BSD (Berkeley System Distribution) en sistemas embebidos y en RTOS (Real Time Operation System).

En el quinto capítulo con el nombre de LA PILA TCP/IP Y SU IMPLEMENTACIÓN EN EL PROYECTO, así como en el resto de los capítulos, nos adentraremos en aspectos más específicos de nuestra metodología hasta aterrizar en el proyecto. Este quinto capítulo se centra en conceptos teóricos de la pila TCP/IP que tienen relación con el proyecto presentado.

El sexto capítulo se llama EL KIT DE DESARROLLO TCP/IP y explica el entorno de desarrollo empleado en nuestro proyecto tanto su software como su hardware. Y por último el séptimo capítulo denominado EL PROYECTO materializa el trabajo en general.

CAPÍTULO 1. EL MODELO OSI.

El modelo de referencia OSI (Open System Interconnect) es el modelo principal para las comunicaciones de red. Un objetivo primordial de este modelo es acelerar el desarrollo de futuros productos de red. Aunque existen otros modelos, la mayoría de los fabricantes actuales relacionan sus productos con el modelo de referencia OSI, especialmente cuando quieren educar a los usuarios en el empleo de sus productos. Lo consideran la mejor herramienta disponible para enseñar como se envían y reciben los datos en la red. Fue introducido por la Organización Internacional de Estándares (ISO) en los años 70's para promover la interoperabilidad entre los diversos vendedores de sistemas. Resulta de mucha ayuda para visualizar cómo la información viaja desde las aplicaciones (hojas de cálculo, documentos, etc..) por cualquier medio (por ejemplo, los cables), hasta otras aplicaciones en otra computadora en red.

En el modelo de referencia OSI, existen siete capas numeradas. Cada capa muestra una función particular de la red. Esta separación de las funciones de la red se llama *layering*. Dividir la red en estas siete capas proporciona las siguientes ventajas:

- Divide la comunicación de red en partes más pequeñas, sencillas y fáciles de desarrollar.
- Facilita la normalización de los componentes de la red, al permite el desarrollo y soporte de múltiples fabricantes.
- Permite que diferentes tipos de hardware y software de red se comuniquen entre si.
- Impide que los cambios en una capa afecten a las otras, por lo que cada una se puede desarrollar rápida e independientemente.
- Divide la comunicación de la red en partes más pequeñas para hacer más fácil su comprensión y entendimiento.

Las siete capas del modelo OSI son:

Capa 7: la capa de aplicación

Capa 6: la capa de presentación.

Capa 5: la capa de sesión.

Capa 4: la capa de transporte.
Capa 3: la capa de red.
Capa 2: la capa de enlace de datos.
Capa 1: la capa física.

El principio de las capas del modelo OSI, radica en que cada capa esconde sus características propias al resto de las capas. Cada capa en la máquina emisora, establece una conexión punto a punto con su correspondiente capa en la máquina receptora gracias al encapsulamiento. Cada capa en el sistema recibe paquetes de la capa inmediata inferior y transmite paquetes hacia la capa inmediata superior.

1.1 CAPA DE APLICACIÓN.

Es la capa más cercana al usuario, proporciona servicios de red, como acceso e impresión de los archivos. No proporciona servicio a ninguna otra capa del modelo OSI, sino a aplicaciones externas a él. Ejemplos de dichas aplicaciones son los programas de hojas de cálculo, los procesadores de texto y los programas de las terminales bancarias.

1.2 CAPA DE PRESENTACIÓN.

Esta capa asegura que la información que se envía a la capa de aplicación de un sistema se va a poder leer por la capa de aplicación de otro sistema. Si fuese necesario, la capa de aplicación traduce múltiples formatos de datos empleando un formato común. Esta capa también es la responsable de la compresión y el cifrado.

1.3 CAPA DE SESIÓN.

Esta capa establece, administra y finaliza las sesiones entre dos hosts. La capa de sesión proporciona su servicio a la capa de presentación. También sincroniza el diálogo entre las capas de presentación de los dos hosts y administra el intercambio de datos. Además de regular la sesión, la capa de sesión ofrece prevención para una eficiente transferencia de datos, clase de servicio y, excepcionalmente, informa de problemas en las capas de sesión, presentación y aplicación.

1.4 CAPA DE TRANSPORTE.

Esta capa segmenta los datos del sistema del host remitente y los reordena en el sistema del host receptor. El límite entre la capa de transporte y la capa de sesión puede imaginarse como el límite entre los protocolos de aplicación y los de flujo de datos. Mientras que las capas de aplicación, presentación y sesión se preocupan por los temas de la aplicación, las cuatro capas inferiores se preocupan por los temas de transporte de datos. Esta capa intenta proporcionar un servicio de transporte sin que las capas superiores se enteren de los detalles del transporte mismo. Lograr un transporte fiable es el objetivo de ésta capa. Esta capa se encarga de establecer, mantener y finalizar adecuadamente los circuitos orientados a la conexión. Al proporcional un servicio fiable, se emplea la detección y recuperación de errores en el transporte y la información en el control de flujo.

1.5 CAPA DE RED.

La capa de red es una capa compleja que proporciona conectividad y una selección de ruta entre dos sistemas host que pueden estar ubicados en redes geográficamente separadas.

1.6 CAPA DE ENLACE DE DATOS.

Esta capa, proporciona tránsito de datos a través de un enlace físico. De este modo, la capa de enlace de datos se ocupa del direccionamiento físico (lo contrario a lógico), de la topología de la red, del acceso al medio a la red y de la detección de errores. El nivel de enlace de datos define cómo se formatean los datos para transmisión y cómo se controla el acceso a la red. Este nivel ha sido dividido por el comité e estandarización IEEE 802 en dos subniveles: control de acceso al medio físico (MAC Media Access control) y control de enlace lógico (LLC, Logical Link Control).

1.7 CAPA FÍSICA.

Esta capa define las especificaciones eléctricas, mecánicas y funcionales para activar, mantener y desactivar el enlace físico entre sistemas finales. Características como niveles de voltaje, cronometraje de los cambios de voltaje, velocidad de los datos físicos, distancias máximas de transmisión, conectores físicos y otros atributos similares, se definen mediante las especificaciones de la capa física.

CAPÍTULO 2. LA PILA TCP/IP.

TCP/IP es un grupo de protocolos para mandar cadenas de datos entre computadoras diferenciadas por una dirección IP (Internet Protocol). TCP/IP puede ser utilizado para transmitir datos entre 2 computadoras, entre otras diversas computadoras conectadas en una red local, o entre 2 computadoras que están conectadas en Internet. Las cadenas de datos se dividen en paquetes y éstos son mandados uno tras otro. TCP/IP se encarga de que los paquetes lleguen a su destino de manera correcta. Los paquetes son ensamblados en el destino para obtener la cadena de datos original. Si los paquetes se pierden, son retransmitidos. TCP/IP puede funcionar eficientemente en redes grandes donde el tiempo de transmisión de un paquete puede ser muy largo, cientos de milisegundos. TCP/IP mantiene un promedio elevado de velocidad de datos en este ambiente mandando el siguiente paquete sin esperar que el primero llegue. La velocidad de transmisión y el tamaño de los paquetes son continuamente ajustados para asegurar una transmisión óptima. La definición formal del un protocolo es realizada por las autoridades encargadas de Internet, y los documentos están disponibles en la Web en forma de RFC (Request for Comments).

2.1 CAPAS DE TCP/IP.

2.1.1 CAPA DE APLICACIÓN.

Los diseñadores de TCP/IP creyeron que los protocolos de nivel superior deberían incluir los detalles de las capas de presentación y de sesión y crearon una capa de aplicación que manejaba los protocolos de nivel superior, los temas de representación, de codificación y el control del diálogo. TCP/IP combina todos los temas relacionados con la aplicación en una sola capa, asegurando así que los datos serán empaquetados correctamente por la capa siguiente.

2.1.1.1 ALGUNOS PROTOCOLOS DE LA CAPA DE APLICACIÓN.

DNS (Domain Name System). Se trata de una base de datos distribuida de nombres de dominio y direcciones IP. Un nombre de dominio es una cadena de caracteres

alfanuméricos separados en segmentos por medio de puntos. El protocolo DNS se encuentra documentado en las RFCs 1035 y 1706.

HTTP (HyperText Transport Protocol). Es el protocolo utilizado por navegadores y servidores web para transferir archivos, como archivos de texto y gráficos. Es precisamente este protocolo el utilizado en el proyecto presentado por este trabajo. El protocolo http se encuentra documentado en la RFC 1945.

TELNET. Es un protocolo de emulación de terminal de TCP/IP. Las opciones dan a TELNET la posibilidad de transferir datos binarios, soportar macros de byte, emular terminales gráficos y distribuir información para soportar una gestión centralizada de terminales. El protocolo TELNET se encuentra documentado en las RFCs 854, 855 y 857.

FTP (File Transfer Protocol). Es un protocolo que se utiliza para transferir archivos entre diferentes hosts conectados en red. El protocolo FTP se encuentra documentado en la RFC 959.

2.1.2 CAPA DE TRANSPORTE.

La capa de transporte trata normalmente con los temas de fiabilidad, control de flujo y retransmisión. Uno de sus protocolos, el protocolo para el control de la transmisión (TCP), proporciona unas maneras excelentes y flexibles de crear comunicaciones de red fiables y con gran flujo. TCP es un protocolo orientado a la conexión. Soporta diálogos entre el origen y el destino mientras empaqueta la información de la capa de aplicación en unidades que se llaman segmentos. Orientado a la conexión no significa que exista un circuito físico entre las computadoras que se comunican.

2.1.2.1 ALGUNOS PROTOCOLOS DE LA CAPA DE TRANSPORTE.

UDP. (User Datagram Protocol). El protocolo UDP intercambia datagramas sin acuse de recibo ni entrega garantizada, y necesita que otros protocolos manipulen el proceso de errores y la retransmisión. UDP está definido en la RFC 768.

TCP (Transmition Control Protocol). El protocolo TCP proporciona la transmisión de datos dúplex completamente fiable por lo que se trata de un protocolo orientado a conexión. TCP puede detectar errores y pérdida de datos y puede activar la retransmisión de datos si éstos no se reciben completos y sin errores. TCP está definido en las RFCs 793, 1072, 1693, 1146 y 1323.

ICMP (Internet Control Message Protocol). ICMP es un grupo de mensajes que informa de errores y otra información relativa al procesamiento de paquetes IP y que requieren de atención. Se encuentra documentado en la RFC 792. Actúa tanto en IP como en TCP y UDP. Y algunos mensajes ICMP son regresados a los protocolos de aplicación.

2.1.3 CAPA DE INTERNET.

El propósito de la capa de Internet es enviar los paquetes desde el origen de cualquier red de la internetwork, y que lleguen a su destino independientemente de la ruta y las redes intermedias que utilicen para conseguirlo. El protocolo específico que gobierna esta capa se llama protocolo de Internet (IP) y se encarga principalmente del ruteo de paquetes.

2.1.3.1 ALGUNOS PROTOCOLOS DE LA CAPA DE INTERNET.

IPv4 (Internet Protocol). Es un protocolo de capa de red de la pila TCP/IP que ofrece un servicio de Internetwork sin conexión. IP proporciona funciones para el direccionamiento, la especificación del tipo de servicio, la fragmentación y el reensamblado y la seguridad. Se define en las RFCs 791 y 1853.

IPv6. Es una versión actualizada del protocolo Internet basada en IPv4. IPv6 incrementa el tamaño de las direcciones IP de 32 bits a 128 bits, para soportar mas niveles de jerarquía de direccionamiento, un número mucho mayor de nodos direccionables y una auto configuración más simple. Se define en las RFCs 1883, 1826 y 1827.

ARP (Address Resolution Protocol). El protocolo ARP se encarga de traducir direcciones virtuales en direcciones físicas. El hardware de la red no entiende

direcciones IP. El protocolo IP utiliza al protocolo ARP para cambiar las direcciones IP (32 bits) en direcciones físicas que cuadren con el esquema en el que se base el hardware. Para el caso de nuestro proyecto, el protocolo ARP traduce las direcciones IP en direcciones MAC, debido a que se trata de el esquema Ethernet. ARP está definido en las RFCs 826, 1390 y 1293.

2.1.4 CAPA DE ACCESO A LA RED.

Esta capa se ocupa de todos los temas que un paquete IP necesita para cruzar un enlace físico desde un dispositivo a otro que esté conectado directamente. Incluye detalles de las tecnologías LAN y WAN, y todos los detalles de las capas de enlace de datos y físicas del modelo OSI.

2.1.4.1 ALGUNOS PROTOCOLOS DE LA CAPA DE ACCESO A LA RED.

SNAP (SubNetwork Access Protocol). Protocolo empleado para encapsular datagramas IP y solicitudes y respuestas ARP en redes IEEE 802. Los datagramas IP se envían en las redes IEEE 802 encapsulados dentro de los niveles de enlace de datos SNAP y LLC 802.2 y los niveles de red físicos 802.3, 802.4 u 802.5. La cabecera SNAP sigue a la cabecera LLC y contiene un códio de organización que indica que los siguientes 16 bits especifican el código EtherType.
El protocolo SNAP está definido en el RFC 1042.

LLC. (Logical Link Control). La más alta de las 2 subcapas de la capa de enlace de datos definidas por la IEEE. La subcapa LLC manipula el control de errores, control de flujo, entramado y direccionamiento de subcapa MAC. El protocolo LLC más extendido es IEEE 802.2, que incluye variantes sin conexión y orientadas a la conexión. Esta capa de enlace lógico permite que la capa de enlace de datos funcionara independientemente de las tecnologías existentes. El protocolo LLC está definido en el RFC 2364.

MAC (Media Access Control). Es la más baja de las 2 subcapas de la capa de enlace de datos definidas por la IEEE. La subcapa MAC trata los protocolos que sigue un host para acceder al medio físico, define como transmitir tramas en el cable físico y además gestiona el direccionamiento físico asociado a cada dispositivo, la definición de la

topología de red y la disciplina de la línea. El IEEE 802.3 (Ethernet) es uno de los protocolos MAC que se pueden emplear. Ethernet es un estándar para redes de comunicación de datos ampliamente utilizado, desarrollado por DEC, Intel y Xerox. Utiliza una topología de bus, y CSMA/CD como método de acceso. Los términos Ethernet y <<estándar IEEE 802.3>> se utilizan a menudo de manera indistinta.

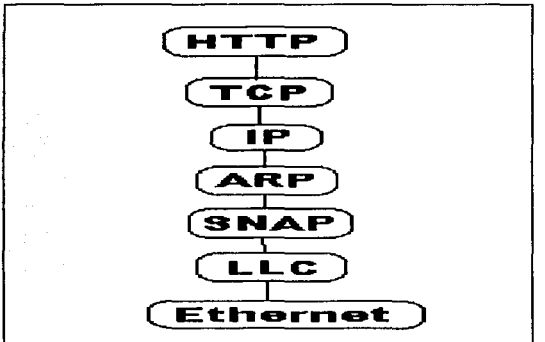


Fig. 2.1. Relación de algunos protocolos TCP/IP

2.2 NOMBRES DE LOS DATOS POR CAPA .

Para que los paquetes de datos viajen desde el origen hasta el destino, cada capa del modelo OSI del origen debe comunicarse con la misma capa del destino. Esta forma de comunicación se le llama comunicación de igual a igual. Durante este proceso, cada protocolo de capa intercambia información, llamada unidad de datos de protocolo (PDU), entre las capas iguales. Cada capa de comunicación de la computadora de origen se comunica mediante una PDU específica de cada capa, con su capa igual de la computadora destino.

Tabla 2.1. RELACION ENTRE CAPAS, PROTOCOLOS TCP/IP y PDU.			
CAPA OSI	CAPA TCP/IP	PROTOCOLOS TCP/IP	PDU
APLICACIÓN PRESENTACION SESIÓN	APLICACIÓN	HTTP, FTP, PING, DNS, etc...	DATOS
TRANSPORTE	TRANSPORTE	ICMP, TCP, UDP	SEGMENTOS
RED	INTERNET	IP, ARP	PAQUETES
ENLACE	ACCESO A LA RED	MAC, LLC	TRAMAS
FISICA		Tecnologías LAN	BITS

2.3 ELEMENTOS ESENCIALES PARA EL FLUJO DE DATOS.

Para que la red tenga unas comunicaciones fiables, se deben enviar los datos mediante unos paquetes manejables e identificables. Esto se consigue a través del proceso de encapsulado. Una breve revisión del proceso nos indica que las tres capas superiores (aplicación, presentación y sesión) del modelo OSI, preparan las datos para la transmisión, creando un formato común para transmitir. La capa de transporte divide los datos en unidades de tamaño manejable que se llaman segmentos. También asigna números de secuencia a los segmentos para asegurar que el host receptor ponga los datos de nuevo en el mismo orden. La capa de red encapsula el segmento creando un paquete, y le añade un destino y una dirección de red de origen, normalmente IP.

La capa de enlace de datos vuelve a encapsular el paquete y crea una trama, a la que añade una dirección de origen y una dirección de destino local (MAC). Después, la capa de enlace de datos transmite los bits binarios de la trama sobre los medios de la capa física.

Cuando los datos se transmiten sólo por una LAN, llamamos tramas a las unidades de datos, porque las direcciones MAC son todo lo que se necesita para obtener el host de origen y de destino. Pero si tenemos que enviar los datos a otro host mediante una Intranet o Internet, los paquetes o datagramas se convierten en las unidades de datos a las que nos referimos. Esto se debe a que la dirección de red del paquete contiene la dirección del host al que se están enviando los datos (paquete), mientras que la información de la capa de enlace de datos es local. O sea, cambia mientras pasa a

través de cada red. Las tres capas inferiores (red, enlace de datos, física) del modelo OSI son los principales manipuladores de datos en una Intranet o en Internet.

2.3.1 FLUJO DE PAQUETES A TRAVÉS DE DISPOSITIVOS CAPA 1.

El flujo de paquetes a través de los dispositivos de Capa 1 es sencillo. Los medios físicos se consideran componentes de Capa 1. Sólo se ocupan de los bits (por ejemplo, voltaje o pulsos de luz). Si los dispositivos de Capa 1 son pasivos (por ejemplo, plugs, conectores, jacks, patch panels, medios físicos), todos los bits viajan a través de los dispositivos pasivos con un mínimo de distorsión. Si los dispositivos de Capa 1 son activos (como repetidores o hubs), los bits se regeneran y se reenvían. Los transceptores, también conocidos como dispositivos activos, pueden actuar como adaptadores de conector (puerto AUI para RJ-45) o convertidores de medios (RJ-45 eléctrico para SST Optical), además de transmitir y recibir en los medios. En todos los casos, los transceptores actúan como dispositivos de Capa 1. Ningún dispositivo de Capa 1 examina las cabeceras o los datos de un paquete encapsulado. Solo trabajan con los bits.

2.3.2 FLUJO DE PAQUETES A TRAVÉS DE DISPOSITIVOS CAPA 2.

Es importante recordar que los paquetes están contenidos en las tramas, así que para comprender cómo viajan los paquetes en los dispositivos de Capa 2, ellos van a trabajar directamente con la trama, o bien, con la forma encapsulada de los paquetes. Cualquier cosa que le ocurra a la trama también le ocurrirá al paquete. Las NIC (Network Interface Card), los puentes y los switches tienen en cuenta la información de las direcciones (MAC) de enlace de datos para dirigir las tramas, lo que significa que son dispositivos de Capa 2. La única dirección MAC reside en una NIC.

Los puentes examinan las direcciones MAC de las tramas entrantes. Si la trama es local (con una dirección MAC en el mismo segmento de red que el puerto entrante del puente), la trama no se envía a través del puente. Si la trama no es local (sin una dirección MAC en el puerto entrante del puente), se envía al siguiente segmento de red. Se puede decir que un switch es un hub con puertos individuales que actúan como

puentes. El switch toma una trama de datos, la lee, examina la dirección MAC de Capa 2 y envía las tramas (las conmuta) a los puertos adecuados.

2.3.3 FLUJO DE PAQUETES A TRAVÉS DE DISPOSITIVOS CAPA 3.

El dispositivo principal que tratamos en la capa de red es el router. Los routers operan en la Capa 1 (bits en los medios de las interfaces del router), en la Capa 2 (tramas conmutadas de una interfaz a otra), y por último en la Capa 3, basándose en la información del paquete (decisiones de enrutamiento). Los paquetes fluyen a través de los routers (selección de la mejor ruta y la conmutación actual al puerto de salida adecuado), involucrando el uso de las direcciones de la Capa 3. Después de seleccionar el puerto de salida, el router encapsula de nuevo el paquete en una trama para enviarlo a su próximo destino. Este proceso sucede en cada router de la ruta desde el host de origen hasta el host de destino.

2.3.4 FLUJO DE PAQUETES A TRAVÉS DE DISPOSITIVOS CAPAS 1 A 7.

Algunos dispositivos (como la PC) son dispositivos de las Capas 1 a 7. En otras palabras, ejecutan procesos que se pueden asociar con cada capa del modelo OSI. Para que la PC se comunique a Internet, es necesario que procese bits de llegada a la NIC, analice la trama para ver si es ella la receptora del paquete, posteriormente analice el paquete para extraer los datos y entregárselos a la aplicación que lo haya requerido. O bien en el sentido inverso, encapsule los datos de las capas superiores en tramas, para mandarlos por la NIC en forma de bits.

2.3.5 FLUJO DE PAQUETES A TRAVÉS DE NUBES.

Las nubes pueden contener varios tipos de medios: NIC, switches, puentes, routers, gateways y otros dispositivos de red. Como una nube no es realmente un dispositivo, sino una colección de dispositivos que opera en todos los niveles del modelo OSI, se puede clasificar como un dispositivo de las Capas 1 a 7.

Como ejemplo de dicha clasificación, se puede seguir la ruta de los datos generada por el comando ping. Este comando envía algunos datos IP al dispositivo que

especifique en dicho comando. Si el dispositivo esta configurado correctamente, la pregunta regresa. Si recibe una respuesta, sabrá que el dispositivo existe y está activo. Si no recibe una respuesta, puede asumir que hay un problema en algún lugar entre su host y el de destino.

CAPÍTULO 3. SISTEMA EMBEBIDO.

Se puede decir que un sistema embebido es una solución basada en una arquitectura de computador que presta una función específica dentro de un sistema complejo. La denominación de sistemas embebidos refleja que son una parte integral del sistema y, en general, son dispositivos utilizados para controlar o asistir la operación. De acuerdo al tipo de aplicación, los sistemas embebidos pueden ser interactivos o reactivos. Los primeros están relacionados con el ambiente de tal manera que reciben órdenes de los usuarios, las procesan y retornan a ellos los resultados. Los sistemas reactivos se relacionan con un ambiente externo no humano, reaccionando constantemente a los estímulos provenientes del ambiente con acciones que generan eventos y otros estímulos. A la vez, un sistema embebido es de tiempo real cuando el funcionamiento correcto depende de que las acciones se ejecuten en intervalos de tiempo restringidos.

El sistema de conexión a Internet para dispositivos electrónicos es un sistema embebido dedicado a una tarea específica con software y hardware especializado debido a que integra otro sistema electrónico mayor. Existen diversas áreas críticas que pueden tener gran cantidad y variedad de sistemas embebidos como pueden ser:

- Salud pública.
- Suministro de Energía.
- Oleoductos y gasoductos.
- Control de desechos radioactivos.
- Plantas atómicas y nucleares.
- Telecomunicaciones y transporte.
- Suministro de agua y desagüe.
- Automatización Industrial de diversos equipamientos.

En el caso específico de la aplicación propuesta en el presente trabajo, debido a que se trata del control a distancia de un motor de pasos y se requiere la intervención directa del usuario, podemos darle la clasificación de sistema embebido interactivo. Y también, como se requiere la pronta respuesta del motor una vez hecha la requisición, podemos clasificarlo también como de tiempo real.

CAPÍTULO 4. IMPLEMENTACIÓN DE LOS PROTOCOLOS DE LA PILA TCP/IP EN UN SISTEMA EMBEBIDO.

Entrando a las raíces de la pila TCP/IP, en primera instancia fue la DARPA (Defence Advanced Research Projects) la encargada de implementar la pila sobre el sistema operativo de BSD (Berkeley System Distribution). Poco tiempo después, la DARPA requirió que todas las computadoras conectadas a Internet corrieran los protocolos TCP/IP. Gracias a esto, TCP/IP fue implementado como parte de BSD y de ahí, a muchas estaciones de trabajo, así como a muchas mini-computadoras. Los sistemas embebidos también se vieron afectados por esto, y muchos productos RTOS (Real Time Operating System) migraron los códigos fuentes de TCP/IP a sus sistemas embebidos. Hoy en día gran parte de los productos que utilizan TCP/IP contienen el código inalterado de Berkeley junto con el Copyright. Por todo esto, el software implementado en el proyecto conserva las bases implementadas por la BSD, con algunas adaptaciones. Esta sección del presente trabajo se centra en la manera en como se programa la pila TCP/IP en un sistema embebido tomando como base la BSD en su versión 4.3. Por lo anterior se hará referencia a estructuras y funciones contenidas en dicha distribución que tienen un papel de importancia en la pila TCP/IP. Las referencias a dichas funciones y estructuras se presentarán con el siguiente formato: ***ejemplo***.

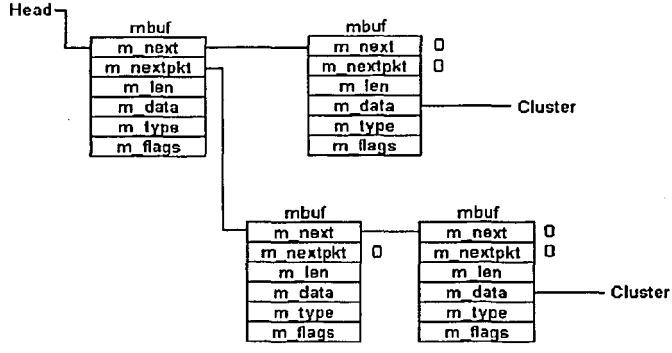
4.1 MANEJO DEL BUFER.

Debido a que TCP/IP convierte los datos en paquetes, se hace necesaria la creación de buffers que almacenen dichos paquetes, y los datos en ellos. También debido a la naturaleza inherente asíncrona de la red, los buffers deben emplearse dinámicamente. Tienen diferentes tiempos de vida dependiendo de muchos factores como la transmisión a través de la red, o si el paquete es parte de una cadena en la transmisión orientada a conexión, o si el paquete es un datagrama, en la transmisión sin conexión. La comprensión del manejo del buffer es fundamental en la implementación del protocolo.

El mecanismo básico para el manejo de bufers utilizado en el TCP/IP es el implementado por Berkeley y es conocido como "mbuf" o buffers de memoria. El

mecanismo mbuf puede expandirse para permitir distintos tamaños de paquetes que son frecuentemente utilizados en comunicaciones IP.

Pequeñas cantidades de datos pueden colocarse directamente en el área de datos del mbuf. Para cantidades mayores, se requiere que los mbufs se expandan con el empleo de clusters, que son estructuras de datos que retienen información para aumentar el manejo de datos de un solo mbuf. Varios mbufs pueden unirse para facilitar la implementación de paquetes.



m_next:	Siguiente mbuf en la cadena mbufs. Una cadena de 1 o mas mbufs se emplea para almacenar un paquete sencillo.	
m_nextpkt:	Siguiente mbuf en la cola de paquetes.	
m_len:	Longitud total del mbuf incluyendo cluster, de existir alguno.	
m_data:	Dirección de inicio de datos.	
m_flags:	**0 flags**	El mbuf contiene sólo datos hasta 108 bytes.
	M_PKTHDR	El mbuf contiene un paquete de encabezado y hasta 100 bytes de datos.
	M_EXT	El mbuf se extiende a un cluster, permitiendo hasta 2,048 bytes de datos

Fig. 4.1. Estructura de "mbuf".

El mbuf básico contiene información de encabezados y un área de datos que puede extenderse con un cluster si un paquete no cabe en dicha área. Los mbufs son convenientes también para almacenar información de control o algún otro dato temporal utilizado en los protocolos.

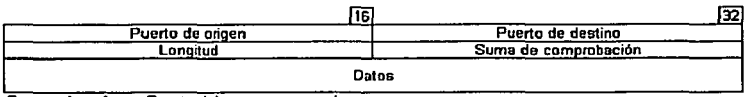
4.2 EL VIAJE DE LOS DATOS HACIA CAPAS INFERIORES.

La mejor manera de entender cualquier protocolo es siguiendo el viaje de los datos desde la capa de aplicación de la máquina emisora, pasando por el cable (medio físico) hasta la máquina receptora. Típicamente una aplicación escribe los datos en un socket. Un socket puede definirse como una pareja de puntos finales lógicos. Uno de los puntos se encuentra en la máquina emisora y el otro en la máquina receptora. A continuación se explicará el viaje de los datos desde la capa de aplicación hasta la capa física.

En primera instancia, la capa de aplicación o de usuario crea un socket con la llamada a la función del sistema ***socket().***

4.2.1 ENVÍO DE DATOS VIA UDP (Transmisión sin conexión).

Asumamos que el usuario desea mandar un simple mensaje a otro sistema vía UDP o protocolo de datagramas. Para esto la aplicación especifica el protocolo ***SOCK_DGRAM*** en la llamada a la función del sistema ***socket().*** Por ejemplo, el usuario invoca la función ***sendmsg()*** pasando el apuntador del mensaje como argumento. El usuario llama a bloques hasta que existe suficiente espacio para el mensaje en el buffer de envío del protocolo UDP. Cuando exista espacio suficiente, el mensaje es colocado dentro de una cadena de mbufs hasta que el mensaje completo está listo para la transmisión. El protocolo UDP entonces antepone su encabezado a los datos, calcula el checksum y manda llamar la función ***ip_output()*** para mandar el segmento. A continuación se muestra el encapsulado UDP.



Puerto de origen: Puerto del proceso transmisor
Puerto de destino: Tiene significado dentro de una dirección Internet destino concreta.
Longitud: longitud en octetos de este datagrama de usuario, incluyendo esta cabecera y los datos.
Suma de comprobación: Complemento a uno de 16 bits de la suma en complemento a uno de una pseudocabecera compuesta por información de cabeceras IP, UDP y de los datos.
Datos: Campo de datos UDP.

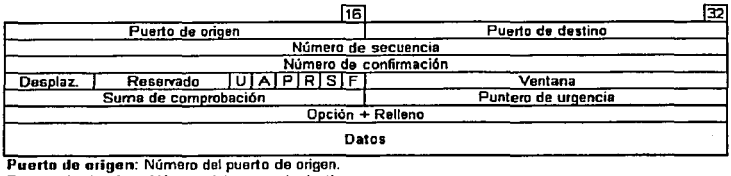
Fig. 4.2. Estructura de la cabecera UDP.

4.2.2 ENVÍO DE DATOS VÍA TCP (Transmisión con conexión)

En caso de que la transmisión no sea vía UDP sino TCP, se especificará el protocolo ***SOCK_STREAM*** en la llamada a la función del sistema ***socket().*** En la máquina que desea utilizar un servicio remoto, la función ***connect()*** establece una conexión entre el socket local y el punto final remoto. La función ***connect()*** espera hasta que la conexión se establezca. El protocolo TCP mantiene una máquina de estados para llevar rastro de la conexión. Una vez que la función ***connect()*** regresa, el socket está listo para recibir datos.

Es ahora cuando la aplicación puede escribir datos en llamadas a funciones que lo soporten como ***sendmsg().*** Los datos del usuario son acumulados en el buffer de envío de TCP. Cuando la cadena de sockets recibe datos, los rompe en paquetes y utiliza buffers para alojarlos. Los encabezados TCP son antepuestos a los datos y el segmento se pasa a la capa IP. Los datos son separados en paquetes individuales para asegurarse que la longitud de los datos es menor que el MTU. Algunos de los campos son utilizados para mantener rastro del estado de la conexión. Por ejemplo, TCP utiliza los campos de número y de conocimiento (acknowledgement) para mantener un orden de los datagramas individuales. Los bits de estado (Finish, Synch, Reset, Push, Acknowledge, y Urgent) son utilizados por los protocolos en ambos extremos para rastrear el estado de la conexión y manejar el establecimiento y ruptura de la transmisión. Es importante hacer notar que la transmisión de datos es asíncrona y manejada por buffers. El sistema embebido debe tener algún tipo de manejo de tiempo para retransmitir de existir pérdida

de paquetes. También debe tener un mecanismo de manejo de buffers dinámico para almacenar paquetes hasta que deban retransmitirse o descartarse. A continuación se muestra el encapsulado TCP.



Puerto de origen: Número del puerto de origen.
Puerto de destino: Número del puerto de destino.
Número de secuencia: Número de secuencia del primer octeto de datos de este segmento (excepto cuando está presente SYNC). Si está presente SYNC, el número de secuencia es el número inicial de secuencia (ISN, initial sequence number) y el primer octedo de datos es ISN+1.
Número de Confirmación: Si el bit de control ACK está activado, este campo contiene el valor del siguiente número de secuencia que el emisor del segmento está esperando recibir. Una vez establecida una conexión, este valor se envía siempre.
Desplazamiento de datos: Número de palabras de 32 bits que componen la cabecera TCP.
Reservado: Para uso futuro. Deben ser cero.
Bits de Control: 6 bits de control, Urgencia, Confirmación, Empuje, Reinicio, Sincronización y Fin.
Ventana: Número de octetos de datos que el emisor está dispuesto a aceptar.
Suma de comprobación: Complemento a uno de 16 bits de la suma en complemento a uno de todas las palabras de 16 bits contenidas en la cabecera y en el texto.
Puntero de urgencia: Sólo con el bit de Urgencia activado, indica el desplazamiento positivo a partir del número de secuencia de este segmento y apunta al octeto que sigue a los datos de urgencia.
Opciones: Hay 2 posibles formatos para una opcion: Un único octeto de tipo de opción y un octeto de opción, un octeto de longitud de opción y los octetos de datos de la opción.
Datos: Datos TCP o protocolo de nivel superior.

Fig. 4.3. Estructura de la cabecera TCP.

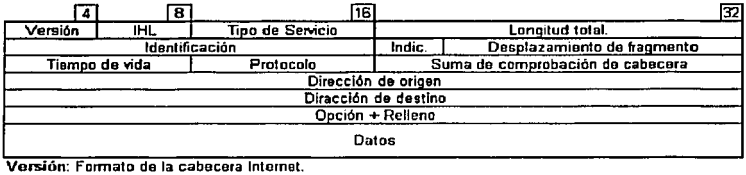
4.2.3 ENCAPSULADO Y FRAGMENTACIÓN EN LA CAPA DE IP.

La capa IP antepone sus encabezados a los datos que recibe de la capa de transporte y establece la ruta de los paquetes de acuerdo a sus tablas de ruteo, insertando la dirección apropiada en el encabezado IP. Posteriormente calcula el checksum y limpia el campo del tiempo de vida (TTL) para paquetes originados en las capas superiores.

La capa IP puede configurarse para auto encaminar los paquetes recibidos en una interfaz hacia otra. Esta es una característica fundamental que puede utilizarse cuando la

capa IP es multiplexada en 2 o más interfaces en la capa física. IP tiene tablas de ruteo que determinan hacia que puerta de enlace encaminar los paquetes para que éstos alcancen su destino final.

El encapsulado IP se muestra en la siguiente figura:



Versión: Formato de la cabecera Internet.
IHL (Internet Header Length): Longitud de la cabecera Internet en palabras de 32 bits.
Tipo de servicio: Calidad de servicio deseada. Las redes pueden ofrecer una característica de precedencia de servicio, lo que significa que aceptan tráfico sólo por encima de una cierta precedencia en los momentos de alta carga entre tres variables: bajo retardo, alta fiabilidad y alta tasa de transferencia.
Longitud total: Longitud del datagrama medida en bytes incluyendo cabecera Internet y datos.
Identificación: Valor de identificación asignado por el emisor para ayudar a recomponer los fragmentos de un datagrama.
Indicadores: Bits indicadores que permiten o no la fragmentación del datagrama e identifican al último fragmento.
Desplazamiento de fragmento: Indica la posición de este fragmento en el datagrama.
Tiempo de vida: Tiempo máximo de autorización del datagrama en el sistema.
Protocolo: Indica el protocolo utilizado por el siguiente nivel quien recibirá los datos.
Suma de comprobación de la cabecera: suma de comprobación sólo de la cabecera.
Dirección de origen/destino: Direcciones IP de origen y destino.
Opciones: Hay 2 posibles formatos para una opcion: Un único octeto de tipo de opción y un octeto de opción, un octeto de longitud de opción y los octetos de datos de la opción.
Datos: Datos IP o cabecera de protocolo de nivel superior.

Fig. 4.4. Estructura de la cabecera IP.

4.2.4 ENCAPSULADO EN LA CAPA DE ENLACE.

En la siguiente figura mostraremos en encapsulado del tipo IEEE 802.3 que soporte varias capas de enlace además de la Ethernet.

Destino	Origen	Longitud	Unidad de datos+ relleno	FCS
(6 bytes)	(6 bytes)	(2 bytes)	(46 - 1500 bytes)	(4 bytes)

Destino: Dirección destino
Origen: Dirección origen
Longitud/Tipo: En el protocolo Ethernet, el valor (>=0x0600 Hex) de este campo es Ethernet List, indicando el protocolo allí contenido.
Unidad de datos + relleno: Protocolo LLC.
FCS: Secuencia de comprobación de trama (Frame Check Sequence).

Fig. 4.5. Estructura de la cabecera Ethernet.

Se necesita tener cuidado en la elección del tipo de encapsulado en la capa de enlace debido a que el encapsulado BSD 4.3, que maneja Ethernet del tipo II, de querer posteriormente utilizarlo con otra capa de enlace diferente, generará problemas por esta confusión.

Si se utiliza una capa de enlace multiplexada, la capa IP puede recibir paquetes de diferentes instancias de la capa de enlace, donde cada capa de enlace pasa paquetes a determinada implementación en la capa física. Para cada interfaz que apunte hacia IP, IP llama a la función de salida para transmitir el paquete. El paquete se pasa como tal más un apuntador a la dirección destino en una estructura ***sockaddr{}.*** Para controladores Ethernet se convierte la dirección IP, en el apuntador de la estructura ***sockaddr{},*** en dirección MAC con la función ***addrResolve()*** dentro de la otra función ***ethernetOutput(),*** y esta última contenida en la llamada a la función de salida. Esta transformación de las direcciones IP en MAC la hace el protocolo ARP . Cuando la capa de enlace llama al protocolo ARP, se pasan apuntadores hacia el paquete y hacia la estructura ***sockaddr{},*** entonces mira en sus tablas para ver si la dirección ha sido resuelta con anterioridad, de lo contrario manda un paquete general (broadcast) que contiene la dirección IP esperando que algún servidor ARP en la red conozca la dirección MAC que corresponde a esa dirección IP. Al ser resuelta la correspondencia, manda el paquete hacia la interfaz para la transmisión con la dirección MAC correcta.

Hasta este punto cualquier implementación en la capa de IP acerca de ruteo, fragmentación o direccionamiento IP, se percibe como meros datos en la capa de enlace para ser encerrados en paquetes del tipo LLC. La capa de enlace coloca la dirección destino en el paquete que recibe desde la capa IP, misma que conoce dicha dirección por sus tablas de ruteo. Ahora la dirección de la fuente (emisor) se coloca en el paquete y la longitud es calculada. Para TCP/IP gran parte de los encabezados LLC y SNAP es codificada (hard coded) como se muestra en la figura siguiente. El campo para el tipo es reservado para el protocolo dependiendo si los paquetes son IP, ARP o RARP.

DSAP	SSAP	Control	Información LLC
(1 byte)	(1 byte)	(1 o 2 bytes)	

DSAP: Punto de acceso a servicio de destino (Destination Service Access Point)
SSAP: Punto de acceso a servicio de origen (Source Service Access Point).
Control.- Información y supervisión de la trasferencia.
Información LLC: Datos LLC o protocolos de nivel superior.

Fig. 4.6. Codificación LLC.

Se puede pensar que la capa de enlace posee dos partes, la superior y la inferior. La superior maneja el encapsulado y el multiplexado con las capas superiores. La inferior maneja la interfaz del dispositivo incluyendo DMA y manejo de interrupciones. Después de anteponer el encabezado LLC a los datos, la parte inferior de la capa de enlace toma todo como datos. La parte inferior configura el DMA y el hardware para la transmisión del paquete.

Código de organización	EtherType
(3 bytes)	(2 bytes)

Código de organización: Fijado a 0
EtherType: Especifica qué protocolo está encapsulado detro de la red IEE 802
IP = 2048, ARP = 2054.

Fig. 4.7. Estructura de cabecera SNAP.

4.2.5 CAPA FISICA.

El manejo de la capa física lo establecen en realidad los controladores de las interfaces de Red en la parte inferior de la capa de enlace. El marco ahora esta listo para la transmisión y es colocado típicamente en un buffer circular en el hardware DMA.

4.3 EL VIAJE DE LOS DATOS HACIA CAPAS SUPERIORES.

4.3.1 RECEPCIÓN EN LA CAPA DE ENLACE.

Los datos son recibidos y colocados en un buffer circular por el hardware DMA. Posteriormente la capa de enlace debe determinar el tipo de paquete. Si un paquete recibido viene de una interfaz Ethernet, puede tener o no marcos del tipo 802.3 o de Ethernet del tipo II. Para distinguir entre ambos, el campo de longitud en el tipo 802.3 está fuera de lugar conforme el campo de tipo en el tipo Ethernet II. Los tipos permitidos se muestran a continuación:

Tabla 4.1 Tipos de paquetes para IP	
800	IP
806	ARP
835	RARP

Si cualquiera de estos tipos fueran interpretados como longitud, representarían un paquete mayor que el MTU de Ethernet (1500 bytes). La función de entrada de la interfaz compara la dirección MAC del paquete recibido con la dirección broadcast. Si el paquete resulta ser del tipo broadcast, se manda a cada protocolo atado en esta interfaz. Si los bits multicast son configurados por esta interfaz, el paquete se pasa a cualquier protocolo que requiera paquetes multicast. Si el paquete recibido es del tipo unicast, el campo de tipo determina que protocolo recibe el marco. Si se tiene un protocolo cuyo controlador desea recibir todos los marcos de manera promiscua, los recibirá directamente desde la capa de enlace. Si el paquete recibido es del tipo ARP se mandará llamar la función de entrada del protocolo ARP, de igual manera se mandará llamar la función de entrada del protocolo IP si el paquete es del tipo IP.

4.3.2 RECEPCIÓN EN IP.

En primera instancia, IP compara la dirección IP destino con su misma dirección IP y su submáscara de red para ver si el paquete viene para él. IP deberá determinar ahora si el paquete viene fragmentado y requiere reensamblarse, o si se trata de un datagrama IP completo. IP debe saber que transporte recibió el paquete. Puede ser un datagrama UDP o puede ser TCP. IP sabe esta información cuando checa el identificador de protocolo en el encabezado IP. Si el paquete lo requiere es reensamblado y enviado al multiplexado de transporte al igual que los datagramas. Los datos son encaminados al transporte adecuado por las tablas de switcheo.

4.3.3 RECEPCIÓN EN UDP.

Si los datos son encaminados a UDP por las capas inferiores, son recibidos por la función de entrada UDP, ***UDP_Input()***. Primero el encabezado IP es separado y almacenado para su uso posterior. Se confirman tanto el tamaño UDP como el checksum, que vienen en los datos del encabezado recién salvado. Ahora se checa si el paquete es del tipo multicast o broadcast, en cuyo caso el paquete es colocado en todos los sockets de entrada que concuerden con la requisición. Si los datos son del tipo unicast pero con un número de puerto erróneo, se manda un paquete ICMP indicando que el puerto es inalcanzable. De otra manera, si todo está correcto, el paquete es colocado en el socket de entrada que se requiera.

4.3.4 RECEPCIÓN EN TCP.

Primeramente TCP checa el checksum, luego se checa el timestamp del paquete para ver el estado de la conexión. TCP debe decidir si se deben mandar más datos hacia el otro extremo o si el tiempo se acabó. Si el paquete recibido tiene datos y no conocimiento (acknowledgement), los encabezados IP y TCP son desechados y los datos son colocados en los bufers de entrada de los sockets habilitados.

CAPÍTULO 5. LA PILA TCP/IP Y SU IMPLEMENTACIÓN EN EL PROYECTO.

A continuación se hace referencia al como se implementó nuestro proyecto utilizando la Pila TCP/IP.

Tabla 5.1. Protocolos empleados en el proyecto.	
CAPA TCP/IP	**PROTOCOLO**
Capa de Acceso al Medio	Ethernet
Capa de Internet	IP, ARP
Capa de Transporte	TCP
Capa de Aplicación	HTTP

5.1 ETHERNET (Capa Acceso a la Red).

En primera instancia, hablando de los fierros (Capa Física), el cable utilizado en nuestro proyecto es el conocido como par trenzado sin apantallar (TIA/EIA UTP de categoría 5), con sus terminaciones RJ-45. Para las primeras pruebas del proyecto presentado se empleó un cable con la configuración conocida como cruzada y este cable representa nuestro dominio de colisión (Área de la red donde se originan y colisionan los paquetes de datos). Para las pruebas se formó una red de entorno punto a punto. Si se desease conectar más hosts a la red, o bien conectar la tarjeta a la red de redes, se puede emplear un concentrador (hub) y por consiguiente, la configuración del cable empleado deberá ser recta y se ampliaría el dominio de colisión, formando un tipo de red de entorno de medio compartido. En el caso de que se deseara implementar una red más grande, la especificación TIA/EIA-568-A normaliza los conceptos que se deben tener en cuenta para su construcción.

Hablando de tecnologías LAN, la implementada en este trabajo es la conocida como Ethernet, por consiguiente, la norma LAN Empleada es la IEEE 802.3, que especifica el cableado y la señalización en las capas física y de enlace de datos del modelo de referencia OSI. Más en específico, la especificación Ethernet empleada es 10BASET, de banda base de 10 Mbps, misma que utiliza una topología física en estrella

extendida, pero con topología lógica de bus (todos los dispositivos de la red ven las señales de todos los demás dispositivos).

El método de acceso que emplea Ethernet, y por lo tanto nuestro proyecto, es el CSMA/CD (Carrier Sense Multiple Access / Collision Detection) y activa 3 funciones.

1. Transmitir y recibir paquetes de datos.
2. Decodificar los paquetes de datos y verificar sus direcciones válidas antes de que pasen a las capas superiores del modelo OSI.
3. Detectar los errores en los paquetes de datos o en la red.

La señalización que utiliza Ethernet es la conocida como la Manchester, que define un cero como una señal que es mayor durante la primera mitad de un periodo y menor durante la segunda mitad. Para materializar en nuestro proyecto los conceptos de capa 1 y 2 del modelo OSI o bien la capa de Acceso al medio en TCP/IP, en el kit de desarrollo utilizado viene incorporado el chip RTL8019 de la compañía Realtek que es compatible con nuestra normalización IEEE 802.3, las especificaciones generales de dicho componente se encuentran anexo A del presente trabajo.

5.2 ARP, IP y TCP (Capas de Internet y Transporte).

Para la materialización de las capas de Internet y Transporte empleamos el entorno de desarrollo de la compañía Z-WORLD conocido como DynamicC. Gracias a este software no fue necesario conocer a fondo el enlace entre los protocolos: IP, ARP y TCP y su implementación en las líneas de código. La asociación entre las compañías Z-WORLD y Rabbit Semiconductor produjo el entorno de trabajo del presente proyecto, ensamblando Hardware (Rabbit Semiconductor) y Software (Z-WORLD). En otras palabras, las características específicas de la tarjeta utilizada están contempladas en las líneas de código del software utilizado.

5.2.1 ARP

La librería ARP.lib que se encuentra dentro de la paquetería de DynamicC, fue la encargada de lidiar con este protocolo en el presente proyecto.

5.2.2 IP

En el presente proyecto como en todas las aplicaciones a base de la pila TCP/IP, IP es el protocolo empleado para el caso de la capa de red (INTERNET). Un aspecto de mucha importancia para el protocolo IP es la asignación y administración de puertos para comunicarse. Se utiliza un puerto cuando 2 computadoras quieren hablar entre ellas sin importar el protocolo empleado en la capa de Transporte. El servidor que ofrece el servicio selecciona un puerto y espera por la conexión. El cliente selecciona un puerto arbitrario para comunicarse, y coloca el número de puerto, la dirección IP de origen y la dirección IP destino en el paquete IP. Enfocándonos en nuestro proyecto, nuestro servidor emplea el puerto 80 (Estándar en el protocolo HTTP) para esperar conexiones de clientes y las requisiciones del cliente acceden a él para comunicarse, intercambiar datos, finalizar la conexión y comenzar otra si el cliente lo requiere.

5.2.3 TCP

La razón por la que nuestro proyecto incorpora TCP como protocolo de la capa de transporte se debe a la naturaleza de nuestra aplicación. Nuestra aplicación requiere garantizar los datos que circulan por el canal de comunicación porque de otra manera no sabríamos si el cliente ha recibido al imagen que puede alojar nuestro servidor, o bién podríamos perder el estado de la conexión completa.

La programación del protocolo TCP en el proyecto se basa en la programación de sockets. Un socket es un punto final para la comunicación entre procesos. Los sockets son creados por el servidor y por los programas de aplicaciones cuando necesitan comunicarse con otros procesos. Cada socket tiene un nombre único, así que otros procesos pueden encontrarlo, conectarse y acceder a él. Un par de sockets conectados constituyen un canal de comunicación para que los procesos, que pueden no estar relacionados, puedan intercambiar datos. Un socket es una estructura de datos que mantiene el estado de la conexión y les da salida y entrada a los datos mientras uno

escribe o lee. Para nuestro caso, los programas en el servidor crearán sockets que servirán de punto de encuentro para los clientes.

Son 4 los pasos generales para utilizar los sockets en un programa:

1. Asignar e iniciar.
2. Conectar.
3. Transferir datos.
4. Cerrar.

5.3 HTTP (Capa de Aplicación).

Para el caso de la capa de aplicación se ha empleado el protocolo HTTP debido a que es el lenguaje que utilizan los navegadores web y servidores web para comunicarse entre si. A continuación se tratará de explicar conceptos generales de las partes que componen este lenguaje y el aspecto de una conversación HTTP habitual.

Se han definido especificaciones para las diferentes estructuras en la web, la especificación HTTP define la estructura adyacente sobre la que se asienta todo el tráfico web. Las especificaciones HTML y URL se presentan en otras estructuras. Para encontrar todas las especificaciones relacionadas con la Web se pueden obtener en la dirección http://www.w3.org/.

5.3.1 ENCABEZADOS HTTP.

Gran parte de la información que se intercambia entre cliente y servidor se presenta en forma de encabezados http, mismo que tiene la siguiente forma:
NombredelEncabezado: Data

Cuando un cliente se conecta con el servidor, envía varios encabezados HTTP por la línea, indicándole al servidor quien es y que es lo que quiere. El servidor envía una serie de encabezados de respuesta describiendo los datos que se devuelven o explicando la razón por la que no se devuelven. Aunque un usuario está más interesado en el cuerpo del mensaje, (la página Web u otro recurso), ésta es la parte menos interesante de la conversación HTTP.

La especificación HTTP define un gran número de encabezados que se pueden usar, pero esto no limita la posibilidad de crear encabezados propios para alguna aplicación específica. La siguiente tabla muestra algunos encabezados generales que puede utilizar tanto el servidor como el cliente.

Tabla 5.2. Algunos Encabezados HTTP y su significado.

Sintaxis del Encabezado	Significado
Cache-Control:directives	Existen diferentes directivas disponibles, dependiendo de si el encabezado lo envía el servidor o el cliente.
Connection:type	Especifica el tipo de conexión como Keep-Alive o Close.
Content-Language:language	Lo utiliza el servidor o el cliente par indicar en que idioma (humano) se encuentra el recurso. Son códigos estándar de dos letras que indican idiomas. Por ejemplo, el inglés está representado por en y el encabezado queda como: Content-Language:en
Content-Length:number_of_bytes	Cuando el cliente o el servidor envía los datos, el encabezado indica el tamaño en bytes de éstos.
Content-Location:URI	Proporciona un URI (Identificador Universal de recursos) donde el contenido está disponible si difiere del URI solicitado.
Content-MD5: MD5 digest	Contiene el resumen MD5 del cuerpo de solicitud o respuesta.
Content-Range: range/content_length	En una solicitud indica que sólo se requiere parte del contenido, en una respuesta indica que sólo se devuelve parte del contenido. Content-Range:0-300/2402.
Content-Type: type/subtype	Indica el tipo de MIME de los datos pasados en el cuerpo del mensaje. Content-type: text/html.
Date: date	La fecha y hora en que se produjo la transacción. Date: Thu, 23 Sep 1999, 22:58:27 EDT.
Expires: date	Indica cuando hay que considerar que los datos del cuerpo están caducos.
Last-Modified: date	Indica la última vez que se modificaron los datos del cuerpo.
Pragma: directive	Se puede usar para incluir directivas de implementación. Pragma: no-cache.
Transfer-Encoding:encoding_type	Indica que codificación tuvo lugar para transferir el mensaje por la conexión HTTP.
Upgrade:protocol/version	Permite que el remitente del mensaje sugiera al destinatario que la comunicación se realizarla mejor en otro protocolo. Esto permite que se inicie la comunicación en un protocolo más antiguo y permite que el cliente y el servidor negocien un nuevo protocolo. Upgrade: HTTP/2.0.
Via:server	En un mensaje se pueden poner uno o más encabezados Via para mostrar que éste llegó a su destino a través de uno o más servidores proxy. Via: 1.1 proxy.com (Apache 1.3.7).

5.3.2 LA CONVERSACIÓN HTTP.

Cada transacción http se maneja como conversación separada, sin memora de las conversaciones anteriores. Por ello se dice que http no tiene estado ya que no recuerda el estado que había al final de la última conversación. La conversación http consta varias partes que se manejarán de manera separada en este capítulo. La estructura de la conversación es como sigue:

1. **Solicitud del cliente.** El cliente (generalmente un navegador web) inicia la conversación conectándose con el servidor y solicitando un URI.
2. **Encabezados de solicitud.** Aparte de la solicitud , el cliente envía algunos encabezados adicionales.
3. **Cuerpo de solicitud.** El cuerpo de la solicitud puede contener datos adicionales.
4. **Estado del servidor.** Como primera parte de la respuesta, el servidor devuelve un código de estado, que indica si la solicitud ha tenido éxito y si no, lo que ha ido mal.
5. **Encabezados de respuesta.** El servidor puede devolver cualquier número de encabezados de respuesta.
6. **Datos solicitados.** Si la solicitud ha tenido éxito, los datos solicitados se devolverán al cliente.
7. **Desconectar.** La conversación ha terminado, por lo que el servidor se desconectará del cliente y esperará que se produzca otra solicitud. Una excepción posible a esto se da cuando Keep-Alive está activado, en cuyo caso la conexión se mantendrá abierta para la próxima solicitud del mismo cliente.

CAPÍTULO 6. EL KIT DE DESARROLLO TCP/IP.

En el presente proyecto se empleó el kit de desarrollo TCP/IP producido por las compañías: Rabbit Semiconductor y Z-World. Rabbit Semiconductor aporta una tarjeta con base en su microprocesador Rabbit y con capacidades de conexión a una red utilizando la pila TCP/IP. Z-World proporciona el software DynamicC, que es la interfaz entre el programador y la tarjeta. Ambas compañías aportaron sus conocimientos específicos para facilitar el desarrollo de aplicaciones y su pronta interconectividad con otras redes.

DynamicC permitió, entre otras cosas, desarrollar nuestro proyecto sin la necesidad de involucrarse a fondo en aspectos como los protocolos de la pila TCP/IP, o bien, en la comunicación entre la tarjeta empleada y la PC en donde se ejecuta la herramienta. DynamicC proporciona un entorno de programación de alto nivel que permite homogeneizar el código, sin necesidad de involucrarse en las características internas del procesador utilizado como puede ser, su lenguaje ensamblador o el manejo de sus registros internos. Cabe mencionar que esta generalización del código puede ser la característica más importante que se debe tomar en cuenta debido a la implementación en diferentes procesadores, permitiendo decidir utilizar el hardware con las mejores características de consumo, o de precio etc...

Rabbit por su parte, aporta una tarjeta funcional, competitiva, con componentes confiables y probados, para alojar y administrar nuestro proyecto una vez cargado en ella.

6.1 DYNAMIC C.

A Continuación hablaremos de los conceptos que atiende la paquetería incorporada en el kit con el nombre de DynamicC. Este software proporcionó la interfaz entre la tarjeta utilizada y el diseño de nuestro proyecto propuesto. DynamicC es una herramienta de desarrollo que funciona con cualquier sistema embebido basado en el microprocesador Rabbit. DynamicC incorpora las funciones de desarrollo de edición, compilación, enlace, carga y debug en un solo programa así como la programación en ensamblador. De hecho, el enlace, la compilación y la carga son todas una sola función. También DynamicC incorpora librerías de utilidad que facilitan la programación de sistemas completos. Este software compila directamente hacia la memoria. Las funciones y librerías son enlazadas, compiladas y cargadas en el momento. En una computadora rápida, puede cargar 30 kbytes de código en 5 segundos a una velocidad de 115,200 bps.

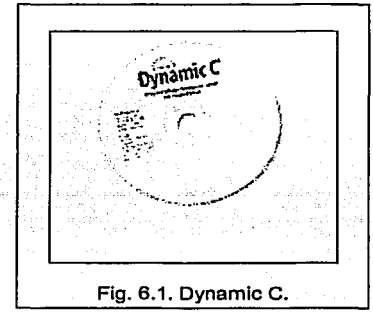


Fig. 6.1. Dynamic C.

DynamicC tiene un editor de texto fácil de usar y los programas pueden ejecutarse interactivamente en el código fuente o a nivel código de máquina. Además, DynamicC tiene menús desplegables y shortcuts de teclado para todos los comandos que facilitan su empleo. DynamicC se basa en el lenguaje C pero difieren por el hecho de que el segundo asume la presencia de un sistema operativo cargado en el sistema y el primero se enfoca en sistemas embebidos con batería de respaldo por si existiesen fallas en la fuente de poder.

DynamicC es una aplicación de 32 bits que interactúa con tarjetas basadas en el procesador Rabbit sin esperar que exista BIOS presente en la tarjeta cuando ésta se prende. DynamicC almacena el kernel BIOS en un archivo de C que es compilado y cargado cuando la tarjeta se inicializa. Lo anterior permite que se puedan utilizar memorias flash limpias sin la necesidad de precargar memorias en el sistema. Las actualizaciones del BIOS se pueden hacer directamente vía software y con la posibilidad de hacerlas disponibles para todos. Los diseños de tarjetas compatibles con DynamicC utilizando el procesador Rabbit se hacen simples, ya que existe una guía de diseño y utilizando un machote del BIOS que lo provee la compañía Z-World. Una característica más del software es su capacidad multitarea, independientemente de que el procesador sólo puede ejecutar una instrucción a la vez, el software se basa en los retardos naturales de cada tarea para incrementar la funcionalidad del sistema, por lo que aparenta un procesamiento en paralelo. También, gracias a la expansión del set de instrucciones del procesador Rabbit, se reduce el tamaño del código y se hace más veloz a su vez.

DynamicC también tiene el grado de inteligencia en su compilador y el la BIOS de la tarjeta capaz de almacenar variables de código o de datos en sus lugares respectivos de memoria. Cabe mencionar que gracias a las funciones implementadas en el software, el proyecto no tuvo necesidad de ser programado en ensamblador.

6.1.1 ARQUITECTURA CLIENTE SERVIDOR.

Para nuestro proyecto, DynamicC fue la pieza clave ya que es él el que se encarga de lidiar con los protocolos de la pila TCP/IP y a la vez, debido a que la arquitectura de nuestro proyecto es la conocida como cliente / servidor, le da a la tarjeta la capacidad de actuar como servidor web para atender requisiciones de varios clientes a la vez. Por lo anterior, es de relevante importancia entender el flujo de información que toma la comunicación para "servir" (entrega de un servicio) de manera general. Una aplicación cliente / servidor funciona repitiendo constantemente la siguiente rutina en bucle: petición del cliente, respuesta del servidor; petición del cliente, respuesta del servidor.

6.1.1.1 PETICIÓN DEL CLIENTE.

En un aspecto más específico y adaptándolo a nuestro proyecto, el cliente, desde su PC, abre un navegador web "explorador" (Netscape, Explorer, Mosaic, etc..), y escribe en la parte correspondiente, una dirección URL de una página en la red a la que se debe encontrar conectado (Internet, Intranet o bien la red de 2 hosts). La PC, es un dispositivo de multicapa y debido a esto, si ella cuenta con la pila TCP/IP en su sistema, empezará la labor de los protocolos TCP/IP y se colocan encabezados, se desciende a las capas inferiores, se vuelven a colocar encabezados y a descender hasta llegar a la capa física. Los bits viajarán a través de todo el entorno de colisión, podrán pasar a otros segmentos de red, y cada host en la subred destino, recibirá los bits emitidos por nuestra PC en su NIC, separará encabezados, interpretará datos para saber si es él el destinatario y, en función de esto, entregar los datos a la capa siguiente o desechar la información.

6.1.1.2 RESPUESTA DEL SERVIDOR.

Uno de los hosts a los que llegan los datos emitidos desde la PC del cliente es nuestro servidor (Tarjeta del kit de desarrollo empleada). Nuestro servidor también es un dispositivo multicapa. Vuelve a suceder el proceso de interpretación de encabezados según la capa de que se trate, y paso de información a la capa superior hasta que llegan los datos a la capa de aplicación. Nuestro servidor debe ser capaz de alcanzar el objetivo de interpretar los datos que envía el cliente y responderle adecuadamente. El servidor debe tener implementado en él los elementos necesarios para lograr la comunicación que es, a fin de cuentas, lo que el cliente pide. En otras palabras, el servidor debe contar con las herramientas necesarias para implementar la pila TCP/IP en sus líneas de código y, en nuestro proyecto, DynamicC fue el encargado de dicha tarea.

6.1.2 LIBRERIAS

DynamicC incorpora un conjunto completo de librerías, entre las que destacan:

- ARP.lib. Para que el proyecto responda a requisiciones ARP de la red.
- ICMP.lib . Para responder a requisiciones ICMP (pings).
- HTTP.lib. Para lograr atender requisiciones HTTP de clientes en toda la red.
- DCRTCP.lib. Para los protocolos TCP y UDP.

Las librerías DCRTCP.lib y HTTP.lib fueron las 2 librerías empleadas en el proyecto. DCRTCP.lib se encarga de llamar otras librerías de otros protocolos de capas inferiores cuando el programa lo requiera y HTTP.lib se encarga de la información en la capa de aplicación. La configuración del software compilado a memoria se determina por medio de macros introducidos en el programa así como con el uso de directivas de compilación.

CAPÍTULO 7. EL PROYECTO.

A continuación nos adentraremos en el proyecto propuesto dividiéndolo en 2 partes, Software y Hardware. La parte de software contiene una explicación detallada del código empleado así como los detalles que conciernen al software de proyecto, como el código HTML, programas CGIs etc. La parte de Hardware comprende las tarjetas, interfaces y dispositivos que se utilizaron en el proyecto incluyendo la tarjeta de Rabbit Semiconductor. Por último, se presentarán las limitaciones que tiene el proyecto con relación a Hardware y Software.

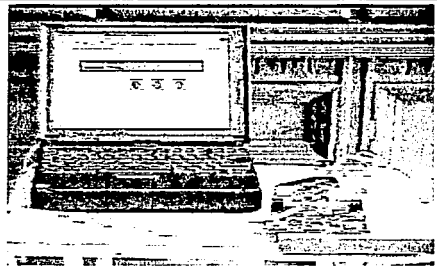


Fig. 7.1 Proyecto.

7.1 SOFTWARE

7.1.1 EL PROGRAMA.

El programa diseñado contiene una serie de puntos relevantes, mismos que fueron implementados en el siguiente orden:

1. Declaración de macros y directivas de carga.
2. Carga de archivos a la tarjeta con la directiva #ximport
3. Declaración de estructuras propias de las librerías empleadas.
4. Declaración de CGIs.
5. Declaración de subrutinas de adaptación al proyecto.
6. Programa principal.

7.1.1.1 DECLARACIÓN DE MACROS Y DIRECTIVAS DE COMPILACIÓN.

Las librerías utilizadas, DCRTCP.lib y HTTP.lib, necesitan ciertas directivas de configuración que nuestro programa debe pasarles para su correcto funcionamiento. Algunas de estas directivas entregan la configuración que tiene nuestra red implementada, otras proporcionan valores importantes para agilizar la transferencia de información, etc...

Las directivas empleadas en nuestro programa son las siguientes:

```
MY_GATEWAY. (x.x.x.x); /*Dirección IP de la puerta de enlace de nuestra red.*/
MY_IP_ADDRESS. (y.y.y.y); /*Dirección IP de la tarjeta empleada.*/
MY_NETMASK. (z.z.z.z); /*Máscara para la identificación de la red.*/
SOCK_BUF_SIZE (2048); /*Tamaño del buffer de los sockets.*/
HTTP_MAXSERVERS (4); /*Máximo número de servidores HTTP escuchando el puerto
80.  También hace referencia a la cantidad de entidades independientes en
nuestra página.*/
MAX_SOCKETS (4); /*Número de sockets empleados. Por lo regular tiene el mismo
valor que la macro HTTP_MAXSERVERS.*/
REDIRECTHOST  (MY_IP_ADDRESS); /*Dirección IP a utilizar en la función
cgi_redirectto(). Al llamar a esta función,  el usuario recibe la URL que se le
pasa como argumento.*/
REDIRECTTO (http://" REDIRECTHOST "/index.shtml); /*Archivo a cargar por lo
función cgi_redirectto().*/
RAPIDEZ (20000); /*Constante interna de nuestro diseño. Existe un contador que
tiene como límite esta macro, de manera que alterando su valor, se altera la
rapidez con la que se escribe a los puertos paralelos, mismos a los que se
encuentra conectada nuestra aplicación. */
```

Dentro de las directivas de compilación se emplearon las siguientes:

```
memmap xmem; /*Controla la memoria default para la carga de funciones, para
éste caso se refiere a la memoria extendida./*
use "dcrtcp.lib"; /*Activa la librería señalada (dcrtcp.lib ) para que sus
módulos puedan ser llamados desde el programa que la activa.*/
use "http.lib"; /*Igual que la directiva anterior pero con la librería
http.lib.*/
```

7.1.1.2 CARGA DE ARCHIVOS CON LA DIRECTIVA #ximport.

La directiva #ximport coloca el tamaño del archivo que se le pasa como primer argumento y su contenido binario, a partir de la siguiente localidad disponible en la memoria extendida flash. El segundo argumento declara una macro de compilación que señala la localidad física donde se almacenó la información del archivo.

A continuación se muestran las líneas que hacen referencia a la directiva ximport.

```
#ximport "samples/tcpip/http/pages/tesis1.shtml"    index_html;
#ximport "samples/tcpip/http/pages/rabbit1.gif"     rabbit1_gif;
#ximport "samples/tcpip/http/pages/button.gif"      button_gif;
```

Son 3 los archivos que se le mandan a la tarjeta, tesis.shtml, rabbit1.gif y button.gif y son parte del diseño de la página HTML de nuestro proyecto. Los archivos con la extensión gif son simplemente imágenes para darle apariencia al proyecto. El componente principal que se le manda a la tarjeta es la página en si (tesis1.shtml) . Esta página empleó el lenguaje de hipertexto HTML y su apariencia es la siguiente:

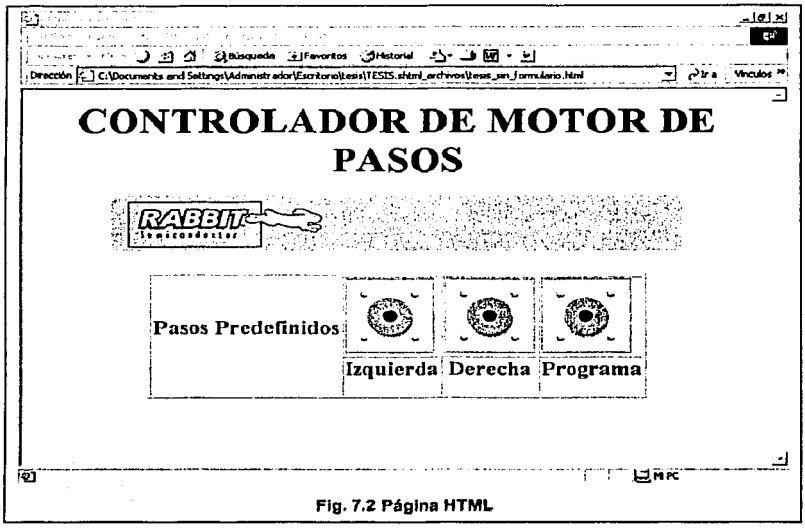


Fig. 7.2 Página HTML

7.1.1.2.1 PASOS PREDEFINIDOS.

El objetivo de la aplicación pasos definidos es el de girar nuestro motor una cantidad de pasos fija con un solo click. Los pasos predefinidos comprenden 3 imágenes del archivo button.gif que asemejan 3 push buttons, y cada una de ellas realiza 1 función específica dentro del proyecto.

La primera función denominada Izquierda hará que gire el motor una cantidad de pasos determinada en sentido antihorario. De manera análoga sucede con la segunda

función llamada derecha, sólo que el giro será en sentido horario. La tercera función con el nombre de Programa, hace que nuestro motor gire algunos pasos en sentido horario, otros en sentido antihorario, para remarcar el hecho de que es posible programar giros en cualquier sentido dentro del proyecto.

En un panorama más a detalle, el accionar los hipervínculos a los que hace referencia cada función (imágenes button.gif), le pide al controlador http_handler que mande una requisición http a nuestro servidor (tarjeta) para que éste le responda. Cada una de nuestras funciones demanda la ejecución de programas cgis a nuestro servidor. La siguiente tabla muestra la tarea y su cgi requerido.

Tabla 7.1. Tarea vs. CGI	
Izquierda	izquierda.cgi
Derecha	derecha.cgi
Programa	programa.cgi

A continuación se muestra el código del archivo tesis1.html .

```
<html>
<head><title> TESIS </title></head>
<body>
<center> <h1> CONTROLADOR DE MOTOR DE PASOS </h1>
<img SRC="rabbit1.gif"><p>
<table border>
<tr><td align=center valign=middle rowspan=3><h3>Pasos
Predefinidos</h3></td></tr>
<tr><td><A HREF="/izquierda.cgi"> <img SRC="button.gif"></A></td>
<td><A HREF="/derecha.cgi"> <img SRC="button.gif"></A></td>
<td><A HREF="/programa.cgi"> <img SRC="button.gif"></A></td></tr>
<tr><td align=center valign=middle><h3>Izquierda</h3></td>
<td align=center valign=middle><h3>Derecha</h3></td>
<td align=center valign=middle><h3>Programa</h3></td></tr>
</table>
</center>
</body>
</html>
```

7.1.1.3 DECLARACIÓN DE ESTRUCTURAS.

A continuación, se hará referencia a las estructuras a declarar, que resultan necesarias para la configuración correcta de nuestro servidor Web. Resulta ser un paso necesario del programa, tal vez protocolario, declarar estas estructuras de datos. Se trata de 2 estructuras, la primera con el nombre HttpSpec y la segunda HttpType.

HttpSpec es la estructura de datos cerebro de nuestro servidor Web. Contiene todos los archivos, variables y funciones a los que nuestro servidor tiene acceso. Se puede decir que es la interfaz entre el código HTML de nuestra página y el programa en si. El código HTML refiere hipervínculos que se muestran en el segundo campo de esta estructura. El programa refiere subrutinas, macros, directivas como #ximport o bien subprogramas contenidos en el código fuente aquí presentado.

Para nuestro proyecto, la siguiente es la estructura HttpSpec empleada:

```
HttpSpec http_flashspec[] =
{
   { HTTPSPEC_FILE, "/", index_html,     NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE, "/index.shtml",  index_html, NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE, "/rabbit1.gif",  rabbit1_gif, NULL, 0, NULL, NULL},
   { HTTPSPEC_FILE, "/button.gif",  button_gif,     NULL, 0, NULL, NULL},
   { HTTPSPEC_FUNCTION, "/derecha.cgi",      0, derecha,    0, NULL, NULL},
   { HTTPSPEC_FUNCTION, "/izquierda.cgi",    0, izquierda, 0, NULL, NULL},
   { HTTPSPEC_FUNCTION, "/programa.cgi",     0, programa,  0, NULL, NULL},

};
```

HttpType es la estructura de datos encargada de asociar la extensión el archivo con el tipo MIME o bien con el archivo que se encargará del tipo MIME de nuestro programa.

Para nuestro proyecto, la estructura HttpType empleada fue la siguiente:

```
HttpType http_types[] =
{
   { ".shtml", "text/html", shtml_handler}, // ssi
   { ".html", "text/html", NULL},           // html
   { ".cgi", "", NULL},                     // cgi
   { ".gif", "image/gif", NULL}             // gifs
};
```

7.1.1.4 DECLARACIÓN DE CGIs.

En esta parte del programa se dan de alta los cgis desarrollados para que se cumpla el objetivo del proyecto en general. Gracias a la relación que se declara en las estructuras HttpType y HttpSpec, el servidor Web sabrá que hacer cuando ocurra una requisición de un archivo con extensión cgi. Si se trata de una requisición del archivo derecha.cgi, en el programa se mandará llamar la función derecha y de manera similar con los archivos izquierda.cgi y programa.cgi.

La programación de los cgis está basada plenamente en la programación de alto nivel del Lenguaje C. Haremos hincapié a las funciones a las que hacen referencia cada cgi posteriormente, en la declaración de subrutinas de adaptación al proyecto, estas son: derecha1(), reseteo() e izquierda1(). Pero existe una función denominada cgi_redirectto(), que se encarga de traer la página que se le envía como argumento una vez que haya terminado de ejecutarse el cgi.

A grandes rasgos, los 3 cgis necesitan una variable entera denominada acumulador, misma que es iniciada con un valor, posteriormente entra a un ciclo donde es decrementada 4 unidades, se manda llamar alguna subrutina (derecha1 o izquierda1), se limpian los puertos de salida con la función reseteo() y se vuelve a comparar nuevamente para saber si vuelve o no a entrar al ciclo.

A continuación se muestra el fragmento de código de los 3 cgis implementados.

```
int acumulador;
int derecha(HttpState* state)
{
   acumulador = 30;
   reseteo();
   while(acumulador>0)
   {
      acumulador=acumulador-4;
      derecha1();
      reseteo();
   }
   cgi_redirectto(state,REDIRECTTO);
   return 0;
}

int izquierda(HttpState* state)
{
   acumulador = 30;
   reseteo();
   while(acumulador>0)
   {
```

```
        acumulador=acumulador-4;
        izquierda1();
         reseteo();
    }
    cgi_redirectto(state,REDIRECTTO);
    return 0;
}

int programa(HttpState* state)
{
        reseteo();
        acumulador = 30;
        reseteo();
        while(acumulador>0){ acumulador=acumulador-4;izquierda1();reseteo();}
        acumulador = 30;
        reseteo();
        while(acumulador>0){ acumulador=acumulador-4;derechal();reseteo();}
        acumulador = 30;
        reseteo();
        while(acumulador>0){ acumulador=acumulador-4;izquierda1();reseteo();}
        acumulador = 30;
        reseteo();
        while(acumulador>0){ acumulador=acumulador-4;izquierda1();reseteo();}
        cgi_redirectto(state,REDIRECTTO);
        return 0;
}
```

7.1.1.5 DECLARACIÓN DE SUBRUTINAS (Adaptación al proyecto).

Las rutinas de adaptación al proyectos son las encargadas de implementar la programación de puertos del procesador de nuestra tarjeta empleada. Esta es la parte que se comunica con nuestro hardware propuesto, el motor de pasos.

Los pines a los que tenemos acceso en la tarjeta están conectados a los puertos paralelos B y D. El puerto B, si se utiliza como puerto paralelo, tiene configuradas 6 entradas y 2 salidas (pines 6 y 7). Con la tarjeta empleada tenemos acceso a los 2 pines de salida del puerto B. Cuando deseamos escribir algo al puerto B, lo escribimos hacia el registro PBDR.

Para el caso del puerto paralelo D, cada pin puede ser configurado como entrada o salida, y es el registro PDDDR el que le indica al procesador cómo interpretar cada pin. Un alto hace que el pin correspondiente se interprete como salida "sólo escritura". También para el puerto paralelo D existe otro registro PDDCR que le indica la procesador si la salida es open-drain (colector abierto) si tiene un alto escrito en el pin correspondiente. Cuando deseamos escribir algo en el puerto D, lo escribimos en el

registro PDDR. Para nuestro proyecto utilizaremos los pines 0 y 1 del puerto D como salidas por lo que habrá que ponerlos en alto en el registro PDDDR.

Creamos un ciclo hasta que se cumpla la macro que definimos en el paso declaración de macros y directivas de compilación, para esperar un pequeño tiempo para seguir con el programa. Cambiando esta macro, podremos hacer que el motor de pasos conectado a los puertos paralelos de nuestro procesador avance más rápido, o más despacio.

A grandes rasgos, estas subrutinas sólo escriben un alto en una de las bobinas de nuestro motor mientras las otras permanecen en bajo, para moverlo un paso.

Es de importancia hacer notar la rutina con el nombre reseteo. Esta rutina sólo escribe altos en todos los puertos a los que está directamente conectado nuestro motor de pasos, de manera que impide que circule corriente a través de alguna de sus bobinas. Al resetear la tarjeta para cargarle un nuevo programa, ésta por default escribe bajos en 2 de los 4 pines de salida a los que se encuentra conectada la bobina, y no es sino hasta que se carga el programa cuando se logra escribir en dichos pines.

A continuación se muestran las rutinas de adaptación al proyecto empleadas:

```
derecha1()
{
   int j;
   WrPortI(PDDDR, &PDDDRShadow, 0x03);
   WrPortI(PDDCR, &PDDCRShadow, 0x00);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 7);
   for(j=0; j<RAPIDEZ; j++);
}
```

```
izquierda1()
{
   int j;
   WrPortI(PDDDR, &PDDDRShadow, 0x03);
   WrPortI(PDDCR, &PDDCRShadow, 0x00);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0x00, 7);
   for(j=0; j<RAPIDEZ; j++);
}

reseteo()
{
   WrPortI(PDDDR, &PDDDRShadow, 0x03);
   WrPortI(PDDCR, &PDDCRShadow, 0x00);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 0);
   BitWrPortI(PDDR, &PDDRShadow, 0xff, 1);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 6);
   BitWrPortI(PBDR, &PBDRShadow, 0xff, 7);
}
```

7.1.1.6 PROGRAMA PRINCIPAL.

Por último se menciona el programa principal. Este programa incorpora las siguientes funciones en sus líneas de código: reseteo(), sock_init(), http_init(), y http_handler. La función de reseteo se acaba de mencionar en la parte de rutinas de adaptación al proyecto. El resto de las subrutinas se señalan a continuación.

Sock_init() inicializa la librería DCRTCP.LIB que se encarga de lidiar con los protocolos de la pila TCP/IP. También sock_init() declara los valores iniciales de las estructuras de datos del sistema e inicializa el controlador de paquetes TCP. La función http_init() inicializa el servidor web y http_handler() es la función básica de control del servidor http, parsea la información recibida y manda llamar otros controladores declarados en la estructura http_type como html_handler.

A continuación se muestran las líneas de código del la rutina principal.

```
main()
{
   reseteo();
   sock_init();
   http_init();
   while (1)
   {
      http_handler();
   }
}
```

7.2 HARDWARE.

El principal objetivo de el presente trabajo es el de llegar a controlar hardware a través de una requisición HTTP. Es por esto que el presente capítulo abarca todo lo relacionado al hardware del proyecto. El tema lo dividimos en los siguientes puntos de relevancia que tienen relación directa con el proyecto.

- TARJETA.
- MOTOR DE PASOS.
- INTERFAZ

7.2.1 TARJETA.

El kit de desarrollo TCP/IP integra una tarjeta de desarrollo funcional cuyos diagramas eléctricos se encuentran en el anexo B del presente trabajo. De la tarjeta en cuestión destacan los siguientes componentes y funciones de importancia:

- PROCESADOR
- RELOJ DEL SISTEMA Y CONSUMO.
- PUERTO DE PROGRAMACIÓN.
- MANEJO DE MEMORIA.
- ENTRADAS Y SALIDAS PARALELAS.

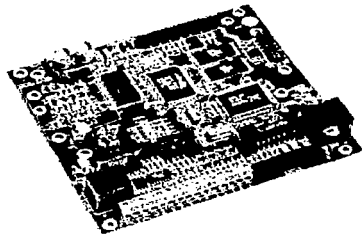


Fig. 7.3 Tarjeta

7.2.1.1 PROCESADOR.

A base de los microprocesadores Z80, Z180 y HD64180, Rabbit Semiconductor crea su microprocesador Rabbit 2000 el cual comparte una arquitectura similar a la de sus antecesores con algunas mejoras. El procesador Rabbit 2000 fue diseñado en conjunta cooperación con la compañía Z-World, encargada del desarrollo de software y todo esto tomando como base el lenguaje estructurado C. Por lo anterior, se puede tener un sistema de desarrollo de aplicaciones con el software desarrollado por Z-World cargado en una PC, nuestra tarjeta de desarrollo y un cable de conexión entre ambos.

Fig. 7.4. Procesador

7.2.1.1.1 ASPECTOS GENERALES:

- Paquete 100 pin PQFP. Voltaje de operación entre 2.5 y 5 Volts. Velocidad de reloj 30 Mhz. Costo menor a $10 USD.
- Arquitectura sin pegamento.
- Reinicio en frío. (Se puede soldar directamente la memoria flash virgen).
- Especificaciones industriales. Voltaje de operación +/- 10% y temperatura entre – 45°C y 85°C. Especificaciones comerciales. Voltaje de operación +/- 5% y temperatura entre 0°C y 70°C.
- Permite hasta 50,000 líneas de código por su espacio de 1 Mbyte.
- 4 niveles de prioridad de interrupciones.
- 40 líneas de entrada y salida (compartidas con puertos seriales).
- 4 puertos seriales.
- Puerto esclavo.
- Relojes y Contadores, 6 en total.
- Un puerto standard de 10 pines para programación del sistema.

7.2.1.2 RELOJ DEL SISTEMA.

El oscilador principal se basa en un cristal externo cuya frecuencia se encuentra entre 1.8 y 2.9Mhz. El reloj del procesador se deriva del oscilador principal ya sea duplicando su frecuencia o bien dividiéndola entre 8. Este reloj también puede ser implementado externamente por un oscilador a una frecuencia de 32.768 kHz para bajo consumo, en cuyo caso el oscilador principal será deshabilitado vía software.

Freq Reloj (MHz)	Voltaje (V)	Corriente (ma)	Consumo (mW)	Freq Reloj (Mhz)	Voltaje (V)	Corriente (ma)	Consumo (mW)
25	5	80	400	6	2.5	10	25
12.5	5	40	200	3	2.5	5	12
12.5	3.3	26	87	1.5	2.5	2.5	6
6	3.3	13	42	0.032	2.5	0.054	0.135

Tabla 7.2. Consumos aproximados según frecuencia de reloj.

7.2.1.3 PUERTO DE PROGRAMACIÓN.

La tarjeta de desarrollo tiene un puerto de programación que emplea el puerto serial A del procesador rabbit para la comunicación. Este puerto de programación provee la interfaz física y eléctrica entre la tarjeta de desarrollo y la plataforma de DynamicC.

Entre sus características principales destacan:

- Configuración para arranque externo o auto-arranque.
- Reseteo externo.
- Línea de reloj integrada para una rápida comunicación serial.
- Configuración en campo y obtención de diagnósticos en tiempo real.

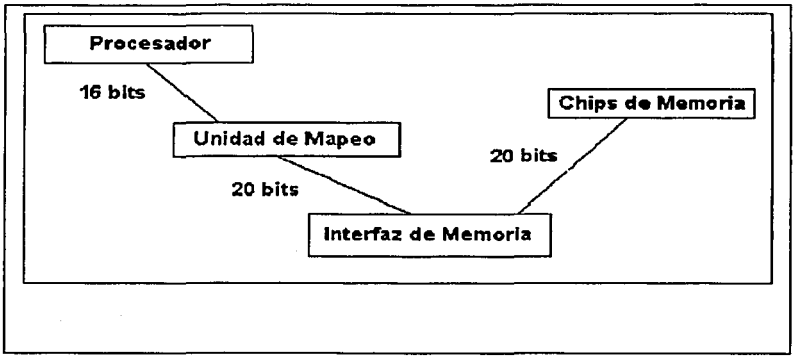
Enfocándolo en nuestro proyecto, el propósito del puerto de programación es el de escribir información directamente a la memoria flash, para posteriormente reiniciar la tarjeta y ejecutar el arranque en frío por medio de un programa dedicado a dicha funcionalidad.

7.2.1.4 MANEJO DE MEMORIA.

La configuración básica para el procesador Rabbit incluye una memoria Flash y otra memoria RAM. Ambos chips se encuentran incluidos en la tarjeta. Aunque la el procesador Rabbit puede manejar hasta 1 Megabyte de código, es sabido que las aplicaciones en sistemas embebidos utilizan 250 Kbytes, equivalentes a 10,000 – 20,000 declaraciones en C. Las variables de acceso directo en C se limitan a 44 K divididas entre la Flash y la Ram, más que suficiente en aplicaciones de sistemas embebidos. Algunas otras aplicaciones requieren almacenar tablas o arreglos de datos con mucha información, para estas aplicaciones, DynamicC incorpora un tipo de memoria extendida, incluso mayor a un MegaByte.

Para el caso de la memoria de stack, su incremento viene relacionado con la opción multitarea empleada, para el caso del proyecto, no tenemos contemplada dicha opción, por lo que este tipo de memoria se incorpora en el segmento de datos.

Las instrucciones del procesador son de 16 bits (64 Kbytes), lo que crea un conflicto cuando se requiere tener aplicaciones con mucho código (1 MegaByte). Para esto se incorpora una unidad de mapeo de memoria que tiene como entrada 16 Bits y como salida 20. A continuación se muestra como se relacionan los diferentes elementos de memoria en el sistema.



Para poder mapear 16 bits de Instrucciones en 20 bits de acceso a memoria, la unidad de mapeo de memoria incorpora 4 registros (SEGSIZE, STACKSEG, DATASEG y XPC) que dividen y mantienen las secciones lógicas de memoria y mapean cada una de ellas en memoria física. Estos registros son de 8 bits y son sumados a los 4 MSB de los 16 bits de entrada para obtener los 20 bits de salida. Entonces cada segmento de memoria es mapeado a una ventana de memoria física. El tamaño de las ventanas viene determinado por el registro SEGSIZE, que es de 8 bits, 4 de ellos para determinar el tamaño entre las 2 primeras ventanas y los 4 restantes para las 2 últimas.

Los nombres atribuidos a cada segmento evocan el uso común de cada uno de ellos. El segmento raíz se mapea a la base de la memoria flash y contiene código de inicio y otros códigos que pueden almacenarse ahí. El segmento de datos varía en función a la estrategia establecida para la memoria. Puede ser una extensión de el código raíz o bien puede contener variables de datos. El segmento de stack tiene

normalmente 4K de espacio y alberga el stack del sistema. Por último el segmento XPC que se utiliza normalmente para ejecutar código que no se encuentra almacenado en ninguno de los segmentos de raíz o de datos.

Si la BIOS arranca desde la flash, su código comienza en la sección de código de raíz en la dirección 0x00000 y se va llenando hacia arriba. El resto del código raíz continuará colocándose siguiendo el código BIOS. Si la BIOS arranca desde la memoria RAM, el código de raíz junto con los datos de raíz y la sección de stack serán colocados a partir de la dirección 0x80000.

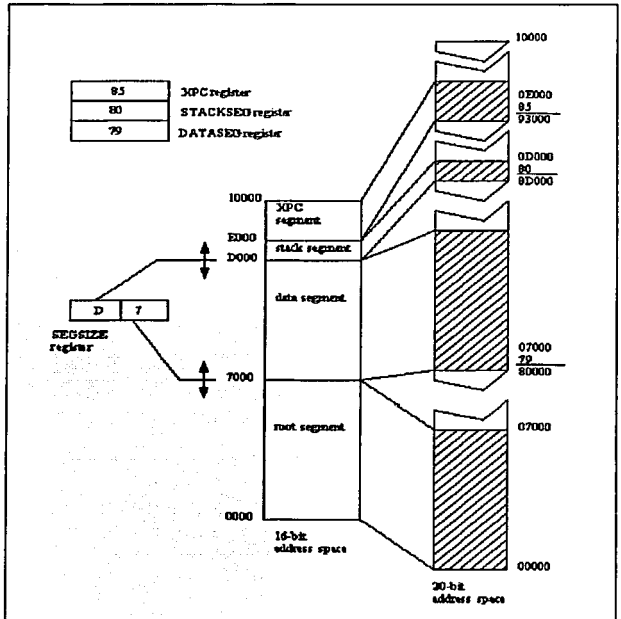


Fig. 7.6. Memoria.

7.2.1.5 ENTRADAS Y SALIDAS PARALELAS.

Hay 40 líneas de entrada y salida distribuidas en 5 puertos de 8 bits denominados con las letras de la A a la E. Muchas de estas líneas tienen tareas a parte como la de puerto serial. Los puertos D y E tienen la capacidad de ser puertos de salida sincronizados. Los registros de salida están configurados en cascada. Datos al puerto son almacenados en primera instancia por un registro que al recibir la señal adecuada transfiere el dato al segundo registro y a la vez puede interrumpir para activar una nueva carga al registro del primer nivel recién vaciado. Con este tipo de configuración, es posible tener salidas precisas en determinados lapsos de tiempo para aplicaciones como señalización en comunicaciones, controladores de modulación por ancho de pulsos o bien controladores de motores de pasos.

7.2.1.5.1 PUERTO D.

Dos de los pines del puerto paralelo D fueron empleados como salidas por nuestro proyecto debido a que la tarjeta de desarrollo venía incorporada de esa manera. Entre las características principales de dicho puerto destacan:

- Sus 8 pines pueden ser programados como entradas o salidas individualmente.
- Las salidas pueden configurarse como salidas estandar o bien como salidas a colector abierto.
- Los registros de salida están en cascada y controlados por tiempo, haciendo posible que se obtengan pulsos precisos.

Los registros de configuración y su función empleados en nuestro proyecto y relacionados con el puerto D se muestran a continuación:

PDDR. Registro de datos del puerto D (Lectura/Escritura).
PDDDR. Registro de dirección de datos del puerto D. Un 1 en determinado pin lo convierte en salida.

PDDCR. Registro de configuración de salidas del puerto D. Un 1 en determinado pin lo convierte en salida a colector abierto de estar éste configurado como salida en el registro PDDDR.

En lo relacionado a la corriente que puede entregar cada pin configurado como salida en el puerto D, la cantidad máxima es de 25mA con plena Carga AC, lo que significa a 22.11 Mhz en su reloj y cargas capacitivas menores a 100pf por pin.

7.2.1.5.2 PUERTO B.

Este puerto tiene 6 entradas y 2 salidas cuando se encuentra configurado como puerto paralelo. Dos de las 4 salidas preestablecidas en la tarjeta de desarrollo apuntan hacia los pines 6 y 7 de este puerto. Estas salidas fueron empleadas por nuestro proyecto por lo que la corriente máxima que puede entregar cada pin es de 8 mA el pin 6 y 12 mA el pin 7 con plena Carga AC, lo que significa a 22.11 Mhz en su reloj y cargas capacitivas menores a 100pf por pin.

7.2.2 MOTOR DE PASOS.

El motor de pasos que se empleó en el proyecto es un motor de pasos unipolar cuyo diagrama eléctrico se muestra a continuación:

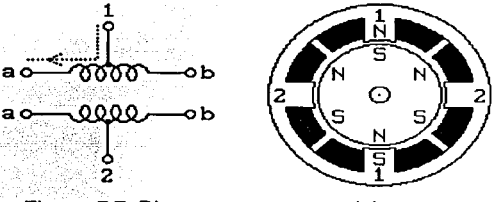


Figura. 7.7. Diagrama y esquema del motor.

Se puede notar que cuenta con un tab central en cada uno de sus 2 embobinados. Típicamente, este tab central se encuentra alambrado a la fuente de poder y los 2 extremos de cada embobinado se conectan alternativamente a tierra para controlar la dirección del campo que provee el embobinado. El embobinado número 1 se distribuye entre la parte de arriba y abajo del estator, y el embobinado número 2 se distribuye entre la izquierda y derecha del mismo. El rotor es un imán permanente con 6 polos, 3 sures y 3 nortes, arreglados a lo largo de su circunferencia. Para lograr una mayor resolución angular, el rotor debe proporcionar más polos. La corriente desde el tab central del embobinado 1 a la terminal a, produce en la parte superior del estator un norte y su parte inferior un sur. Esto provoca que el rotor se posicione como se muestra en la figura. Si se retira el poder del embobinado 1 y se energiza el embobinado 2, el rotor girará un paso. Para que rote el motor continuamente, debemos energizar las bobinas en secuencia.

Asumiendo una lógica positiva, donde un 1 significa energizar el embobinado del motor, las siguientes secuencias girarán el motor 24 pasos.

```
Embobinado 1a   10001000100010001000100010001
Embobinado 1b   00100010001000100010001000100
Embobinado 2a   01000100010001000100010001000
Embobinado 2b   00010001000100010001000100010
              Tiempo  --->

Embobinado 1a 11001100110011001100110011001
Embobinado 1b 00110011001100110011001100110
Embobinado 2a 01100110011001100110011001100
Embobinado 2b 10011001100110011001100110011
              time --->
```

Note que las 2 mitades de cada embobinado nunca se energizan al mismo tiempo.

Ambas secuencias girarán el imán permanente un paso a la vez. La primera secuencia energiza un embobinado a la vez, por lo que emplea menos potencia. La segunda secuencia energiza 2 embobinados a la vez, lo que produce un torque 1.4 veces mayor, y requiere mayor potencia.

Para lograr mayor resolución en cada paso, se pueden combinar las 2 secuencias mostradas y lograr así medios pasos. La secuencia resultado sería:

```
Embobinado 1a 11000001110000011100000111
Embobinado 1b 00011100000111000001110000
Embobinado 2a 01110000011100000111000001
```

```
Embobinado 2b 00000111000001110000011100
           tiempo --->
```

7.2.3 INTERFAZ.

Para realizar la conexión física entre el kit de desarrollo TCP/IP y el motor de pasos, se utilizaron 4 de los 7 controladores que tiene el chip ULN2003. Este chip está diseñado para servir de interfaz entre circuitos lógicos de bajo nivel y múltiples periféricos de gran carga (500 ma x 50V por controlador). Entre estos periféricos destacan relevadores, solenoides, calentadores, displays incandescentes y motores de pasos. Los 7 controladores del chip vienen integrados con salidas a colector abierto y diodos de volanteo. Las especificaciones eléctricas del chip se encuentran el anexo C del presente trabajo.

Algunas de las razones por las que se decidió a implementar el chip se presentan a continuación:

- Encendido simultaneo de los 7 controladores.
- Entradas compatibles: TTL, DTL, PMOS y CMOS.
- Corriente de salida hasta 500 ma.
- Voltaje de Salida hasta 95 V.
- Protección a transitorios.
- Fácil integración física la proyecto por su empaque estandarizado.
- Integración de resistencias en serie en cada entrada así como la integración de diodos de volanteo (reducción de espacio y elementos).

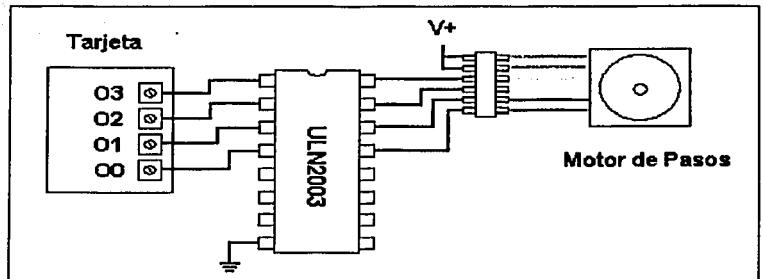


Fig. 7.8 Esquema del Proyecto.

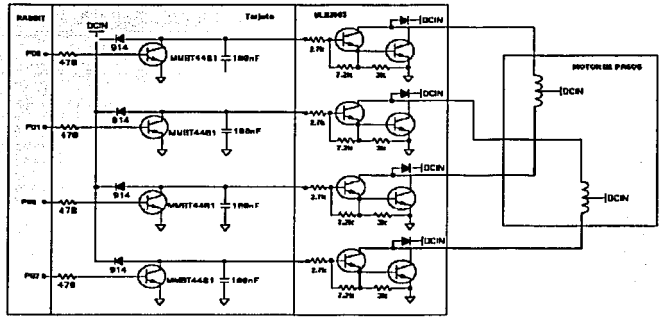


Fig. 7.9. Diagrama eléctrico.

7.3 LIMITACIONES DEL PROYECTO.

A continuación nos enfocaremos en el proyecto como una aplicación funcional y no experimental como estuvo contemplado a lo largo de su desarrollo.
Dentro de las limitaciones del proyecto en el entorno Web podemos resaltar:

- Control de Usuarios: Si 2 usuarios deseasen controlar el motor de pasos al mismo tiempo no obtendrán los resultados que cada uno de ellos espera, sino una combinación de ambas requisiciones.
- Restricción de accesos: Cualquier persona conectada a la misma red que nuestro proyecto puede controlar la aplicación.
- No hay realimentación. No es posible para un usuario saber si el motor atendió su requisición. Tal vez con la implementación de una cámara web acoplada a nuestro motor o bien la incorporación de sensores como entradas al procesador se pueda monitorear la respuesta y presentársela al usuario.

A nivel de hardware las limitaciones del proyecto son:

- Acceso a los pines del procesador. La tarjeta de desarrollo utilizada tiene únicamente algunas entradas y salidas paralelas accesibles, independientemente de que el procesador contempla muchas mas. Si bien el acceso a pines seriales está disponible, la incorporación de otros dispositivos que funcionen como interfaz de proyectos con mayores elementos a controlar se vuelve inevitable.
- Valor predeterminado en los registros de configuración. Al iniciar la tarjeta, el procesador configura sus registros por default a determinados valores, mismos que pueden causar conflicto con el hardware del proyecto. Por ejemplo, si un registro de salida tiene configurado por default un bajo, cuando nuestra aplicación espera un alto, es posible que el hardware sufra desgastes.
- Hardware predeterminado en la misma tarjeta. Referido a que se confía en que Rabbit Semiconductor realizó pruebas de consumo, diseño y funcionalidad a cada elemento incorporado en su tarjeta.

A nivel software podemos resaltar las siguientes limitaciones:

- Independencia de la aplicación. Debido a que no fue alterada la boot del proyecto, el programa necesita ser cargado vía PC cada que se desee tener el proyecto funcionando.
- Optimización del código. Debido a que el proyecto estaba enfocado a una prueba experimental no se analizó el caso de mejor rendimiento del programa o la optimización de los recursos del proyecto.

CONCLUSIONES.

En el trabajo aquí presentado se logró controlar el movimiento de un motor de pasos empleando la tarjeta desarrollada por la compañía Rabbit Semiconductor, misma que, por incorporar en su diseño la pila TCP/IP, permitió que dicho control se haya realizado en red. En un aspecto más general, el proyecto sólo es una pequeña aplicación de las que se pueden hacer cuando se conjunta la pila TCP/IP y el diseño electrónico.

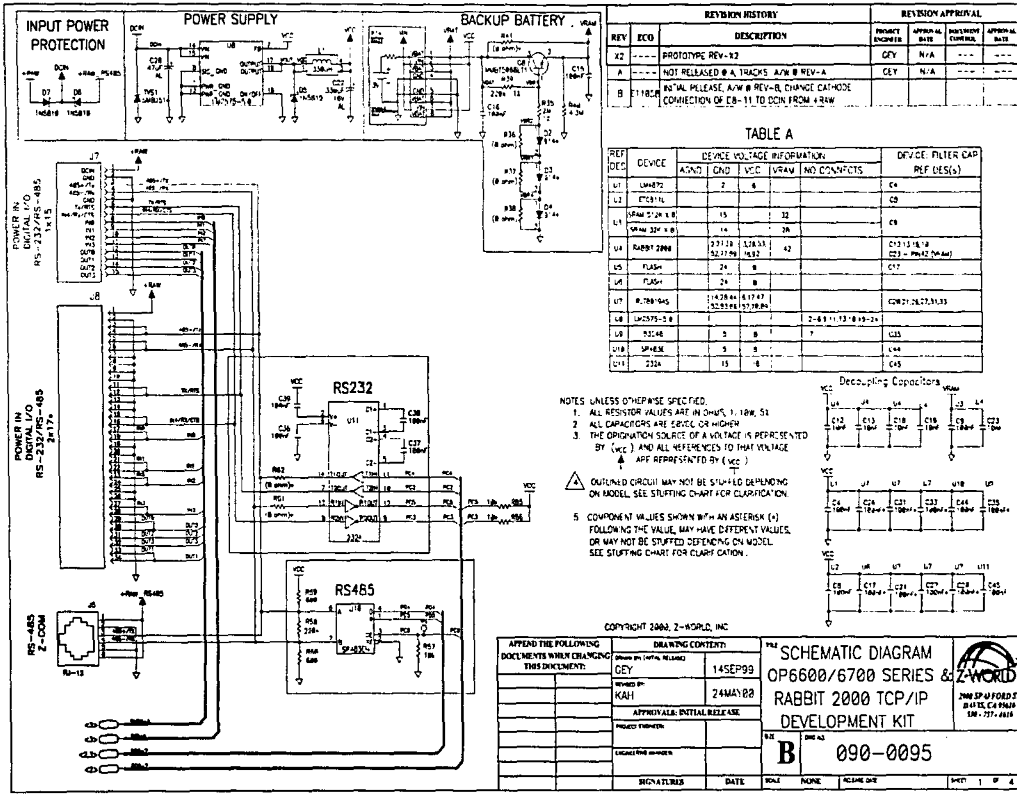
Internet y el conjunto de protocolos TCP/IP, se han convertido en un estándar de comunicación de datos de la industria y el mercado teleinformático. Se puede prever que las redes de comunicaciones en general, diferidas entre sí por el tipo de servicio (televisión, telefonía o datos), convergen en una red de paquetes de alta velocidad capaz de soportar: voz, datos, video, multimedia y una gran diversidad de aplicaciones en tiempo real. A su vez, la inter-conectividad entre todas estas redes no se limita a PCs y estaciones de trabajo, también se pueden incluir teléfonos celulares, PDAs (Personal Digital Assistant), electrodomésticos, máqinas de venta y de equipo industrial.

Internet sigue creciendo y las oportunidades de desarrollo para Ingenieros en general en este campo son bastas, fomentando el diseño y la creatividad de sus proyectos. Día tras día se alcanzan más metas tecnológicas en mejora de equipo y eficiencia de programas tanto en tiempo como en recursos, por lo que, se hacen accesibles y baratas las aplicaciones de gran envergadura; con acceso a bases de datos, programación dinámica, disposición inmediata de respuestas, etc.. Si a estas aplicaciones les agregamos el elemento "donde sea" aunado al avance tecnológico en comunicaciones, si se deseara controlar un sistema embebido desde cualquier pare del mundo, bastará con que el servicio incorpore el elemento "ESTAR EN LINEA" en su código.

ANEXO A. DIAGRAMAS ELÉCTRICOS DE LA TARJETA.

TESIS CON
FALLA DE ORIGEN

89



REVISION HISTORY			REVISION APPROVAL			
REV	ECO	DESCRIPTION	PROJECT ENGINEER	APPROVAL DATE	DOCUMENT CONTROL	APPROVAL DATE
X2	----	PROTOTYPE REV-X2	GEY	N/A	---	---
A	----	NOT RELEASED @ A, TRACKS A/W @ REV-A	GEY	N/A	---	---
B	E1185B	INITIAL RELEASE, A/W @ REV-B, CHANGE CATHODE CONNECTION OF D8-11 TO DCIN FROM +RAW				

TABLE A

REF DES	DEVICE	DEVICE VOLTAGE INFORMATION					DEVICE FILTER CAP REF DES(s)
		AGND	GND	VCC	VRAM	NO CONNECTS	
U1	LM4872		2	6			C4
U2	ETC811L						C9
U3	SRAM 512K x 8		15		32		C8
	SRAM 32K x 8		14		28		
U4	RABBIT 2000		2,27,38, 52,77,99	3,28,53, 78,92	42		C12,13,18,19 C23 - PN42 (VRAM)
U5	FLASH		24	8			C17
U6	FLASH		24	8			
U7	RJ7801945		14,28,44, 52,93,66	8,17,47, 57,70,84			C20,21,26,27,31,33
U8	LM2575-5.0					2-6,9,11,13,18,19-24	
U9	93C46		5	8		7	C35
U10	SP483E		5	8			C44
U11	232A		15	16			C45

NOTES UNLESS OTHERWISE SPECIFIED.

1. ALL RESISTOR VALUES ARE IN OHMS, 1/10W, 5%
2. ALL CAPACITORS ARE 50VDC OR HIGHER
3. THE ORIGINATION SOURCE OF A VOLTAGE IS REPRESENTED BY (VCC), AND ALL REFERENCES TO THAT VOLTAGE ARE REPRESENTED BY (VCC)
4. OUTLINED CIRCUIT MAY NOT BE STUFFED DEPENDING ON MODEL, SEE STUFFING CHART FOR CLARIFICATION.
5. COMPONENT VALUES SHOWN WITH AN ASTERISK (*) FOLLOWING THE VALUE, MAY HAVE DIFFERENT VALUES, OR MAY NOT BE STUFFED DEPENDING ON MODEL. SEE STUFFING CHART FOR CLARIFICATION.

APPEND THE FOLLOWING DOCUMENTS WHEN CHANGING THIS DOCUMENT:

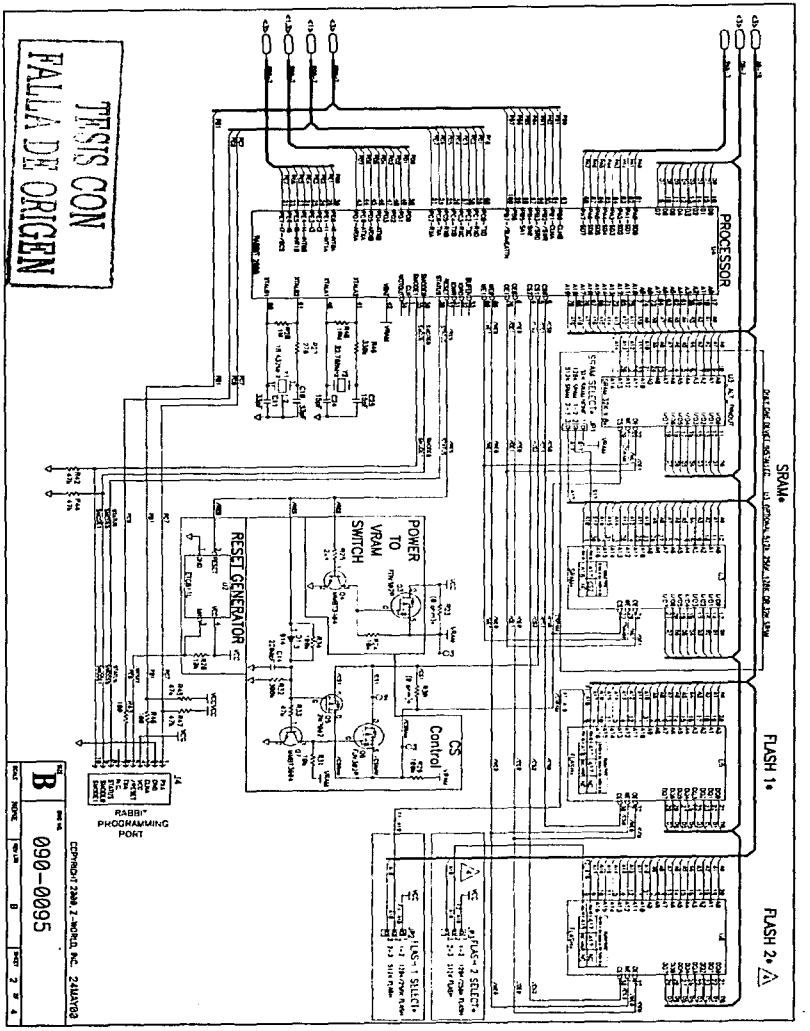
DRAWING CONTENT	
GEY	14SEP99
REVISED BY: KAH	24MAY00
APPROVALS: INITIAL RELEASE	
PROJECT ENGINEER	
ENGINEERING MANAGER	
SIGNATURES	DATE

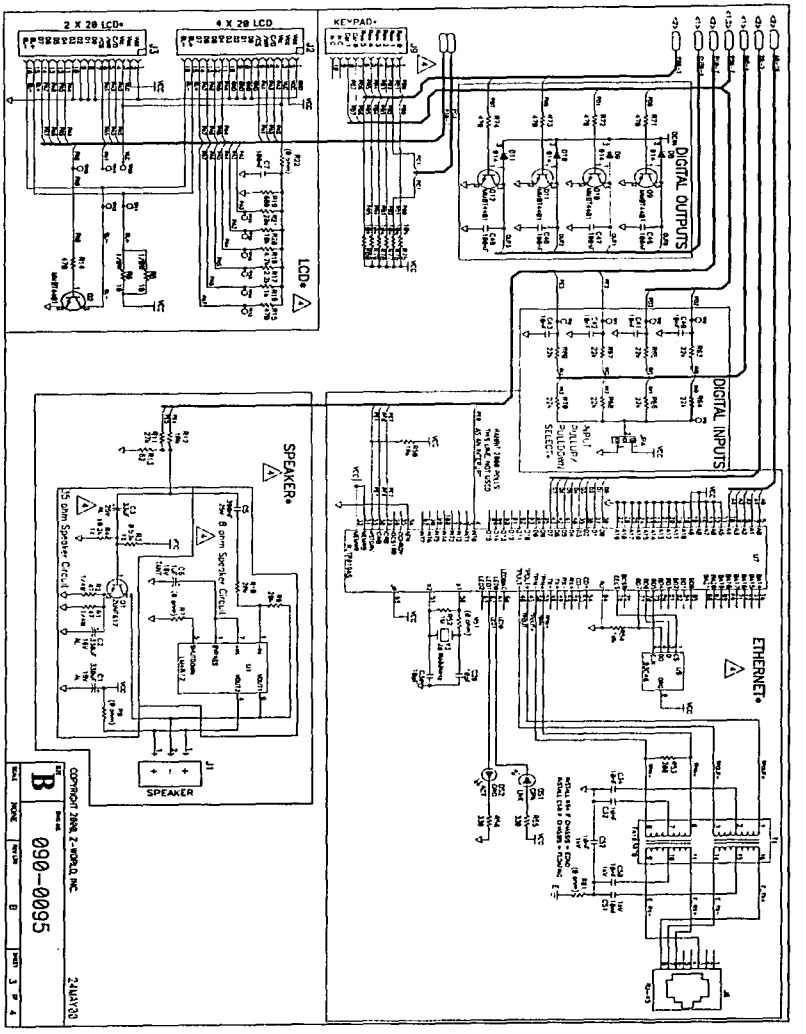
SCHEMATIC DIAGRAM
OP6600/6700 SERIES & RABBIT 2000 TCP/IP DEVELOPMENT KIT

Z-WORLD
2900 SPAFFORD ST. DAVIS, CA 95616
530-757-4616

SIZE B — DWG NO 090-0095

SCALE NONE — SHEET 1 OF 4





TESIS CON
FALLA DE ORIGEN

STUFFING TABLE

	CIRCUIT	PART	MODEL: RABBIT 2000 TCP/IP	OP6700 SERIES	OP6600 SERIES
BACKUP BATTERY	BATTERY MODULE	X1	NOT INSTALLED	INSTALLED	INSTALLED
	MAIN	R48	4.3M	4.3M	4.3M
		R41	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
		C15	100nF	100nF	100nF
		C16	100nF	100nF	100nF
	REGULATOR	Q8	5088 npn	5088 npn	5088 npn
		R35	2M	2M	2M
		R39	220k	220k	220k
	REGULATOR TEMPERATURE COMPENSATION ADJUST	D2	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
		D3	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
		D4	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
		R36	ZERO ohm	ZERO ohm	ZERO ohm
		R37	ZERO ohm	ZERO ohm	ZERO ohm
		R38	ZERO ohm	ZERO ohm	ZERO ohm
CS CONTROL BYPASS		R30	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
PWR- VRAM SW BYPASS		R23	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
PWR SPLY TRANSZORB		TVS1	NOT INSTALLED	SMBJ51	SMBJ51
SRAM	MAIN	U3	128K SRAM	128K SRAM	128K SRAM
		C9	100nF	100nF	100nF
	SRAM SELECT	JP1	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2
FLASH 1	MAIN	U5	256K FLASH	256K FLASH	256K FLASH
	FLASH 1 SELECT	JP2	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2
FLASH 2	MAIN	U6	256K FLASH	NOT INSTALLED	NOT INSTALLED
		C17	100nF	NOT INSTALLED	NOT INSTALLED
	FLASH 2 SELECT	JP3	ZERO ohm ACROSS PINS 1-2	NOT INSTALLED	NOT INSTALLED
RS485		R58	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
RS232		R61	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
		R62	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
POWER IN CONN 2 x 17		J8	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
KEYPAD CONNECTOR		J9	NOT INSTALLED	INSTALLED	INSTALLED
LCD	4 X 20 CONN	J2	NOT INSTALLED	INSTALLED	INSTALLED
	2 X 20 CONN	J3	NOT INSTALLED	NOT INSTALLED	NOT INSTALLED
	MAIN	C7	NOT INSTALLED	100nF	100nF
		Q2	NOT INSTALLED	MMBT4401	MMBT4401
		R5	NOT INSTALLED	10 ohm 1/8W	10 ohm 1/8W
		R6	NOT INSTALLED	10 ohm 1/8W	10 ohm 1/8W
		R14	NOT INSTALLED	470 ohm	470 ohm
		R15	NOT INSTALLED	470 ohm	470 ohm
		R16	NOT INSTALLED	1k	1k
		R17	NOT INSTALLED	2.2k	2.2k
		R18	NOT INSTALLED	4.7k	4.7k
		R19	NOT INSTALLED	680 ohm	680 ohm
		R20	NOT INSTALLED	10k	10k
		R21	NOT INSTALLED	20k	20k
		R22	NOT INSTALLED	ZERO ohm	ZERO ohm

STUFFING TABLE

	CIRCUIT	PART	MODEL: RABBIT 2000 TCP/IP	OP6700 SERIES	OP6600 SERIES
DIGITAL INPUT PULLUP/ PULLDOWN SELECT		JP4	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2	ZERO ohm ACROSS PINS 1-2
ETHERNET	MAIN	U7	RLT8019AS	RLT8019AS	NOT INSTALLED
		C20	100nF	100nF	NOT INSTALLED
		C21	100nF	100nF	NOT INSTALLED
		C26	100nF	100nF	NOT INSTALLED
		C27	100nF	100nF	NOT INSTALLED
		C31	100nF	100nF	NOT INSTALLED
		C33	100nF	100nF	NOT INSTALLED
		R50	10k	10k	NOT INSTALLED
	ETHERNET CRYSTAL	Y3	20.0000MHz	20.0000MHz	NOT INSTALLED
		C29	18pF	18pF	NOT INSTALLED
		C30	18pF	18pF	NOT INSTALLED
		R51	ZERO ohm	ZERO ohm	NOT INSTALLED
		R52	1 MEG	1 MEG	NOT INSTALLED
	EEPROM	U9	93C46	93C46	NOT INSTALLED
		C35	100nF	100nF	NOT INSTALLED
		R54	10k	10k	NOT INSTALLED
	LEDS	DS1	GRN	GRN	NOT INSTALLED
		DS2	ORG	ORG	NOT INSTALLED
		R55	330 ohm	330 ohm	NOT INSTALLED
		R56	330 ohm	330 ohm	NOT INSTALLED
	TRANSFORMER	T1	FA163079	FA163079	NOT INSTALLED
		C32	10nf	10nf	NOT INSTALLED
		C34	10nf	10nf	NOT INSTALLED
		C58	10nf 1KV	10nf 1KV	NOT INSTALLED
		C51	10nf 1KV	10nf 1KV	NOT INSTALLED
		C52	10nf 1KV	10nf 1KV	NOT INSTALLED
		J6	RJ-45	RJ-45	NOT INSTALLED
		R53	200 ohm	200 ohm	NOT INSTALLED
		R81	ZERO ohm	ZERO ohm	NOT INSTALLED
SPEAKER	MAIN	R11	NOT INSTALLED	27k	27k
		R12	NOT INSTALLED	10k	10k
		R13	NOT INSTALLED	62 ohm	62 ohm
	35 ohm CIRCUIT	C1	NOT INSTALLED	330uF 16V	330uF 16V
		C2	NOT INSTALLED	330uF 16V	330uF 16V
		C3	NOT INSTALLED	33uF 25V	33uF 25V
		J1	NOT INSTALLED	INSTALLED	INSTALLED
		Q1	NOT INSTALLED	ZUMT617	ZUMT617
		R1	NOT INSTALLED	47 ohm 1/4W	47 ohm 1/4W
		R2	NOT INSTALLED	47 ohm 1/4W	47 ohm 1/4W
		R3	NOT INSTALLED	9.53k 1%	9.53k 1%
		R4	NOT INSTALLED	10.2k 1%	10.2k 1%
		R8	NOT INSTALLED	ZERO ohm	ZERO ohm
	8 ohm CIRCUIT		NOT INSTALLED	NOT INSTALLED	NOT INSTALLED

24MAY00

SIZE	DWG NO	REV
B	090-0095	
SCALE: NONE	REV LTR: B	SHEET 4 OF 4

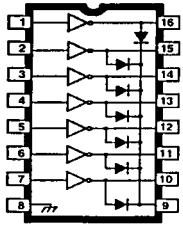
TESIS CON
FALLA DE ORIGEN

ANEXO B. CHIP ULN2003

2003 THRU 2024

Data Sheet 29304F

HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS



Dwg. No. A-9584

Note that the ULN20xxA series (dual in-line package) and ULN20xxL series (small-outline IC package) are electrically identical and share a common terminal number assignment.

ABSOLUTE MAXIMUM RATINGS

Output Voltage, V_{CE}
(ULN200xA and ULN200xL) ..... 50 V
(ULN202xA and ULN202xL) ..... 95 V
Input Voltage, V_{IN} ..... 30 V
Continuous Output Current, I_C ..... 500 mA
Continuous Input Current, I_{IN} ..... 25 mA
Power Dissipation, P_D
(one Darlington pair) ..... 1.0 W
(total package) ..... See Graph
Operating Temperature Range, T_A ..... -20°C to +85°C
Storage Temperature Range, T_S ..... -55°C to +150°C

Ideally suited for interfacing between low-level logic circuitry and multiple peripheral power loads, the Series ULN20xxA/L high-voltage, high-current Darlington arrays feature continuous load current ratings to 500 mA for each of the seven drivers. At an appropriate duty cycle depending on ambient temperature and number of drivers turned ON simultaneously, typical power loads totaling over 230 W (350 mA x 7, 95 V) can be controlled. Typical loads include relays, solenoids, stepping motors, magnetic print hammers, multiplexed LED and incandescent displays, and heaters. All devices feature open-collector outputs with integral clamp diodes.

The ULN2003A/L and ULN2023A/L have series input resistors selected for operation directly with 5 V TTL or CMOS. These devices will handle numerous interface needs — particularly those beyond the capabilities of standard logic buffers.

The ULN2004A/L and ULN2024A/L have series input resistors for operation directly from 6 to 15 V CMOS or PMOS logic outputs.

The ULN2003A/L and ULN2004A/L are the standard Darlington arrays. The outputs are capable of sinking 500 mA and will withstand at least 50 V in the OFF state. Outputs may be paralleled for higher load current capability. The ULN2023A/L and ULN2024A/L will withstand 95 V in the OFF state.

These Darlington arrays are furnished in 16-pin dual in-line plastic packages (suffix "A") and 16-lead surface-mountable SOICs (suffix "L"). All devices are pinned with outputs opposite inputs to facilitate ease of circuit board layout. All devices are rated for operation over the temperature range of -20°C to +85°C. Most (see matrix, next page) are also available for operation to -40°C; to order, change the prefix from "ULN" to "ULQ".

FEATURES

- TTL, DTL, PMOS, or CMOS-Compatible Inputs
- Output Current to 500 mA
- Output Voltage to 95 V
- Transient-Protected Outputs
- Dual In-Line Plastic Package or Small-Outline IC Package

x = digit to identify specific device. Characteristic shown applies to family of devices with remaining digits as shown. See matrix on next page.

Allegro MicroSystems, Inc.

2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

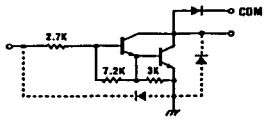
DEVICE PART NUMBER DESIGNATION

$V_{CE(MAX)}$	50 V	95 V
$I_{C(MAX)}$	500 mA	500 mA
Logic	**Part Number**	
5V TTL, CMOS	ULN2003A* ULN2003L*	ULN2023A* ULN2023L
6-15 V CMOS, PMOS	ULN2004A* ULN2004L*	ULN2024A ULN2024L

*Also available for operation between -40°C and +85°C. To order, change prefix from "ULN" to "ULQ".

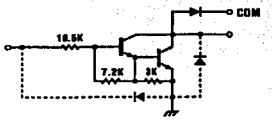
PARTIAL SCHEMATICS

ULN20x3A/L (Each Driver)

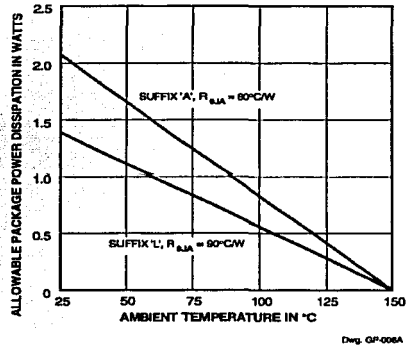


Dwg. No. A-9951

ULN20x4A/L (Each Driver)



Dwg. No. A-9908A



Dwg. GP-006A

X = Digit to identify specific device. Specification shown applies to family of devices with remaining digits as shown. See matrix above.

Allegro MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

Types ULN2003A, ULN2003L, ULN2004A, and ULN2004L
ELECTRICAL CHARACTERISTICS at +25°C (unless otherwise noted).

Characteristic	Symbol	Test Fig.	Applicable Devices	Test Conditions	Limits Min.	Typ.	Max.	Units
Output Leakage Current	I_{CEX}	1A	All	V_{CE} = 50 V, T_A = 25°C	—	< 1	50	µA
				V_{CE} = 50 V, T_A = 70°C	—	< 1	100	µA
		1B	ULN2004A/L	V_{CE} = 50 V, T_A = 70°C, V_{IN} = 1.0 V	—	< 5	500	µA
Collector-Emitter Saturation Voltage	$V_{CE(SAT)}$	2	All	I_C = 100 mA, I_B = 250 µA	—	0.9	1.1	V
				I_C = 200 mA, I_B = 350 µA	—	1.1	1.3	V
				I_C = 350 mA, I_B = 500 µA	—	1.3	1.6	V
Input Current	$I_{IN(ON)}$	3	ULN2003A/L	V_{IN} = 3.85 V	—	0.93	1.35	mA
			ULN2004A/L	V_{IN} = 5.0 V	—	0.35	0.5	mA
				V_{IN} = 12 V	—	1.0	1.45	mA
	$I_{IN(OFF)}$	4	All	I_C = 500 µA, T_A = 70°C	50	65	—	µA
Input Voltage	$V_{IN(ON)}$	5	ULN2003A/L	V_{CE} = 2.0 V, I_C = 200 mA	—	—	2.4	V
				V_{CE} = 2.0 V, I_C = 250 mA	—	—	2.7	V
				V_{CE} = 2.0 V, I_C = 300 mA	—	—	3.0	V
			ULN2004A/L	V_{CE} = 2.0 V, I_C = 125 mA	—	—	5.0	V
				V_{CE} = 2.0 V, I_C = 200 mA	—	—	6.0	V
				V_{CE} = 2.0 V, I_C = 275 mA	—	—	7.0	V
				V_{CE} = 2.0 V, I_C = 350 mA	—	—	8.0	V
Input Capacitance	C_{IN}	—	All		—	15	25	pF
Turn-On Delay	t_{PLH}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Turn-Off Delay	t_{PHL}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Clamp Diode Leakage Current	I_R	6	All	V_R = 50 V, T_A = 25°C	—	—	50	µA
				V_R = 50 V, T_A = 70°C	—	—	100	µA
Clamp Diode Forward Voltage	V_F	7	All	I_F = 350 mA	—	1.7	2.0	V

Complete part number includes suffix to identify package style: A = DIP, L = SOIC.

2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

Types ULN2023A, ULN2023L, ULN2024A, and ULN2024L
ELECTRICAL CHARACTERISTICS at +25°C (unless otherwise noted).

Characteristic	Symbol	Test Fig.	Applicable Devices	Test Conditions	Limits Min.	Typ.	Max.	Units
Output Leakage Current	I_{CEX}	1A	All	V_{CE} = 95 V, T_A = 25°C	—	< 1	50	µA
				V_{CE} = 95 V, T_A = 70°C	—	< 1	100	µA
		1B	ULN2024A/L	V_{CE} = 95 V, T_A = 70°C, V_{IN} = 1.0 V	—	< 5	500	µA
Collector-Emitter Saturation Voltage	$V_{CE(SAT)}$	2	All	I_C = 100 mA, I_B = 250 µA	—	0.9	1.1	V
				I_C = 200 mA, I_B = 350 µA	—	1.1	1.3	V
				I_C = 350 mA, I_B = 500 µA	—	1.3	1.6	V
Input Current	$I_{IN(ON)}$	3	ULN2023A/L	V_{IN} = 3.85 V	—	0.93	1.35	mA
			ULN2024A/L	V_{IN} = 5.0 V	—	0.35	0.5	mA
				V_{IN} = 12 V	—	1.0	1.45	mA
	$I_{IN(OFF)}$	4	All	I_C = 500 µA, T_A = 70°C	50	65	—	µA
Input Voltage	$V_{IN(ON)}$	5	ULN2023A/L	V_{CE} = 2.0 V, I_C = 200 mA	—	—	2.4	V
				V_{CE} = 2.0 V, I_C = 250 mA	—	—	2.7	V
				V_{CE} = 2.0 V, I_C = 300 mA	—	—	3.0	V
			ULN2024A/L	V_{CE} = 2.0 V, I_C = 125 mA	—	—	5.0	V
				V_{CE} = 2.0 V, I_C = 200 mA	—	—	6.0	V
				V_{CE} = 2.0 V, I_C = 275 mA	—	—	7.0	V
				V_{CE} = 2.0 V, I_C = 350 mA	—	—	8.0	V
Input Capacitance	C_{IN}	—	All		—	15	25	pF
Turn-On Delay	t_{PLH}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Turn-Off Delay	t_{PHL}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Clamp Diode Leakage Current	I_R	6	All	V_R = 95 V, T_A = 25°C	—	—	50	µA
				V_R = 95 V, T_A = 70°C	—	—	100	µA
Clamp Diode Forward Voltage	V_F	7	All	I_F = 350 mA	—	1.7	2.0	V

Complete part number includes suffix to identify package style: A = DIP, L = SOIC.

Allegro MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

TEST FIGURES

FIGURE 1A

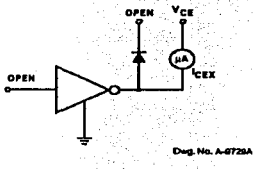


FIGURE 1B

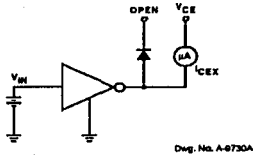


FIGURE 2

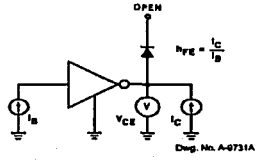


FIGURE 3

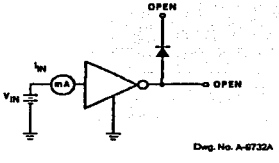


FIGURE 4

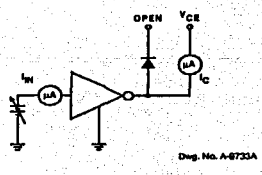


FIGURE 5

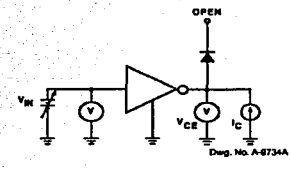


FIGURE 6

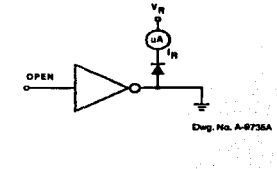


FIGURE 7

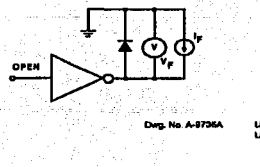
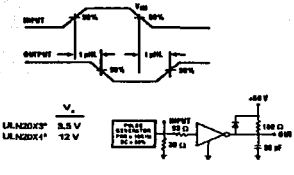


FIGURE 8



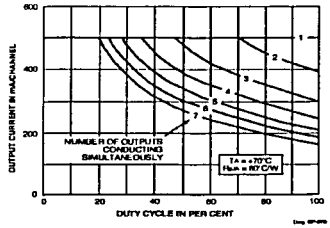
* Complete part number includes a final letter to indicate package.

X = Digit to identify specific device. Specification shown applies to family of devices with remaining digits as shown.

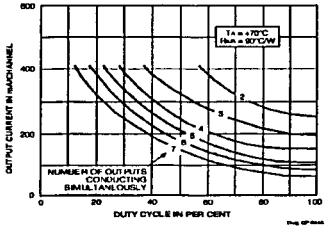
2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

ALLOWABLE COLLECTOR CURRENT AS A FUNCTION OF DUTY CYCLE

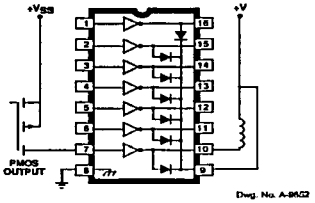
(Dual In-line-Packaged Devices, Suffix 'A')



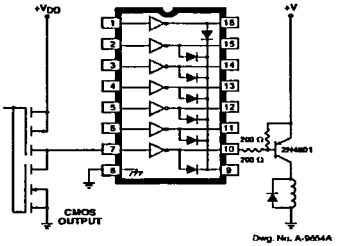
(Small-Outline-Packaged Devices, Suffix 'L')



TYPICAL APPLICATIONS

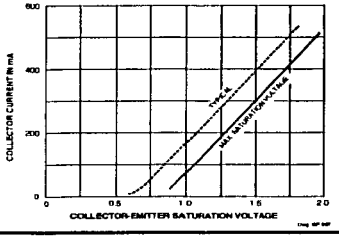


Dwg. No. A-9662

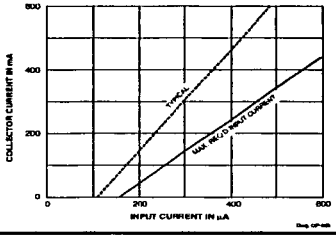


Dwg. No. A-9664A

SATURATION VOLTAGE AS A FUNCTION OF COLLECTOR CURRENT



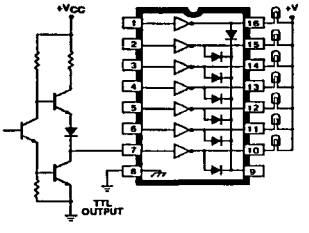
COLLECTOR CURRENT AS A FUNCTION OF INPUT CURRENT

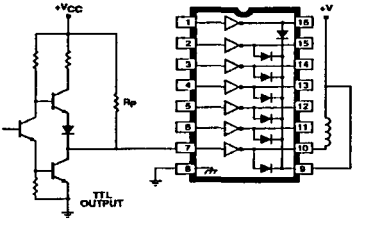


Allegro MicroSystems, Inc.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

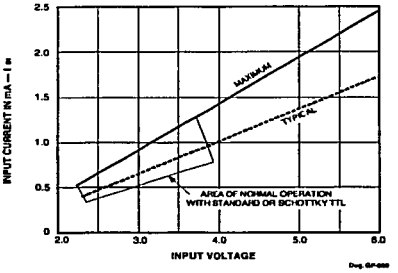
TYPICAL APPLICATIONS



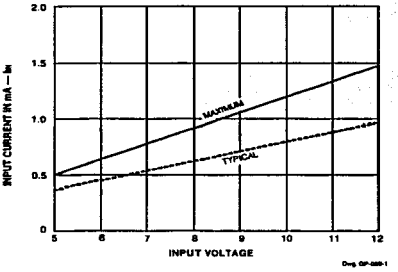


INPUT CURRENT AS A FUNCTION OF INPUT VOLTAGE

Types ULN2003A, ULN2003L, ULN2023A, and ULN2023L

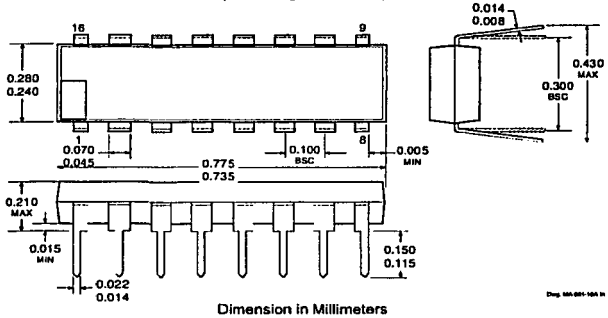


Types ULN2004A, ULN2004L, ULN2024A, and ULN2024L

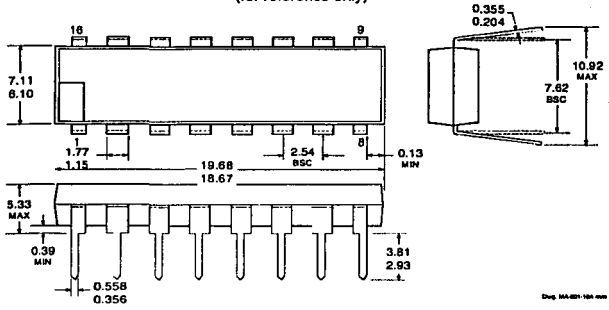


PACKAGE DESIGNATOR "A"

Dimensions in Inches
(controlling dimensions)



Dimension in Millimeters
(for reference only)

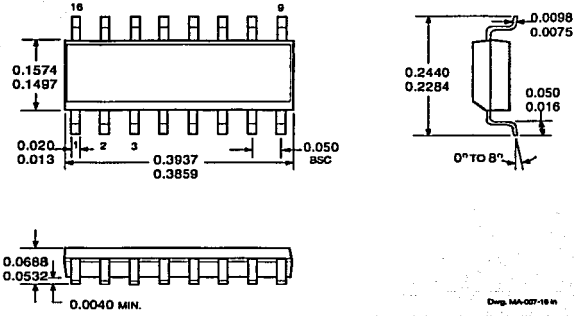


NOTES: 1. Leads 1, 8, 9, and 16 may be half leads at vendor's option.
2. Lead thickness is measured at seating plane or below.
3. Lead spacing tolerance is non-cumulative.
4. Exact body and lead configuration at vendor's option within limits shown.

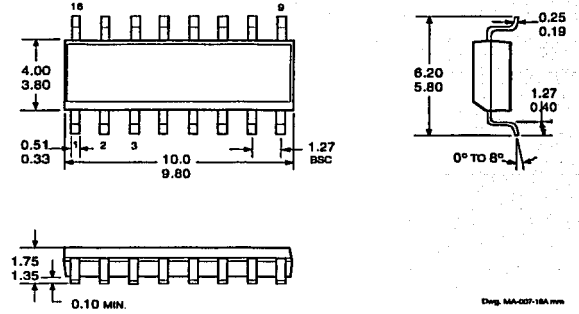
Allegro MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

PACKAGE DESIGNATOR "L"

Dimensions in Inches
(for reference only)



Dimension in Millimeters
(controlling dimensions)



NOTES: 1. Lead spacing tolerance is non-cumulative.
2. Exact body and lead configuration at vendor's option within limits shown.

2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

Allegro MicroSystems, Inc.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

BIBLIOGRAFÍA

1.- Guía Completa de Protocolos de Telecomunicaciones.
García Brage A.
Ed. McGrawHill / Interamericana de España.
Madrid, Primera Edición en Español 2002.

2.- Academia de Networking de Cisco Systems: Guía del primer año.
PEARSON EDUCACIÓN, S.A.
Madrid, Segunda Edición 2002.

3.- Protocolos de Internet. Diseño e implementación en sistemas UNÍX.
Lopez Gonzales A. / Novo Lopez A.
Ed. ALPHAOMEGA GRUPO EDITOR S.A de C.V.
México DF, 2000.

4.- Servidor Apache.
Rich Bowen / Den Coar.
Ed. PRENTICE HALL.
Madrir, Primera Edición 2000.

5.- Programación en LINUX.
Kurt Wall.
Ed. PRENTICE HALL.
Madrir, Segunda Edición 2001.

Sitios de Internet de referencia:

1. http://w3.org
2. http://rabbitsemiconductor.com
3. http://zworld.com
4. http://[illegible]
5. http://[illegible]/clases.html