PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

DESARROLLO DE UNA INTERFAZ USB PARA EL CONTROL DE ESTACIONES DE RADIO HF Y VHF PARA COMUNICACIÓN DE DATOS

TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

PRESENTADA POR:

HIM CUPER CANSAYA HERRERA

LIMA – PERÚ

2005

DESARROLLO DE UNA INTERFAZ USB PARA EL CONTROL DE ESTACIONES DE RADIO HF Y VHF PARA COMUNICACIÓN DE DATOS

RESUMEN

El desarrollo de este proyecto, denominado “Placa estación USB” está dividido en cuatro partes.

La primera consiste en la descripción teórica del funcionamiento del puerto USB y del funcionamiento del sensor de ROE; ambas descripciones, sobre todo la primera, son cruciales para poder usar luego los conceptos y la terminología necesarias para el desarrollo de las siguientes partes que son netamente de cálculo e implementación.

La segunda parte consiste en el desarrollo de todo lo concerniente al software que se subdividen en el software contenido en el dispositivo USB, denominado Firmware, y en el software de computadora con el que interactuará dicho dispositivo. En la tercera parte se desarrolla el hardware que se necesita para que el dispositivo USB pueda interactuar con el mundo físico, es decir se describen sus elementos de entrada o sensores y los elementos de salida o actuadores para que de esta manera el dispositivo se pueda convertir en la “Placa Estación USB” encargada del control de las estaciones de radio HF y VHF.

En la cuarta y última parte se detalla el funcionamiento del equipo en conjunto tanto de hardware como de software así como la instalación en una computadora de escritorio para un usuario final, habiendo logrado con esto darle un uso práctico a nuestro desarrollo en un campo real como es el de transmisión de datos en las comunicaciones rurales

Este trabajo está dedicado a mis padres Teresita y Cúper y a todos mis hermanos por su inmenso apoyo y cariño

ÍNDICE

ÍNIDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS

ÍNDICE DE TABLAS

ANEXOS

1. INTRODUCCIÓN

El proyecto Enlace Hispano–Americano de Salud (EHAS) cuyo beneficiario directo es el Ministerio de Salud (MINSA) ofrece a los agentes prestadores de servicios de salud una red de telecomunicaciones de bajo costo y un conjunto de servicios de telemedicina adaptados al entorno rural. Un objetivo específico del EHAS es desarrollar y sistematizar la tecnología adecuada a ser utilizada para conseguir el objetivo general, que es el mejoramiento de la atención de salud en el sector rural.

La presente tesis describe el trabajo de ingeniería, investigación, medición, cálculo, diseño, implementación e instalación de un equipo electrónico denominado Placa Estación USB, el cual se encontrará en estaciones de radios HF y/o VHF rurales y en estos lugares remotos se encargará de automatizarlas, logrando de esta manera conocer el estado de los parámetros más importantes de cada locación y así dejarlos listos para poder leerlos desde cualquier punto de la red. También se trata de optimizar el uso de cada estación facilitando las funciones del usuario no especializado.

Como parte complementaria, se desea introducir al desarrollador al mundo del manejo de las interfaces de computadora de vanguardia, siendo una de ellas la interfaz del puerto USB, que tiene como característica principal la sencillez en el manejo para el usuario, pero por lo mismo mayor complejidad para el desarrollador.

Para lograr un mejor entendimiento se ha trabajado paralelamente en los dos sistemas operativos más conocidos y usados: Windows (98SE, 2000, XP) y LINUX DEBIAN en la versión Metadistro EHAS-GNOME Versión 2.44. Éste último es en donde se da mayor énfasis a nuestro desarrollo pues es aquí en donde está implementado el software módem del programa EHAS "**Ehas-Station**" para comunicación de datos en HF y VHF, y sobre el cual se ha agregado nuestra aplicación, quedando integrados como un solo paquete tanto de hardware como de software.

2. CONSIDERACIONES TEÓRICAS

2.1. DESCRIPCIÓN DE UN RADIO ENLACE.

Un radio enlace de estaciones HF y VHF como el desarrollado por el programa EHAS tiene el siguiente esquema básico:

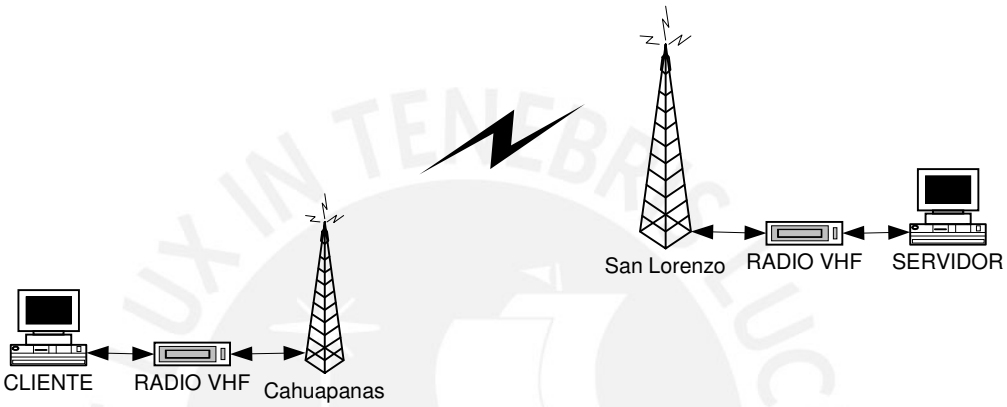


Figura 2.1.1 Enlace de dos estaciones de radio

Los elementos presentes son los siguientes:

- Computadora (de escritorio o portátil).
- Radio VHF o HF.
- Torre con la antena montada según sea el caso HF o VHF
- Accesorios como: Cables de interconexión entre los equipos, inversor de voltaje e interfaz de datos (Esta parte se explica más adelante pues va incluida en nuestro circuito).

En la mayoría de los centros de salud no hay fluido eléctrico constante por lo que se alimentan a los equipos mediante un sistema fotovoltaico que sólo mencionaremos en este trabajo.

2.1.1. COMUNICACIONES EN VHF

Una estación servidor VHF es la encargada de manejar las transacciones de datos entre sus clientes que tenga a su cargo como se muestra en el siguiente diagrama:

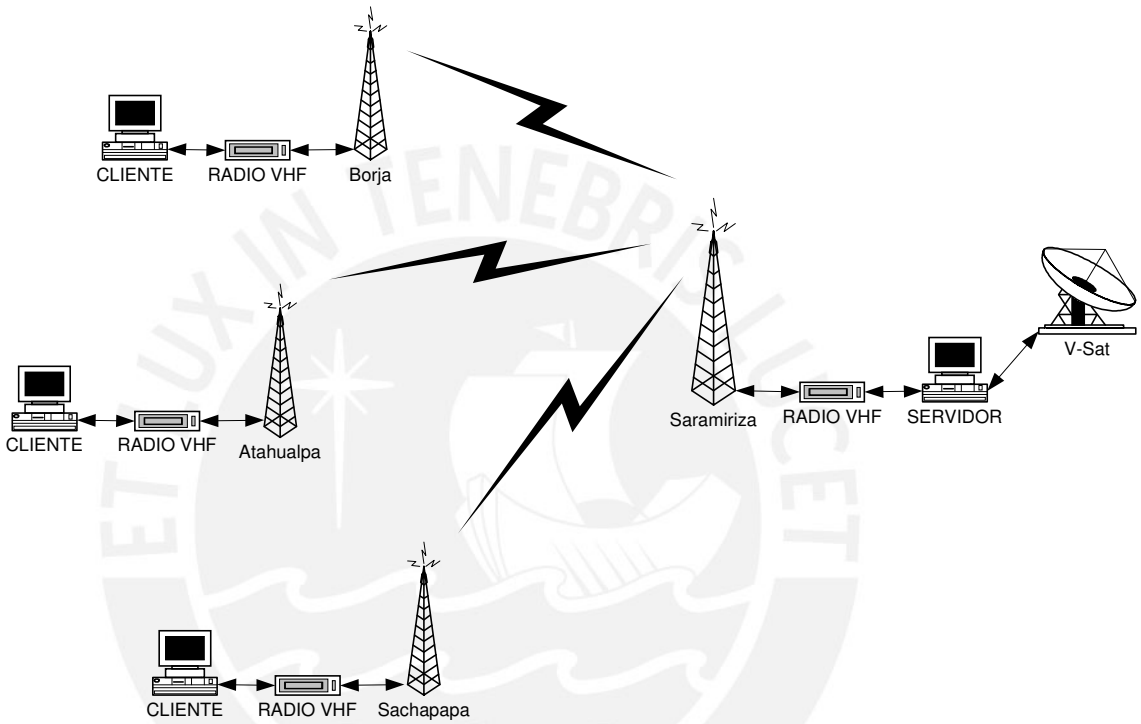


Figura 2.1.1.1 Red centralizada VHF

Como las comunicaciones en la banda VHF requieren de cierta línea de vista estas estaciones no pueden estar muy alejadas del servidor lo que les da un rango medio de hasta 40km aproximadamente para una comunicación óptima y la salida a Internet se da mediante una conexión Vsat.

2.1.2. COMUNICACIONES EN HF

Ya que las comunicación es HF no requieren de línea de vista y pueden cubrir grandes distancias tendremos un solo servidor que se encargue de manejar las transacciones entre los clientes ubicados en distintas zonas del país como se muestra en el diagrama.

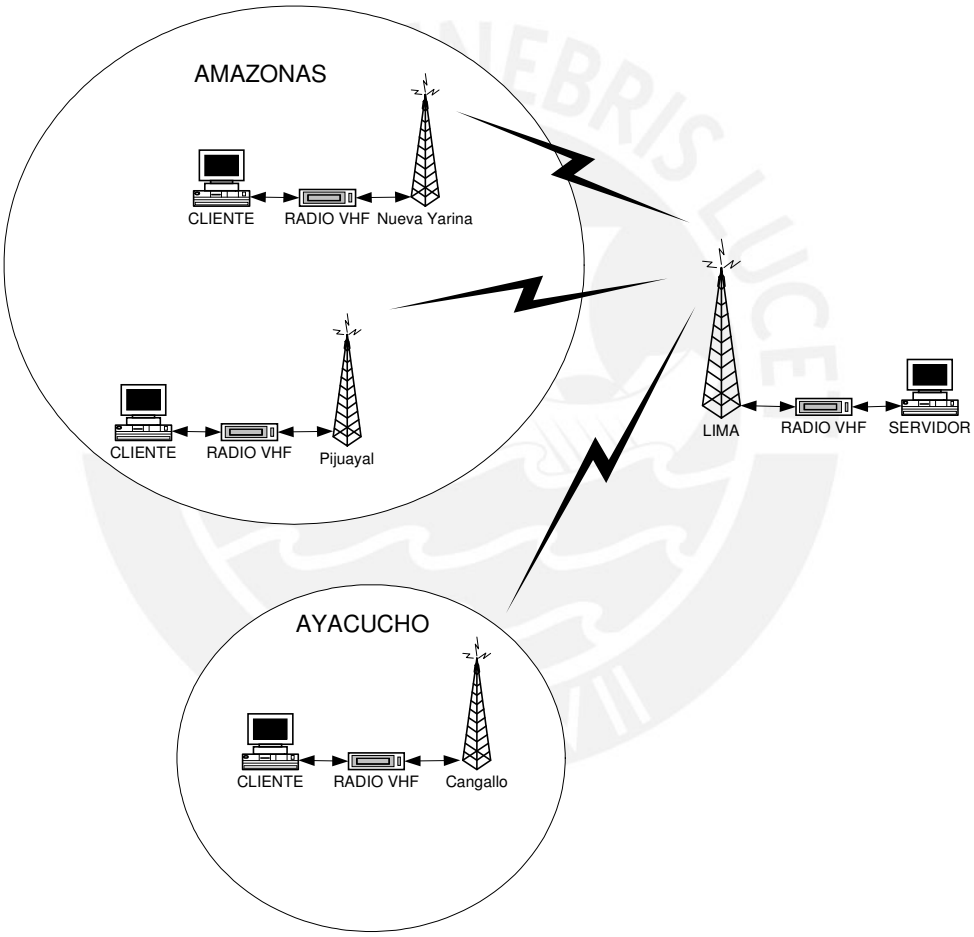


Figura 2.1.2.1 Red centralizada HF

2.2. EL PUERTO USB

El puerto USB (Universal Serial Bus) es un interfaz muy flexible que fue diseñado para interconectar dispositivos externos o periféricos a la computadora en el cual ya vienen implementados uno o más puertos USB. Esta interfaz es lo suficientemente versátil como para soportar periféricos estándar como los HID (Dispositivos de Interfaz Humana), dispositivos de almacenamiento masivo o también dispositivos más especializados inclusive los diseños propios siempre y cuando se ajusten a las especificaciones necesarias.

La idea es que el usuario final no tenga ninguna dificultad al momento de hacer uso del dispositivo, ya sea tener que configurar el software o el hardware; es por eso que la dificultad se presenta del lado del desarrollador, ya que en este sentido la complejidad es mayor, para ello es necesario conocer primero cómo funciona la interfaz USB tanto del lado del dispositivo externo como del de la computadora.

2.2.1. CARACTERÍSTICAS GENERALES

En el siguiente cuadro comparativo entre el puerto USB y otros interfaces conocidos podemos apreciar algunas de sus características generales.

INTERFAZ	FORMATO	Nro. Máximo de dispositivos	Longitud del cable (m)	Velocidad máxima (bps)	Uso típico
USB	Serial asíncrono	127	5 (o hasta 30 m con 5 hubs)	1.5M, 12M, 280M	Mouse, teclados, unidades externas, módem, audio
RS-232 (EIA/TIA-232)	Serial asíncrono	2	15 - 30	20k (115k con algún adicional)	Módem, mouse, instrumentación
SPI	Serial sincrónico	8	3	2.1M	Microcontroladores
I²C	Serial sincrónico	40	5.5	3.4M	Microcontroladores
Fire Wire (IEEE-1394)	Serial	64	4.5	400M (3.2G con 1394b)	Vídeo, almacenamiento
Ethernet	Serial	1024	400	10M/100M/1G	Redes de computadoras
Puerto Paralelo	Paralelo	2 (8 con hardware extra)	3 – 10	8M	Impresoras, escaners, discos externos

Tabla 2.2.1.1 Comparación con otros interfaces

Existen dos versiones para el puerto USB, que básicamente se diferencian en cuanto a la velocidad de transferencia:

- USB 1.0, con dos velocidades:
  - Velocidad baja a 1.5 Megabits por segundo
  - Velocidad media a 12 Megabits por segundo
- USB 2.0, con tres velocidades, las dos anteriores y la que sigue:
  - Velocidad alta a 280 Megabits por segundo

De acuerdo a los requerimientos de nuestra aplicación nos centraremos, en adelante, a trabajar con la versión USB 1.0 de baja velocidad.

2.2.2. ESQUEMA DE CONEXIÓN Y DEFINICIÓN DE TÉRMINOS

A continuación nos referiremos al siguiente esquema de conexión.

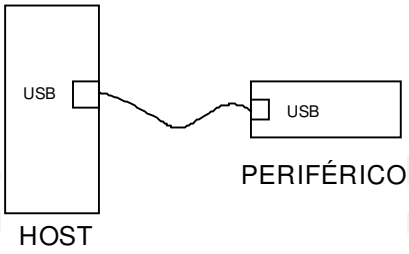


Figura 2.3.2.1 Conexión del Host al periférico

Donde el **HOST** representa al mecanismo USB, con su respectivo hardware y firmware de comunicación, que interactúa entre el **DISPOSITIVO PERIFÉRICO** y la Computadora.

Debido a la flexibilidad de conexión se puede tener también el siguiente esquema de conexión.

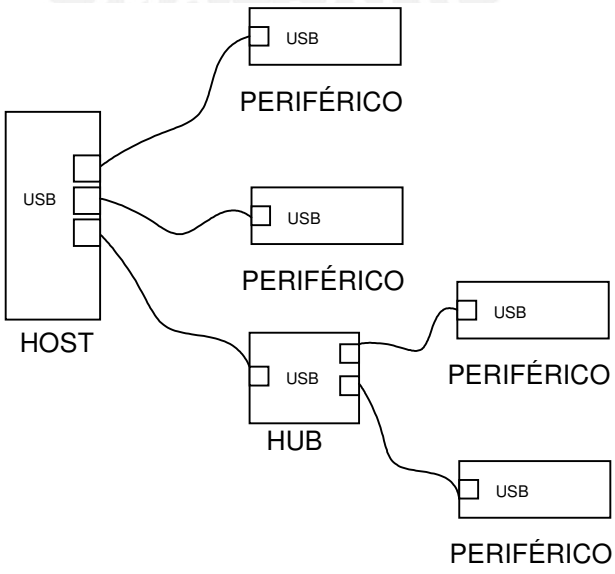


Figura 2.3.2.2 conexión en cascada de los periféricos al Host

También podemos apreciar algunos **HUB** que básicamente sirven para expandir el número de entradas al Host, los cuales, en gran medida, no afectarán a nuestras aplicaciones. El número máximo de periféricos y HUBs que se pueden conectar a un mismo Host es de 127 (incluyendo el HUB raíz) y en cascada se pueden colocar no más de 5 HUB externos.

Hay una confusión a la hora de definir la palabra **"puerto"** ya que no podemos compararlo a la idea de lo que inicialmente se tenía con respecto al puerto serial o paralelo, ya que estos tienen una sola ruta o dirección específica al Host, en cambio en un puerto USB la cosa cambia ya que todos los periféricos comparten la misma ruta, en este caso para diferenciar a un dispositivo de otro ya es cuestión de programación del firmware y del controlador del respectivo periférico y se efectúa en el proceso de Enumeración que se explicará más adelante.

2.2.3. TRANFERENCIA USB

Para desarrollar una aplicación que interactúe con un dispositivo USB es necesario entender lo que pasa dentro de una transferencia en el bus USB.

Un enlace USB se divide en dos categorías, dependiendo de cómo están interactuando el dispositivo con Host.

- Enlace de Configuración.- Aquí la comunicación es exclusivamente para que el Host aprenda del Periférico y ambos se preparan para el intercambio de información. Durante la enumeración el Host le hace una serie de pedidos al Periférico el cual debe identificar y contestar cada unos de estos para después tomar acción de cada uno de los pedidos. En la computadora el Sistema Operativo se encarga de realizar el proceso de la enumeración sin que el desarrollador tome parte aún, en este proceso el sistema le asigna un controlador al dispositivo y ya se puede dar la siguiente etapa

- Enlace de Aplicación.- Aquí la comunicación está dedicada a la transacción de información entre la aplicación respectiva y el Dispositivo para lo cual fue destinado. Para acceder a los datos que se recogen del dispositivo se tiene que acceder a las aplicaciones de hardware USB con que cuente el sistema operativo tales como las funciones API del sistema operativo, que explicaremos con más detalle en nuestra aplicación.

2.2.3.1. ELEMENTOS DE LA TRANSFERENCIA

Para entender mejor una transferencia USB necesitamos ver lo que pasa en cada nivel. Una transferencia está compuesta por transacciones y cada transacción por paquetes y cada paquete contiene información. Para entender cada uno de ellos debemos empezar por entender acerca de los “Puntos terminales” y los “túneles”.

PUNTOS TERMINALES

Un punto terminal es un buffer que almacena múltiples bytes, típicamente es una memoria de datos o un registro en un microcontrolador. Todas las transmisiones van de un punto terminal o hacia otro punto terminal. El host también tiene buffers en donde se almacena la data recibida o los data lista para ser transmitida los cuales son conocidos como los “puntos de partida” que son los que inician la comunicación con los puntos terminales del dispositivo. Por definición un punto terminal lleva información en un sólo sentido, siempre y cuando no se trate de una transferencia de **control** que se verá más adelante.

Los puntos terminales tienen como características al número de punto terminal que varía de 0 a 15 y el sentido del flujo de información que desde la perspectiva del host se define como:

IN: Los datos fluyen del dispositivo hacia el host.
OUT: Los datos fluyen del host al dispositivo.

Sólo en una transferencia de control se tiene un par compuesto de puntos terminales IN y OUT que comparten un sólo número de punto terminal, que por lo general es el punto **terminal 0**. En raras ocasiones se requerirá usar otra transferencia de control las cuales sólo las soportan algunos microcontroladores, no siendo éste el caso para nuestra aplicación. En una transferencia la información fluye en una sola dirección, sin embargo en un flujo de estatus y control los datos pueden ir en direcciones opuestas. Un sólo número de punto terminal puede soportar una transferencia IN y una OUT configurando debidamente el dispositivo.

Cada transacción en el bus incluye el número de punto terminal y un código que indique la dirección del flujo de datos o si la transacción va a iniciar una transferencia de control, los códigos son IN, OUT y SETUP; lo cual se puede apreciar mejor en el siguiente cuadro:

TIPO DE TRANSACCIÓN	FUENTE DE INFORMACION	TRANSFERENCIAS QUE LO USAN	CONTENIDO
IN	DISPOSITIVO	TODAS	GENÉRICO
OUT	HOST	TODAS	GENÉRICO
SETUP	HOST	CONTROL	UN PEDIDO

Tabla 2.2.3.1.1 Características de puntos terminales

TÚNELES

Antes que se dé la transferencia de información entre el host y el dispositivo se debe establecer un túnel o camino virtual, que es simplemente una asociación entre el punto terminal del dispositivo y el software del controlador en el host.

Estos túnel se establecen ni bien el sistema se incializa o un dispositivo

es conectado, y se quitan cuando se quita el dispositivo del bus o también cuando el host así lo pide.

2.2.3.2. TIPOS DE TRANSFERENCIA

Ya que el sistema USB fue diseñado para manejar muchos tipos de periféricos con requerimientos diferentes ya sea en su tasa de transferencia, tiempo de respuesta y corrección de errores, para ello se cuenta con 4 tipos de transferencia de datos, cada uno para diferentes necesidades

TIPO DE TRANSFERENCIA	CONTROL	AVALANCHA	INTERRUPCIÓN	ISOCRÓNICO
USO TÍPICO	configuración	Impresora, scanner	Ratón, teclado	Audio
¿REQUERIDO?	Sí	No	No	No
¿PERMITIDO EN DISPOSITIVOS DE BAJA VELOCIDAD?	Sí	No	Sí	No
Vel. Máxima de transferencia por túnel en baja velocidad (bytes/milisegundo)	24 (3 transacciones de 8 bytes)	No permitido	0,8 (8 bytes en 10 milisegundos)	No permitido
Dirección del flujo de datos	IN y OUT	IN o OUT	IN o OUT (Sólo IN para la versión 1.0)	IN o OUT
¿Corrección de errores?	Sí	Sí	Sí	No
¿Información de tipo Ráfaga o Mensaje?	Mensaje	Ráfaga	Ráfaga	Ráfaga
¿Tasa fija de transferencia garantizada?	No	No	No	Sí
¿Latencia garantizada?	No	No	Sí	Sí

Tabla 2.2.3.1.2 Características de los tipos de transferencia

2.2.4. ENUMERACIÓN

Ésta es la parte en donde el host aprende acerca del dispositivo. Aquí se da el primer intercambio de información entre ambas partes y éste proceso se debe dar lo siguiente:

- Asignar una dirección al dispositivo.
- Leer la estructura de datos del dispositivo.
- Asignar y cargar el controlador respectivo.
- De las opciones presentadas en la información recuperada, seleccionar la configuración en que trabajará en adelante.

Después de este proceso el dispositivo ya está configurado y listo para transferir información con los puntos terminales debidamente configurados.

2.2.4.1. PASOS EN EL PROCESO DE ENUMERACIÓN

1.- El usuario conecta físicamente el dispositivo a un puerto USB.- También se da cuando el sistema operativo inicia con el dispositivo ya conectado. El puerto puede estar en el hub raíz o en uno conectado en cascada, aquí es donde el dispositivo se energiza y queda en estado de Energizado.

2.- El hub detecta al dispositivo.- El hub monitorea el voltaje en cada línea de sus puertos. El hub tiene resistencias pull-down de 15Kohm en cada uno de sus puertos, mientras que el dispositivo tiene una resistencia de 1.5Kohm pull-up bien en el puerto D+ para los dispositivos que trabajen a full-speed o en D- par los que trabajen en low-speed. Los dispositivos que trabajen en Hi-speed se conectan como los de full-speed.

3.- El host aprende del nuevo dispositivo.- Una de las obligaciones de un hub (en nuestro caso el del hub raíz) es detectar si un dispositivo se

conecta o se retira, esto lo hace con un túnel de interrupción IN que trae cada hub para reportar estos eventos al host. Dicho evento indica solamente si el hub o qué puerto ha experimentado un evento, luego el host le envía un pedido de Get_Port_Status para averiguar más, el cual es un pedido de cualquier hub entiende. De la información recogida el host sabe cuándo un nuevo dispositivo es conectado.

4.- El hub detecta si un dispositivo es de baja o media velocidad.- Justo antes de que el hub reinicia el dispositivo, el hub determina si el dispositivo es de velocidad baja o media examinando los voltajes en las líneas D+ y D-, al ver qué línea tiene voltaje en alta cuando está en reposo. El hub envía información al host en respuesta del siguiente pedido de GET_PORT_STATUS. Los dispositivos USB 1.x permiten que el hub detecte su velocidad justo después del reincio. Las versiones de USB 2.0 requieren, en cambio, que la detección de velocidad se dé justo antes del reincio de manera que él pueda averiguar si el dispositivo es de alta velocidad, esto durante el reincio.

5.- El hub reinicia el dispositivo.- Cuando el host aprende del nuevo dispositivo el controlador del host le envía al hub un pedido de *Set_Port_Feature*. que le pide al hub que reinicie el puerto. El hub coloca las líneas del dispositivo en condición de Reset por al menos 10 milisegundos, esto se hace sólo cuando hay un nuevo dispositivo, mientras que otros hubs o dispositivos ignoran dicha señal.

6.- El host aprende si un dispositivo de velocidad media soporta alta velocidad.- Se emplean dos señales especiales. El estado Chirp J en donde solamente se usa la línea D+, y el estado Chirp K para la línea D-. Durante el reinicio, un dispositivo de alta velocidad envía una seña Chirp K que lo detecta un hub, también de alta velocidad, y le responde con señales alternadas de Chirp K y J, cuando el dispositivo detecta el patrón KJKJKJ, retira los pull-up de velocidad media y establece una comunicación de alta velocidad.

7.- El hub establece una señal para abrir camino entre el dispositivo y el hub.- El host verifica que el dispositivo haya salido del estado de reinicio para enviarle un pedido de *Get_Port_Status*, un bit dentro de la información regresada se encarga de eso. Cuando el hub quita el estado de reinicio, el dispositivo queda en su estado por defecto, y sus registros USB están en estado de reinicio, entonces el dispositivo está listo para responder a la transferencia de control usando el túnel por defecto que apunta al punto terminal 0; el dispositivo ya puede comunicarse con el host usando la dirección por defecto 00h.

8.- El Host envía un pedido de Get_Descriptor para averiguar el tamaño máximo de paquete del túnel por defecto.- En vista que el host enumera un dispositivo a la vez, se usa siempre la dirección 0 y el punto terminal 0. El octavo byte del descriptor del dispositivo contiene el tamaño máximo de paquete que va a soportar el punto terminal 0. El host del sistema operativo pide 64 bytes, pero después de recibir un paquete (tenga o no 64 bytes), empieza la etapa de *status* en la transferencia. Para completar esta etapa el host del sistema operativo solicita al hub que reinicie el dispositivo (volviendo al paso 5). De acuerdo a las especificaciones, no se es necesario el reinicio ya que los dispositivos deberían ser capaces de manejar que el host abandona una transferencia de control en cualquier momento respondiendo al siguiente paquete de Setup, pero el reinicio es una precaución que asegura que el dispositivo estará en un estado conocido cuando termine el reinicio.

9.- El Host asigna una dirección.- El controlador host asigna una única dirección al dispositivo enviándole un perdido Set_Adress. El dispositivo lee el pedido, da un aviso de reconocimiento y almacena la nueva dirección. El dispositivo se encuentra en esta parte en el estado **Direccionado** Cualquier comunicación a apartir de este punto usarán esta nueva dirección, esto es válido hasta que el dispositivo se desconecte o reinicie o se apague la computadora. Para la siguiente enumeración la dirección que se le asigne puede ser diferente.

10.- El Host aprende las habilidades del dispositivo.- El host le envía un pedido de *Get_Descriptor* a la nueva dirección para leer el descriptor del dispositivo, esta vez para leerlo por completo. El descriptor es una estructura de datos que contiene el tamaño máximo del paquete para el punto terminal 0, el número de configuraciones que el dispositivo soporta y otra información básica del dispositivo.

11.- El host asigna y carga el controlador del dispositivo.- Después de aprender todo lo que se pueda del dispositivo a partir de sus descriptores, empieza la búsqueda del controlador que mejor le cuadre para la administración en la comunicación con el dispositivo.

12.- El controlador del dispositivo selecciona una configuración.- En esta parte el controlador asignado solicita la configuración al dispositivo enviándole un pedido de *Set_Configuration* con el número de configuración deseada. La mayoría de dispositivos soportan un único tipo de configuración, pero si el dispositivo soporta múltiples configuraciones entonces el dispositivo se encarga de elegir el tipo de configuración en base a la información recogida del dispositivo o le pide al usuario que elija. El dispositivo se encuentra en esta parte en el estado **Configurado** y la interfaz al dispositivo está ya habilitada.

A partir de este momento el dispositivo ya se encuentra listo para ser usado. Existen otros dos estados para el dispositivo, colgado y suspendido que se pueden dar en cualquier momento.

Colgado.- Se da cuando el hub no está alimentando (Vbus) al puerto que ocurre cuando el hub ha detectado una condición de sobre-corriente o si el hub le pide al hub que quite la alimentación del puerto. Sin energía en Vbus el host y el dispositivo no pueden comunicarse, es como si el dispositivo no estaría conectado.

Suspendido.- Significa que el dispositivo no ha visto actividad, incluso marcadores de inicio-de-trama, en el bus por al menos 3 milisegundos. En este estado el dispositivo debe consumir lo mínimo de la alimentación del bus. Tanto los dispositivos configurados como los no configurados deben soportar este estado.

2.3. EL CONTROLADOR

El controlador del dispositivo es el que hace posible la comunicación entre los controladores de USB del sistema y las aplicaciones que quieran acceder al dispositivo y a su vez al hardware que éste maneje el cual puede ser una impresora, módem, teclado, etc. Estos dispositivos pueden ser periféricos estándar o diseños específicos para aplicaciones especiales. Algunos controladores son conocidos como “Controladores Clase” que manejan las comunicaciones de una variedad de dispositivos con funciones similares.

Una característica del controlador del dispositivo es el de aislar a las aplicaciones de tener que saber o manejar cualquier detalle acerca de las conexiones físicas, señales y protocolos que se requieran en la comunicación con el dispositivo. Una aplicación es el programa con el que finalmente el usuario interactúa y soporta el hardware que necesite usar, en nuestro caso un hardware USB. El controlador tiene la misión de hacer de traductor entre el código de nivel de aplicación y el código de nivel de hardware. El código de nivel de aplicación usa funciones soportadas por el sistema operativo para comunicarse con el controlador del dispositivo; y el código de nivel de hardware maneja los protocolos necesarios para acceder al circuito que maneje el dispositivo el cual incluye el detectar las etapas de las señales *status* y el uso o no de las señales de control en tiempos apropiados.

El sistema operativo ya sea Windows o Linux incluye funciones de interfaz de aplicación para programadores o su sigla en inglés API, que habilitan a las aplicaciones para comunicarse con el controlador del dispositivo. Las aplicaciones escritas en lenguaje de alto nivel como C, C++, Delphi o Python

pueden llamar a las funciones API. Tres de las funciones que el controlador puede soportar para la lectura y escritura de los dispositivos USB son: Lectura (ReadFile), escritura (WriteFile) y entrada/salida de control (DeviceIOControl).

Para elegir un controlador que se acomode a las necesidades del diseñador es necesario saber con qué opciones se cuenta, algunas veces ya el sistema operativo los tiene o el chip que se tenga con el que se trabaje los provea, o se pueda conseguir de algún otro sitio, y para algunas dispositivos puede ser necesario escribir un controlador personalizado, para lo cual se puede contar con una variedad de herramientas que faciliten y acorten esta tarea. Algunas veces hay más de una manera de conseguir el controlador lo que dependerá del costo, facilidad y rendimiento que se considere a la hora de la elección del camino a tomar.

1.- DISPOSITIVO ESTÁNDAR.- Muchos dispositivos periféricos muy usados pertenecen a esta clase tales como disqueteras, impresoras, teclados, etc. Los cuales ya están dentro de un lista y para ellos el sistema operativo incluye a los controladores clase.

2.- DISPOSITIVOS PERSONALIZADOS.- Algunos periféricos son dispositivos personalizados hechos para una aplicación específica tales como unidades de adquisición de datos, controladores de motores, instrumentos de prueba o la aplicación que se desarrollamos nosotros. Estos dispositivos pueden usar controladores personalizados o pueden ser acomodados de tal forma que cumplan con los requerimientos de un controlador clase. Por ejemplo, un sistema de adquisición de datos puede usar tranquilamente un controlador clase HID (Dispositivo de Interfaz Humano) que explicaremos más adelante.

MODO USARIO Y MODO KERNEL

Dentro del sistema operativo hay una de dos modos de correr el código: kernel y usuario. Cada uno permite un diferente nivel de privilegios de acceder a la

memoria u otros recursos del sistema. Todas las aplicaciones deben correr en modo usuario. La mayoría de los controladores, incluidos todos los controladores USB, corren en el modo kernel, sin embargo un dispositivo USB puede tener también un controlador de modo usuario complementario. Normalmente el modo usuario está mas restringido al acceso a la memoria y recursos del sistema que éste considere protegidos lo que hace posible que el sistema operativo corra diferentes aplicaciones a la vez, en cambio el modo kernel el código tiene acceso irrestricto a tales recursos y memoria del sistema como son la capacidad de ejecutar instrucciones de administración de memoria y control de acceso a puertos de entrada salida E/S. En Windows 98 y Me y linux, las aplicaciones pueden acceder a los puertos E/S directamente siempre y cuando un controlador de bajo nivel no haya reservado tal puerto. En Windows NT y 2000, solamente el modo kernel puede acceder a los puertos E/S. La figura 2.4.1 muestra los componentes más comunes tanto en modo usuario como el modo kernel usados en una comunicación USB.

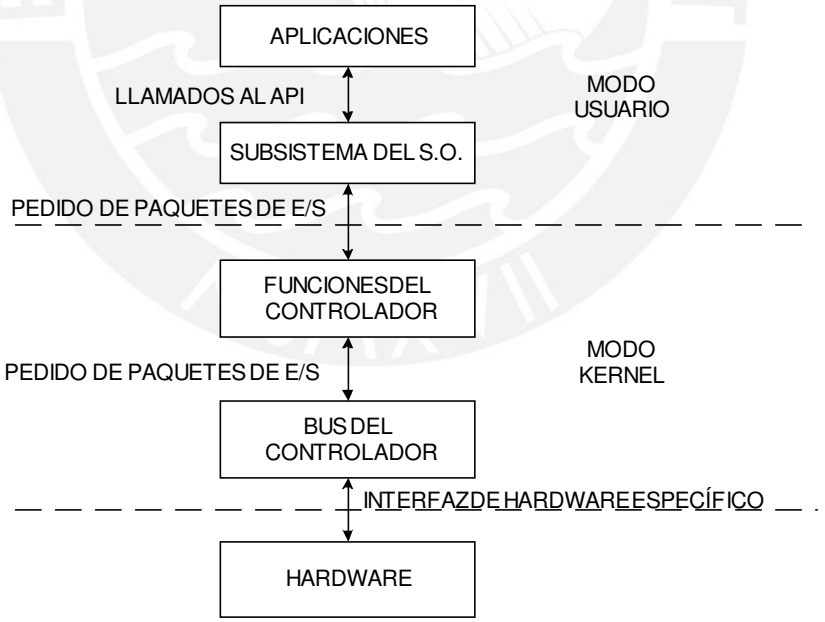


Figura 2.3.1.- Capas de un modelo de controlador en el sistema operativo con diferentes controladores tanto para el dispositivo como para los buses

CONTROLADOR DE FUNCIONES

Un controlador de funciones le permite a la aplicación conversar con el dispositivo USB usando funciones API, las cuales son parte del subsistema Win32 en Windows y están a cargo de las funciones de usuario tales como correr aplicaciones, administrar las entradas del usuario a través del teclado o ratón y mostrar las salidas en el monitor, en Linux se encuentran en las librerías del directorio "usr". El controlador de funciones sabe cómo comunicarse con los niveles más bajos del controlador del bus que maneja el hardware. La figura 2.3.2 muestra cómo trabajan tales controladores en una comunicación USB.

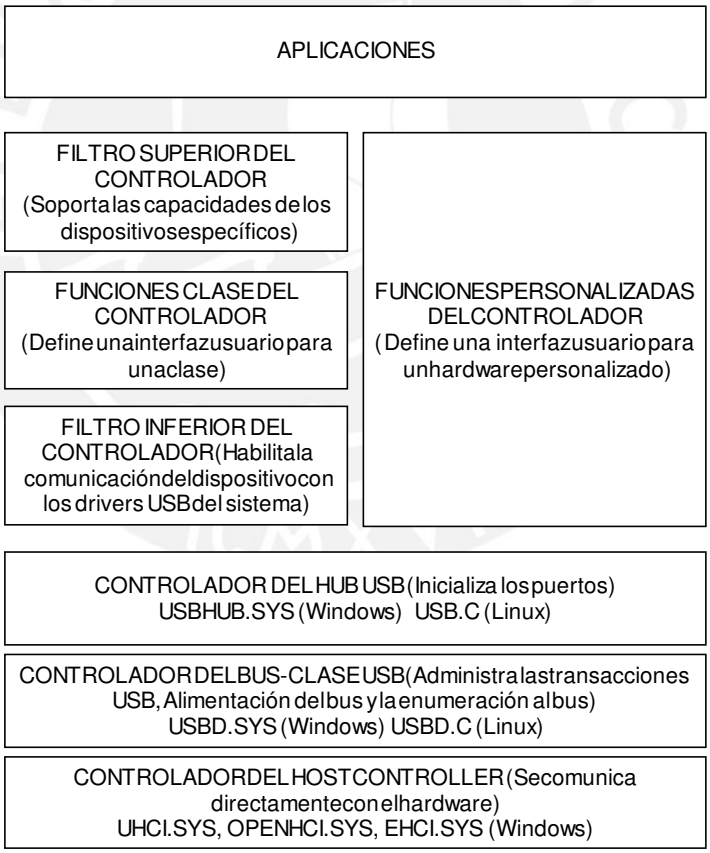


Figura 2.3.2.- Comunicaciones realizadas por el controlador

Que van del Host controller, controlador clase, controlador del hub y un controlador de funciones que puede tener más de un archivo.

Normalmente se habla de un controlador del dispositivo, pero en el fondo éste no trabaja por sí sólo pues en el fondo tiene que complementarse con el controlador de funciones y el controlador de bus para trabajar como uno solo.

2.3.1. CONTROLADOR PARA DISPOSITIVOS DE INTERFAZ HUMANO

Los dispositivos de interfaz humano o su sigla en inglés HID son los primeros que soporta el sistema operativo, por esta razón un dispositivo USB que pertenezca a esta clase son normalmente los más fáciles de implementar, siempre y cuando cumplan con las especificaciones que se requieran para pertenecer a esta clase.

La simple designación “interfaz humano” sugiere que el dispositivo deba interactuar directamente con las personas ya sea este un teclado, ratón, joystick, o algún otro de este tipo. Pero un dispositivo HID no tiene que ser necesariamente un interfaz humano, como se mencionó anteriormente sólo necesitamos que cumpla con las especificaciones dadas a continuación, lo que también lo restringe en algunas cosas.

- El intercambio de datos se da en estructuras llamadas reportes. El firmware del dispositivo debe soportar el formato de un reporte HID el cual es flexible y puede manejar cualquier tipo de información. El host usa las transferencias de tipo control y de interrupción para enviar y recibir información y pedir reportes.

- Cada transacción está limitada en la cantidad de información que pueda transferir, de acuerdo al tipo de dispositivo el máximo es de:

* 8 bytes para los de velocidad baja
* 64 bytes para los de velocidad media
* 1024 bytes para los de velocidad alta

- Un dispositivo puede enviar información a la computadora en cualquier momento, por ejemplo un clic de ratón, por lo que el host inspecciona al dispositivo periódicamente.

- La velocidad máxima de transferencia está limitada, especialmente en los dispositivos de media y baja velocidad. El host garantiza un máximo de:
* 800 bytes/segundo para los de velocidad baja
* 64000 bytes/segundo para los de velocidad media
* 24.576 bytes/segundo para los velocidad alta

- No se garantiza un taza de transferencia fija sobre todo en los dispositivos de baja velocidad.

2.3.2. CONTROLADOR EN WINDOWS

Para implementar cualquier manejador de hardware para el sistema operativo Windows 95/98/2000/XP Microsoft ha desarrollado la herramienta para el desarrollo de manejadores de nombre WinDDK (Windows Driver Developer Kit) el cual se puede solicitar desde su página web http://www.microsoft.com/winddk. Este software trabaja en conjunto con la herramienta de programación Visual Studio 6.0, en donde se puede trabajar con cualquiera de sus lenguajes de programación que nos ofrece, en nuestro caso Visual C++ pues la mayor cantidad de ejemplos e información que encontramos en la red referentes al desarrollo de un manejador USB estaban hechos en este lenguaje.

Para seguir con la implementación se debe tener en cuenta con qué versión de Windows se va a trabajar, porque dependerá de esto las librerías que se empleen del WinDDK.

La ventaja de usar el WinDDK es que nos da más opciones en cuanto a la diversidad de librerías que nos brinda y a su vez mayor facilidad en el desarrollo, en comparación a tener que partir casi desde cero en el caso de

hacerlo en Linux. Con esta ventaja se pueden desarrollar un manejador completamente personalizado con propio ID que no salte las librerías HID, tocando desde luego el ID del firmware del dispositivo. Este caso puede ser ventajosa por si queremos implementar un dispositivo que no tenga ningún conflicto con algún dispositivo HID, para lo cual basta con seguir el esqueleto del manejador, cargar las librerías necesarias, compilar el programa y probarlo, para lo último se cuentan con programas de evaluación del manejador como son el USB Driver Tester que viene incluido en el paquete del compilador CCS utilizado en nuestro firmware, o el USB Command Verifier del Forum de Implementación USB.

En el caso de que nuestro dispositivo se enumere como HID entonces no hará falta tener que implementar un manejador, pues ya viene incluido en el sistema operativo como lo vimos en el caso de linux; pero siempre hará falta tener el CD de instalación de Windows a la mano por si se requiera volver a instalar el manejador HID para un funcionamiento correcto.

2.4. LA RELACIÓN DE ONDA ESTACIONARIA.

La relación de onda estacionaria (ROE) en una línea de transmisión está relacionada matemáticamente a la relación entre la potencia de onda reflejada y directa, en general, cuanto mas grande es esta relación mayor es la ROE. También se cumple que cuando la ROE en una línea de transmisión es alta, las perdidas de potencia en dicha línea son mayores que las pérdidas que ocurren cuando la ROE es 1. Esta pérdida exagerada, conocida como pérdida de ROE, puede ser muy significante, especialmente cuando la ROE excede de 2 y la línea de transmisión tiene pérdidas significantes. Por esta razón nuestro objetivo es tratar de minimizar la ROE en las líneas de transmisión de comunicaciones. Una alta ROE también puede tener efectos indeseables, tales como el sobrecalentamiento o incluso la ruptura del dieléctrico separador de los conductores en una línea de transmisión y sobre todo la sobre exigencia al equipo transmisor que en muchos casos llegan a deteriorarlo, es aquí en donde se trabajará en la presente tesis.

En algunas situaciones, tales como aquellas encontradas en frecuencias relativamente bajas de RF, a niveles de potencia también bajas, y cortos tramos de líneas de transmisión de baja pérdida, un moderado valor elevado de ROE no produce una pérdida significativa de potencia o sobrecalentamiento en la línea, y por lo tanto puede ser tolerado.

La onda armónica:

$$V(z) = V_1(z) \times e^{-yz} + V_2(z) \times e^{yz} \quad \text{.....(2.4.1)}$$

$$I(z) = \frac{V_1(z) \times e^{-yz} + V_2(z) \times e^{yz}}{Z_0} \quad \text{.....(2.4.2)}$$

Donde Zo es la impedancia característica y V_1 y V_2 se determinan por medio de condiciones de borde.

De la definición de ROE se deduce la siguiente relación:

$$ROE = \frac{|V(d)|_{\max}}{|V(d)|_{\min}} \quad \text{..... ..(2.4.3)}$$

Donde $|V(d)|_{\max}$ es la magnitud del voltaje en un máximo sobre el patrón de onda estacionaria, y $|V(d)|_{\min}$ es la magnitud de voltaje en un mínimo.

Para un circuito de muestreo por inducción con circuito de rectificación la relación se reduce a:

$$ROE = \frac{\Gamma_D \cdot V_D - \Gamma_I . V_I}{\Gamma_D V_D + \Gamma_I V_I} \quad \text{.....(2.4.4)}$$

Donde V_D es el voltaje rectificado de la onda directa inducida y V_I es el voltaje rectificado de la onda inversa inducida, Γ_D y Γ_I son las características físicas del circuito.

Asumiendo que el sensor es totalmente simétrico, es decir $\Gamma_D = \Gamma_I$ podemos despejarlo tanto del numerador como del denominador lo que nos quedaría:

$$ROE = \frac{V_D - V_I}{V_D + V_I} \quad \text{.....(2.4.5)}$$

Ésta es la fórmula con la que se trabajará y se implementará en el programa final de decodificación del ROE, cualquier no linearidad, o deformación mínima producida por los parámetros Γ , será normalizado por software.

3. SOFTWARE DEL SISTEMA

El circuito desarrollado no necesita una velocidad mayor a 20Kbps para la transmisión de datos, es por eso que cualquier conexión convencional, a través del puerto serial o el paralelo, sería útil; pero en vista a que estos últimos han ido quedando relegados y más aún si hablamos de computadoras portátiles en donde ya se usa el puerto USB para la mayoría de sus periféricos.

Para cualquier proyecto de nuestra clase es importante contar con al menos 2 criterios de selección del puerto de comunicaciones. La velocidad requerida para lograr una comunicación efectiva entre el dispositivo externo y la computadora; y la disponibilidad del puerto de así no interferiríamos con otras tares que tenga que realizar la computadora en otros dispositivos periféricos. Existen también otros criterios a tomarse en cuenta tales como el costo y la factibilidad o complejidad de la implementación, es por esta última razón por la que mayormente se escoge el puerto serial y paralelo; pero como veremos mas adelante sólo dependerá de las herramientas con las cuales contemos la que harán mas accesible el manejo del puerto USB.

3.1. FIRMWARE DEL MICROCONTROLADOR

El microcontrolador que se usará es el **PIC16C745/JW**, cuyas características más resaltantes son:

- Procesador de arquitectura RISC de 35 instrucciones máquina
- 8 Kbytes de EPROM (Memoria de programa), 256 bytes de RAM (Memoria de variables).
- 22 pines o puertos de Entrada/Salida digital, 5 canales de entrada al conversor analógico a digital A/D.
- Oscilador de cristal de hasta 20MHz, (Para el uso de USB se recomienda un cristal de 6MHz).
- Puerto USB compatible con las especificaciones de USB 1.1

Un modelo hermano de este chip es el PIC16C765/JW con características similares sólo que con más puertos de salida tanto analógicos como digitales.

> ***NOTA:***
> ***Se debe tener muy en cuenta que este modelo tiene un PLL interno el cual multiplica el valor de frecuencia del cristal por 4, es decir que el cristal de 6MHz recomendado para el módulo USB, estará haciendo funcionar a todo el sistema a 24MHz.***

El Diagrama 3.1.1 muestra las diferentes entradas y salidas que tiene placa, así como los dos bloques internos, la parte de audio y los sub-bloques dentro del microcontrolador.

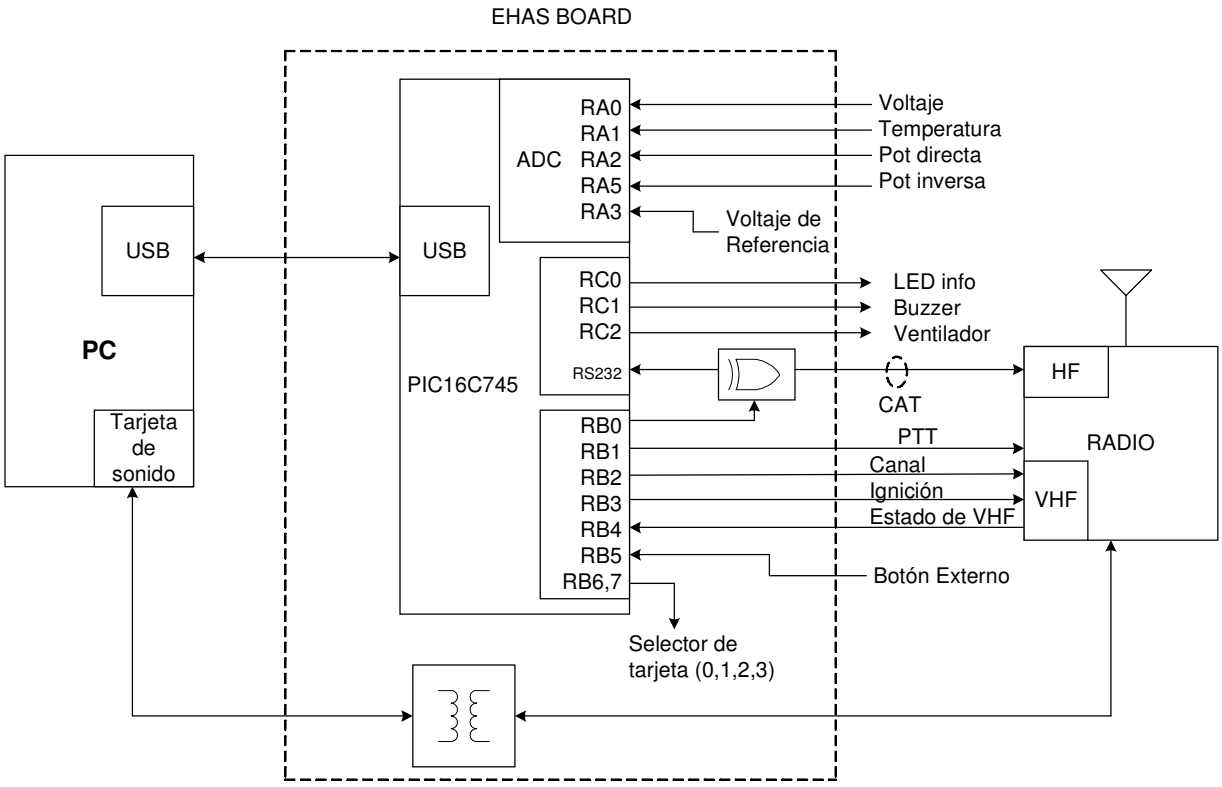


Diagrama 3.1.1 Esquema general del sistema

Se muestran tres bloques bien diferenciados: La computadora, la interfaz USB (EHAS BOARD) y la radio; las cuales tienen a grandes rasgos las siguientes funciones:

Computadora.- Tiene dos bloques internos: Transceptor de audio mediante la tarjeta de sonido y comunicación con el interfaz vía USB.

Interfaz USB.- Con dos funciones: Adquisición de señales analógicos(Voltaje, Temperatura, SWR) y manejo de la radio.

Radio.- Se encarga de enviar las señales de audio al espacio.

El firmware se separa en 2 partes, una que se encargue del manejo de la interfaz USB, que es la más compleja, y la otra de aplicación específica que se encargue de la adquisición de datos y las funciones de entrada salida del microcontrolador.

3.1.1. FIRMWARE DE TRANSFERENCIA USB

En la implementación del firmware que se encarga de realizar la transferencia USB con el host de la computadora se sigue el mismo formato que se explicó en la parte teórica para este tipo de transferencia. Para la presente aplicación el firmware tendrá todas las características de una transferencia HID pero se evitará tocar el controlador HID del sistema operativo.

El punto más importante en esta parte está en la tablas de descriptores en donde se encuentra el ID y el número de dispositivo y las características de transferencia descritas en la parte teórica, acá es donde “tomamos prestado” un ID, en este caso el 461 perteneciente a PRIMAX ELECTRONIC, y con el número de producto podremos tener más de un dispositivo usando el mismo ID.

Hay dos maneras de implementar el firmware de transferencia, la primera es hacerlo solamente en lenguaje ensamblador y la segunda es utilizando un lenguaje de alto nivel con un respectivo compilador correspondiente al microcontrolador elegido.

Para implementar el firmware en lenguaje ensamblador, lo cual resulta más económico, se usan las herramientas que brinda la empresa Microchip, entre las cuales están su compilador para lenguaje ensamblador MPLAB y sobre todo los diversos ejemplos que se pueden encontrar en su página web. Basta con usar los archivos “usb_cha9.asm” y “descript.asm” estando en éste último las tablas de descripción del dispositivo, ambos archivos se deben agregar al programa principal “main.asm” en donde ya se podrán usar las rutinas de transferencia como simples llamados de rutina “call send” y “call read” usados por el firmware de aplicación.

En el caso de elegir un compilador de lenguaje de alto nivel hemos visto por conveniente usar el programa CCS PIC C Compiler con un costo de $ 430 dólares en el mercado, que es un compilador de lenguaje C para una gran variedad de familias de microcontroladores de la empresa Microchip, pero siendo una de sus ventajas más importantes el que ya cuenta con las funciones de transferencia USB implementadas, lo cual por supuesto ahorra mucho trabajo y hace mucho más legible el código, teniendo las funciones de transferencia USB como simples comandos de “send_usb” y “read_usb”. Su tabla de descriptores se puede modificar en el archivo “descript.h” con las características antes mencionadas.

Para asegurarnos que el firmware de transferencia usb está bien implementado se puede ver si al menos se está enumerando en el bus USB haciendo las conexiones básicas de los pines D+, D-, Vcc y GND, lo conectamos al puerto USB y debemos ver lo siguiente:

En Windows debe activarse una ventana en donde avise del nuevo hardware encontrado y el sistema empieza la búsqueda del controlador, si no lo encuentra sale la siguiente ventana en donde se deben seguir los pasos para instalar el controlador deseado, mostrados en la figura 3.1.1.1.

Figura 3.1.1.1 Reconocimiento de nuevo hardware

En Linux se usa el siguiente comando:

tail -f /var/log/messages

y a continuación se muestra en pantalla

Aug 6 21:55:38 localhost kernel: hub.c: new USB device 00:10.0-2, assigned address 12
Aug 6 21:55:38 localhost kernel: usb.c: USB device 12 (vend/prod 0x461/0x20) is not claimed by any active driver.

Allí se ve aparece un dispositivo con ID 461 y número de dispositivo 20 que no tiene controlador.

3.1.2. FIRMWARE DE APLICACIÓN ESPECÍFICA

En esta parte se implementan las funciones adicionales que deba realizar el microcontrolador para controlar la estación, aprovechando que el microcontrolador cuenta además con puertos de entrada/salida digitales y analógicos.

Se describe el programa en el diagrama 3.1.2.1.

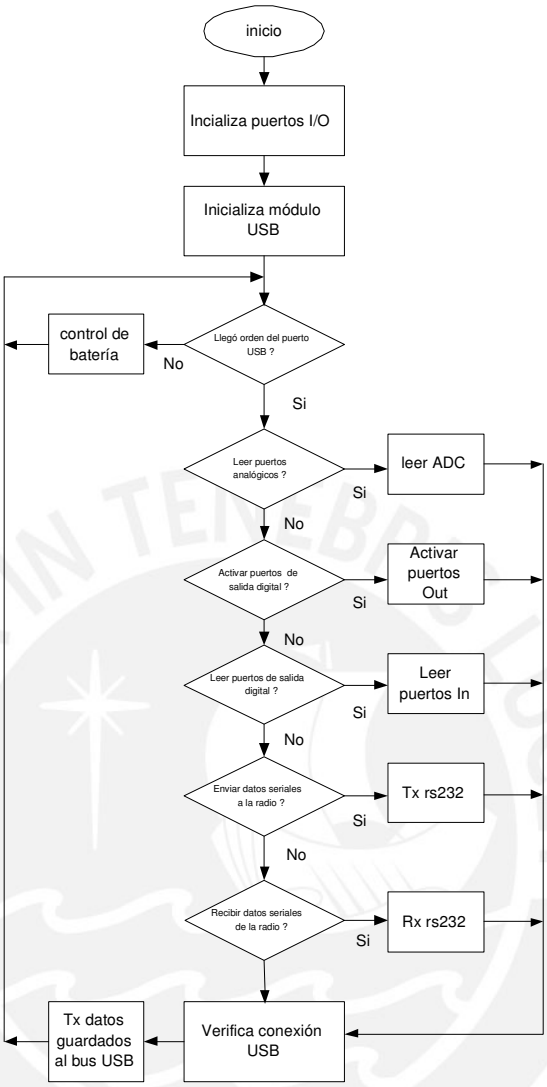


Diagrama 3.1.2.1 Firmware del microcontrolador

Para la parte analógica se ha implementado un filtro de software el cual, si bien disminuye la velocidad en la conversión, nos da mayor precisión a la hora de arrojar el resultado leído y esto se logra haciendo un promedio de lecturas en un intervalo de tiempo razonable (Aproximadamente 2mS).

$$ValorFinal = \frac{\sum 16 \cdot Valores \cdot Leídos}{16}$$

Puesto que el voltaje, la temperatura y la potencia no requieren tanta velocidad de muestreo, entonces se justifica este retraso aumentando la precisión de lectura.

3.2. PSEUDO-CONTROLADOR DEL DISPOSITIVO USB PARA LINUX

Lo hemos denominado pseudo-controlador porque no tendrá la estructura propiamente dicha de un controlador de linux, el cual es similar a Windows, en donde hay un archivo de información del producto que debe ser agregado al directorio de módulos que lista los controladores. Básicamente se ha implementado un programa que toma directamente comandos API pues en linux se trabaja en modo kernel.

Dentro de todas las librerías existentes en linux para manejar al puerto USB escogemos “libusb” en donde se encuentran los comandos API con los que podremos acceder a nuestro dispositivo. Se rellenan la tablas de los buses, las tablas de dispositivo, y se empieza buscar entre los dispositivos encontrados hasta que coincidan los descriptores, a partir de este momento ya tenemos control sobre nuestro dispositivo y se procede con el resto de las acciones.

Viendo el ID y el número de producto tenemos al dispositivo seleccionado, como se había implementado en el firmware, podemos tener más de un dispositivo usando el mismo ID cambiando simplemente el número de producto (en nuestro caso tendremos hasta cuatro dispositivos).

Una vez enumerado y reconocido el dispositivo ya podemos comunicarnos con el dispositivo usando simplemente los comandos de transferencia “*usb_interrupt_write*” para enviar datos y “*usb_interrupt_read*” para leer datos.

Para que coincida con lo implementado en el firmware los paquetes de datos a enviar tanto de lectura como de escritura son de 4 bytes.

Ya con esto se pueden crear comandos un nivel más elevados como: set, unset, analog, reset y txserial, todos ellos contenidos en el pseudo-controlador "**eboard**", con los que se pueden acceder a cada una de los puertos configurados en el dispositivo; explicados en el capítulo siguiente.

3.3. CONTROLADOR DEL DISPOSITIVO PARA WINDOWS

Se usará un controlador de aplicación específica el cual se implementa siguiendo las especificaciones que solicita el Windows DDK, rellenando las tablas de descriptores, el ID y número de dispositivo para finalmente crear un archivo con extensión ".sys" ejemplo "testusb.sys" y su respectivo archivo de información ".inf", en nuestro caso "himcansaya.inf". De allí en adelante bastará con seguir los pasos que pide Windows para la instalación del controlador.

Al conectar el dispositivo saldrá la ventana de la figura 3.3.1.

Figura 3.3.1 Reconocimiento de hardware por Windows

Los datos que salen allí los lee directamente del dispositivo, a continuación pide que se seleccione el controlador apropiado mostrado en la figura 3.3.2.

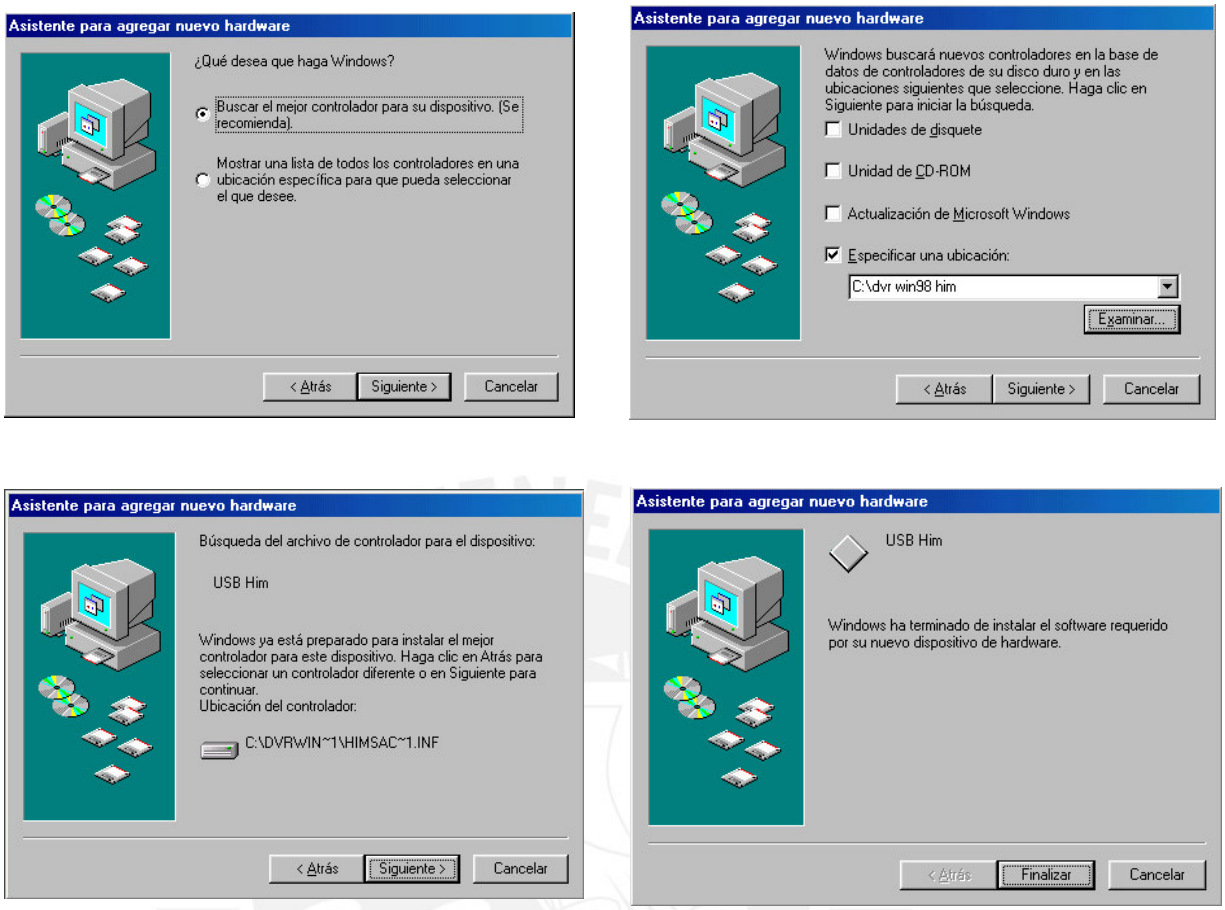


Figura 3.3.2 Instalación del driver

Se encuentra el archivo de información “USB Him” que solicita el dispositivo.

La instalación satisfactoria se puede apreciar enla figura 3.3.3

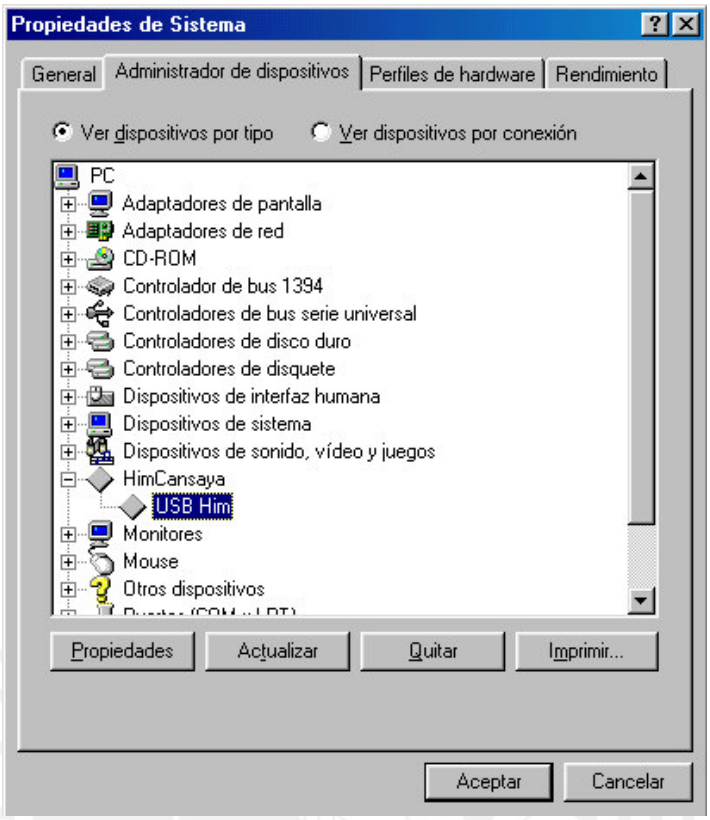


Figura 3.3.3. Vista del controlador instalado de forma correcta

Una vez instalado el controlador, deberá aparecer en la lista del Administrador de dispositivos, si sale con un signo de interrogación quiere decir que el controlador no fue bien instalado y se deberá volver a instalar, pero antes se deberá borrar cualquier archivo de información correspondiente al dispositivo contenido en el directorio “/windows/inf”

Ya con el controlador instalado se puede hacer cualquier aplicación en lenguaje C llamando al dispositivo usando el comando “`HANDLE hdevice`” y para leer y escribir en el dispositivo “ read_device” y “write_device” respectivamente.

Una aplicación sencilla de escritura y lectura de puertos digitales y analógicos la podemos ver en el anexo “aplicación en windows” cuyos resultados vemos a continuación:

Para 4 leds conectados a los pines del puerto B: RB0 al RB3 y 4 canales activados del puerto A.

```
encendido (H) y apagado (L) de los 4 leds: HHHL

4 bytes fueron enviados al dispositivo.
bytes recibidos:

El valor de conversion es:
Canal 0: 0.000 Voltios (Voltaje de batería)
Canal 1: 26 grados (Temperatura de la radio)
Canal 2: 0.251 Voltios (Potencia directa)
Canal 3: 0.232 Voltios (Potencia reflejada)

ROE = 1.23

CNTL C  para salir

encendido (H) y apagado (L) de los 4 leds:
```

Figura 3.3.4 Pantalla de una consola DOS con las operaciones realizadas

3.4. SOFTWARE DE APLICACIÓN EN LINUX

El software de aplicación es el que se encuentra en la escala más elevada de toda la estructura del software del dispositivo y es con el que el usuario finalmente interactuará. Dicho software puede usar cualquiera de las funciones implementadas en el controlador.

La aplicación presentada a continuación se denomina “ehas-station” la cual, aparte de sus funciones de módem (en donde sólo interviene en el manejo del PTT), aplica directamente las funciones de la placa estación. Este paquete tiene como aplicación de configuración a “config-ehas” que explicaremos con más detalle en el capítulo siguiente, para ello se vale del paquete denominado “ehas-board” en donde están implementadas las siguientes funciones:

- Control del ventilador por histéresis (Configurables en config-ehas)
- Control de transmisión serial para radios HF (Configurable en config-ehas)
- Habilitación de alarmas por lecturas de los valores analógicos.

- Control del PTT de la radio.
- Visualización de mensajes de advertencia.

Todas estas funciones se valen del controlador “eboard.c” (Incluido en el anexo) descrito anteriormente el cual también va dentro de ehas-board.

Como nuestro dispositivo actuará de forma pasiva, será la aplicación la encargada de ordenarle el envío de datos y esto se da implementado un demonio denominado “edaemon” que quedará activo en el sistema exclusivamente para tener en comunicación al dispositivo. De esta manera se tienen los valores analógicos del dispositivo muestreados constantemente.

Las acciones de control del dispositivo tales como el control del ventilador, el control de la transmisión serial con las radios HF y el control de voltajes de advertencia están a cargo de la computadora, pero el control de voltaje de corte está a cargo del dispositivo, eso se detalla en el siguiente capítulo.

Finalmente las operaciones que realice la aplicación serán invisibles para el usuario, lo único que podrá ver son los siguientes indicadores:

Figura 3.4.1 : Vista de los íconos de sensado

4. HARDWARE DEL SISTEMA

Para que el microcontrolador trabaje con su entorno a las características eléctricas adecuadas, necesita de ciertos acondicionamientos en sus puertos de entrada y salida tanto digitales como analógicos, los cuales se ven a continuación.

4.1. HARDWARE DE ADQUISION DE DATOS

En la parte de adquisición de datos se usa el conversor analógico a digital embebido al microcontrolador que tiene las siguientes características.

- Ocho bits de muestreo, lo que nos permite aproximaciones de $2^8 = 256$ escalas de voltaje, que junto con el rango de voltaje de entrada nos permite determinar la resolución del valor sensado.
- Cinco canales de entrada al conversor A/D para el modelo PIC16C745. En nuestro caso sólo usaremos 4 canales pues uno lo usaremos para la entrada de referencia para tener una mayor precisión en nuestra lectura.
- Registros especialmente implementados dentro de las variables del microcontrolador para el manejo del conversor A/D. Con esto el firmware se hace más ligero.

4.1.1. VOLTAJE DE REFERENCIA

El siguiente diagrama muestra la entrada de referencia a 4V.

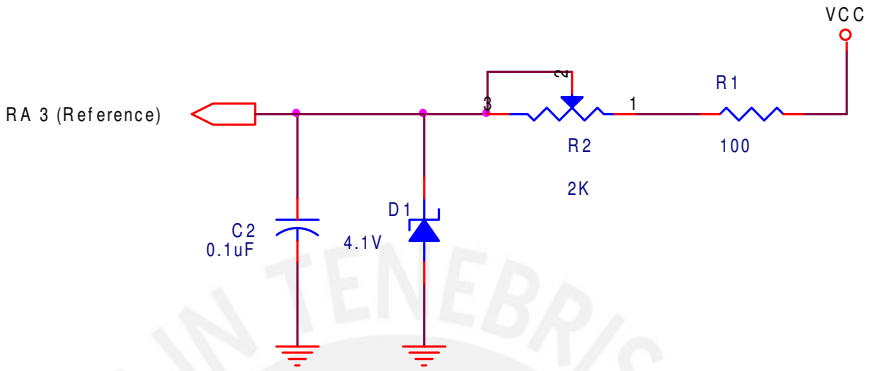


Figura 4.1.1 Circuito de calibración del voltaje del diodo zener

El circuito contiene un diodo zener de 4.1V con ±5% de error, que tiene una resistencia de 100Ωen serie con un potenciómetro de 2 kΩ de calibración (la variación será entonces de 100Ω a 2kΩ), los cuales en conjunto funcionan como la resistencia de polarización del diodo cuyo valor nominal varía dependiendo de la temperatura ambiental y de la carga que alimente, como en nuestro caso la entrada al canal A/D de referencia tiene un valor alto de impedancia, la variación de corriente de carga no varía, dejando cómo únicos parámetros de variación a la temperatura y al potenciómetro. Teniendo el control del valor nominal de voltaje dejaremos que la corriente de polarización del diodo zener lleve el voltaje nominal a 4V con aproximadamente 1mA de corriente de polarización, para que de esta manera también facilite los cálculos que se harán en el software. El condensador es de protección de ruido.

4.1.2. SENSADO DE VOLTAJE

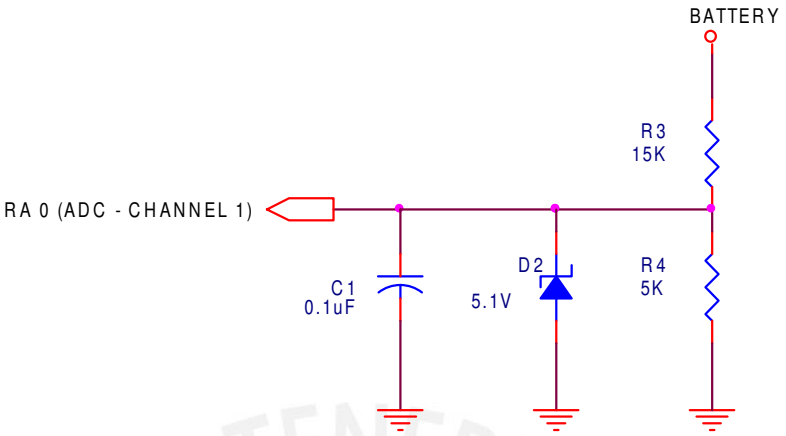


Figura 4.1.2 Circuito de sensado de voltaje

En la figura 4.4.2.1 se ve el circuito de sensado de voltaje que adapta la señal de entrada a la entrada del puerto analógico del microcontrolador, mediante un divisor resistivo que divide en un factor de 4 a la entrada de voltaje de batería, teniendo al voltaje de referencia igual a 4V, entonces el voltaje máximo de batería que podríamos medir es de 16V, si este valor se excediera se tiene un diodo zener de 5.1V como protección de sobrevoltaje.

4.1.3. SENSADO DE TEMPERATURA

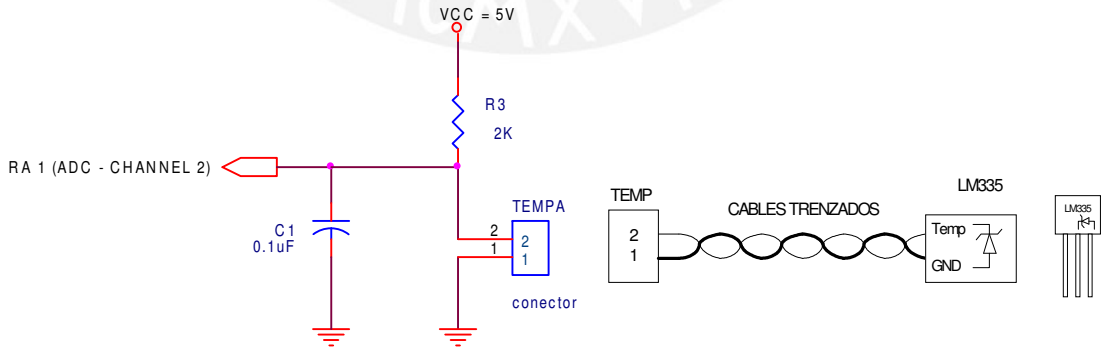


Figura 4.1.3.1 Circuito de sensado de temperatura

Para el sensado de temperatura usamos el circuito integrado LM335 cuya precisión es suficiente para nuestra aplicación, el cual tiene las siguientes características:

- Rango de sensado de -40 a 100°C.
- Corriente de funcionamiento = 10mA
- Valor de referencia: 3V @ 20 °C
- 1°C = 10 mV

Para tenerlo en su corriente de funcionamiento de 1mA usamos la resistencia de 2Kohm, y de los datos del circuito podemos sacar la resolución de temperatura que podremos medir:

Resolución = 1.56 Grados Celsius por muestra

Se debe tener en cuenta que el cable no debe sobrepasar los 2 metros y se recomienda usar par trenzado o un cable apantallado como usan los cables de audio para evitar problemas de ruido.

4.2. SENSOR DE ROE

Existen muchas clases de transductor de ROE, los cuales difieren solamente en algunas características como la forma en que capturan la señal, pero en general cuentan con una etapa rectificadora tanto en dirección de la señal como en dirección contraria, pues es esta relación la que nos interesa. Se han usado dos modelos los que se presentan a continuación.

4.2.1. TRANSDUCTOR DE BARRAS PARALELAS

Consiste en una caja metálica con tres terminales, una salida DC para la potencia directa, otra para la inversa y una de referencia, en nuestro caso será la misma masa, adentro contiene dos barras de inducción paralelas al núcleo desnudo, éstas señales inducidas pasan por un circuito rectificador que son básicamente un diodo de cristal y un condensador, la rectificación se da de

manera que una vaya en dirección de la potencia directa y en la otra barra en dirección de la potencia inversa, estos valores de DC ya pueden ser digitalizados, ya que a 100W genera una DC de 1.44V aprox para la barra de potencia directa, claro que al final se tiene que calibrar por software comparando con un instrumento de precisión como el BIRD.

Este circuito es conveniente para los rangos menores a 40MHz que es ideal para las radios HF, en otras frecuencias el acoplamiento es muy bajo, por menor a los –14dB.

El esquema se presenta a continuación:

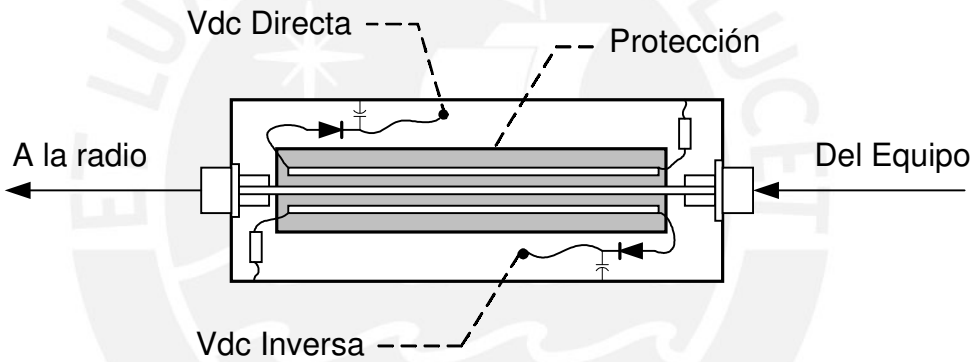


Figura 4.2.1.1 Sensor de ROE de barras paralelas

4.2.2. TRANSDUCTOR EN IMPRESO

El siguiente circuito es óptimo para los rangos de frecuencia que usamos en VHF los cálculos se basan en la fórmula 2.4.5 y para que su funcionamiento sea óptimo se debe mantener el máximo de simetría para ambas tomas de inducción.

La resistencia de 50 ohm se compone de tres resistencias de 150 ohm en paralelo de tipo SMD (de montaje superficial de potencia)

El diagrama se muestra a continuación:

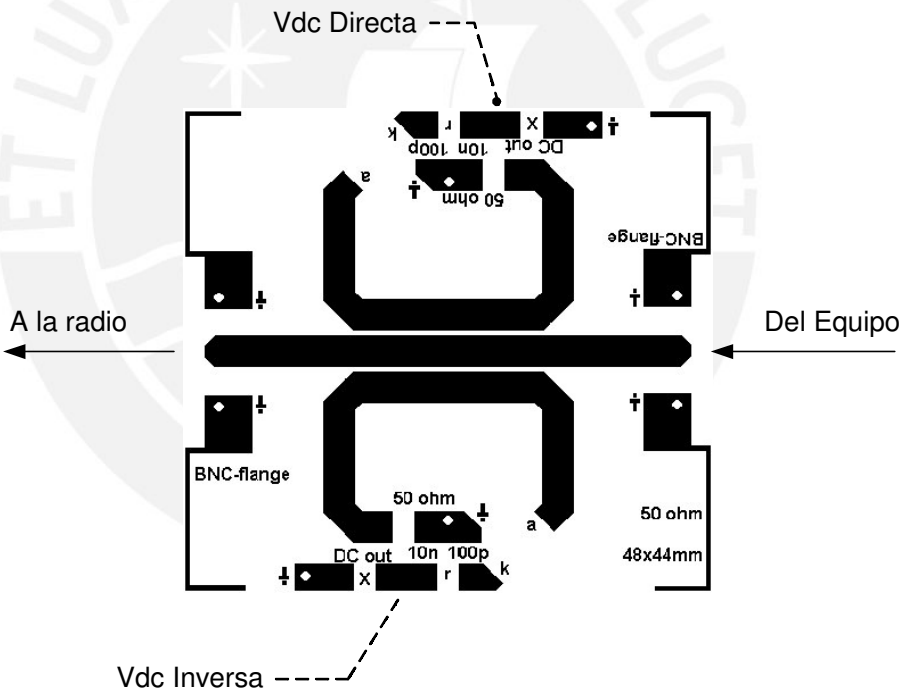


Figura 4.2.2.1 Sensor de ROE en circuito impreso

Los rangos de voltaje a la salida para el rango óptimo varían dependiendo a la frecuencia y la potencia que se transmita y se muestra en la tabla 4.2.2.1 en donde el puerto A es Vdc inversa y el puerto B es Vdc directa.

Puerto A/B	25W	2,5W	0.25W
145MHz	Voltios	Voltios	Voltios
A directa	1,91	0,520	0,070
B directa	1,90	0,52	0,065
A reflejada	0,039	0,0038	0,0001
435MHz			
A directa	5,28	1,475	0,33
B reflajada	0,300	0,0485	0,0043
A reflejada	0,361	0,057	0,005

Tabla 4.2.2.1 Variación de valores de sensado a diferentes potencias

En base a este cuadro se genera la ecuación de compensación que finalmente se opera en nuestra fórmula que hallará el ROE descrita en la parte teórica.

4.3. HARDWARE DE LOS PUERTOS DIGITALES

4.3.1. INTERFAZ SERIAL CON LAS RADIOS

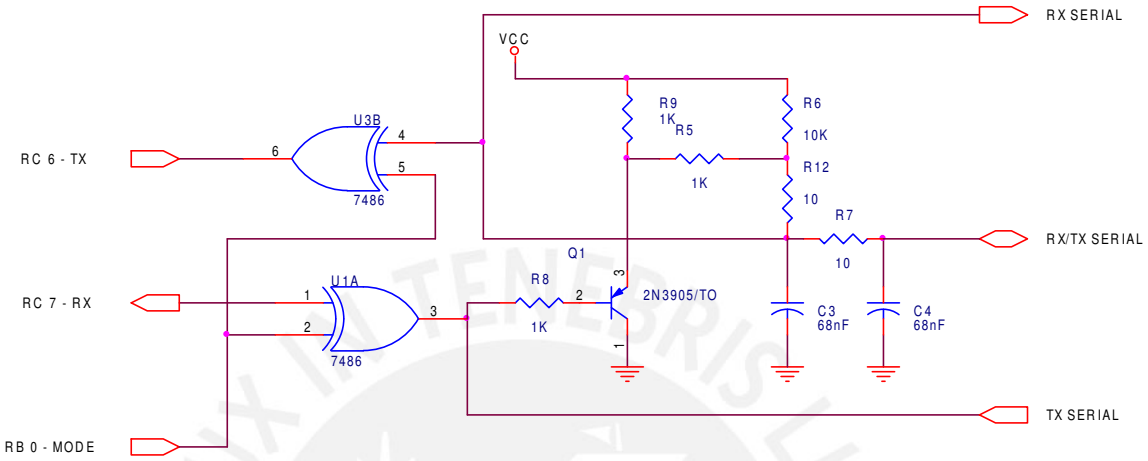


Figura 4.3.1.1 Circuito anticolisión con transistores

En este diagrama se muestra el circuito anticolisión de la interfaz serial del microcontrolador a la radio ICOM el cual usa una sola línea tanto para la transmisión como para la recepción. El transistor hace de interruptor para la recepción de la trama serial y conmutador para la salida.

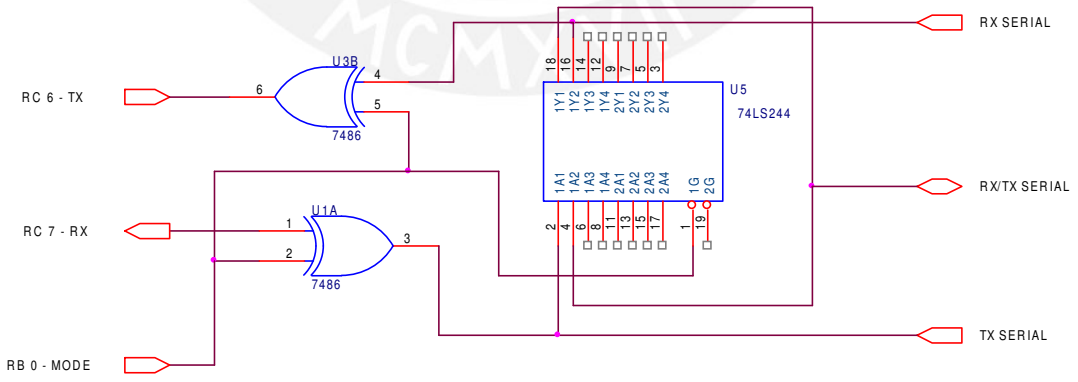


Figura 4.3.1.2 Circuito anticolisión con buffer triestado

Esta es otra forma de implementar un circuito anticolisión para interfaces que usen una sola línea de transmisión utilizando un buffer 74LS244.

4.3.2. ESTADO DE LA RADIO VHF

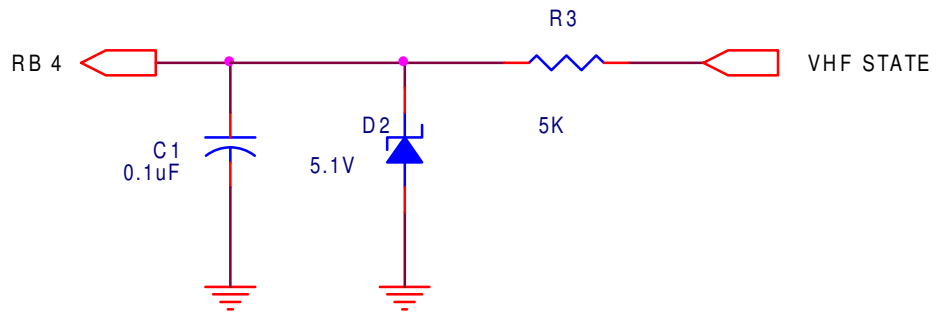


Figura 4.3.2.1 Adaptación de la entrada a valor TTL

La salida que nos indica el estado de la radio VHF (encendido o apagado) se lee usando el puerto RB4 del microcontrolador, pero este valor leído no es TTL sino que supera los 10V lo cual se limitan con el circuito mostrado en la figura 4.3.2.1

4.3.3. ZUMBADOR

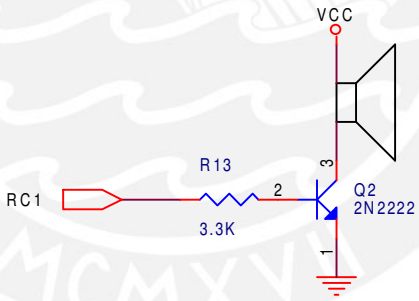


Figura 4.3.3.1 Circuito de activación del zumbador

Se tiene también un zumbador como se muestra en la figura para dar los avisos de alerta al usuario, dicho zumbador funciona con una señal cuadrada que puede variar de 1.5 a 4KHz, siendo 2KHz la frecuencia óptima, dicho zumbador es controlado por el puerto RC1 del microcontrolador el cual cuenta con una salida generadora de señal cuadrada ya implementada y cuya frecuencia es fácilmente programable. El circuito consta de un transistor como interruptor que será el encargado de entregar la señal cuadrada al zumbador.

4.3.4. VENTILADOR

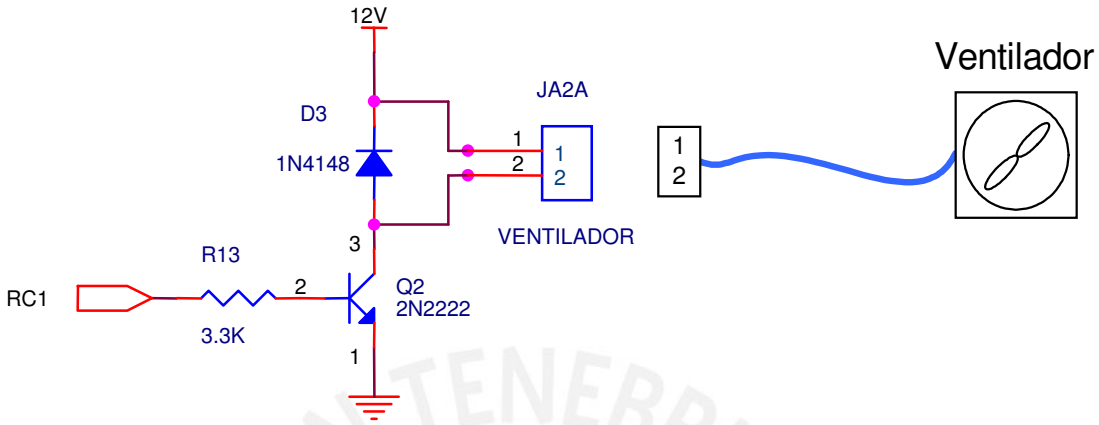


Figura 4.3.4.1 Circuito de manejo del ventilador

El circuito adapta la salida digital del puerto del microcontrolador con 1.5mA para manejar entre 100 y 200mA a 12V que se requiere para accionar al ventilador.

4.4. PROTECCIÓN Y ADAPTACIÓN DE IMPEDANCIAS A LA TARJETA DE SONIDO

La entrada de audio a la computadora debe estar protegida ya que el sistema de radio es propenso a sufrir descargas atmosféricas y si los protectores de línea, ya sean de gas o de otra clase, reaccionan con cierto retardo, se corre el riesgo dañar algo más que la entrada de sonido de nuestra computadora. Para esto se ha previsto acondicionar unos transformadores de banda base (Voz) para aislar la conexión externa, que incluye el sistema de radio junto con la placa estación, de la conexión interna, que sería la computadora y todo los periféricos que tenga conectados; además la conexión de audio va con un adaptador de impedancias para acondicionar los niveles de entrada y salida de audio que van tanto entrada como de salida de la radio a la tarjeta de sonido de la computadora. El diagrama se muestra en la figura 4.4.1

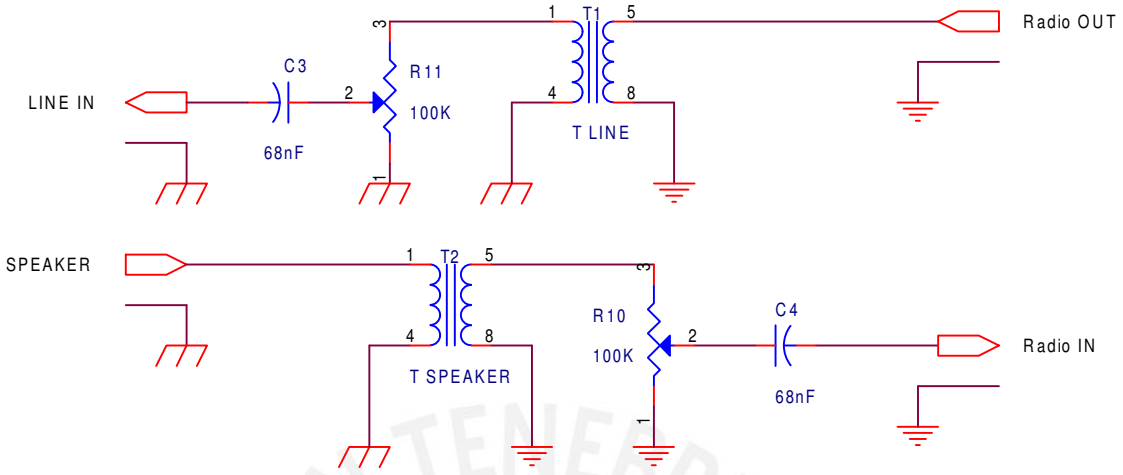


Figura 4.4.1 Protección de la tarjeta de sonido

En caso se usara la entrada de micrófono en vez de la entrada de línea, la mayoría tiene agregada a su entrada una salida de 2VDC aproximadamente para alimentar los micrófonos y lo cual eliminamos con el condensador C3. Algunas radios también tienen este nivel DC en sus entradas por eso se usa simétricamente el condensador C4

Los transformadores empleados no deben distorsionar demasiado la señal de audio porque eso dará lugar a errores en la transmisión de datos, en nuestro caso se ha usado un transformador con las siguientes características:

- Relación 1:1
- R = 45 Ohm
- L = 100 mH

4.5. ALIMENTACIÓN DE LA TARJETA

Una de las ventajas al usar el puerto USB es que se cuenta con alimentación propia del bus, siempre y cuando el dispositivo no exceda los 500mA de la fuente de 5V que nos brinda. En nuestro caso la tarjeta tendrá dos modalidades de alimentación, por fuente USB y por fuente externa, esta última para manejar

los accesorios de la placa estación que requieran mayor corriente que la permitida por el bus.

En la figura 4.5.1 se muestra la fuente externa usada, empleando el circuito integrado LM2576-5 P+, el cual es un regulador switching de 5V con muchas mejoras cualidades que los reguladores lineales tanto en el consumo como en rango de entrada de voltaje. Sus características más importantes son:

- Voltaje de salida: 5V DC estable con variación de ± 0.1V
- Voltaje de entrada: De 8 a 40V DC no regulada.
- Corriente máxima: Hasta 2A (también depende de la bobina y el diodo).
- Temperatura de operación: De –40 a 100 grados Celsius.

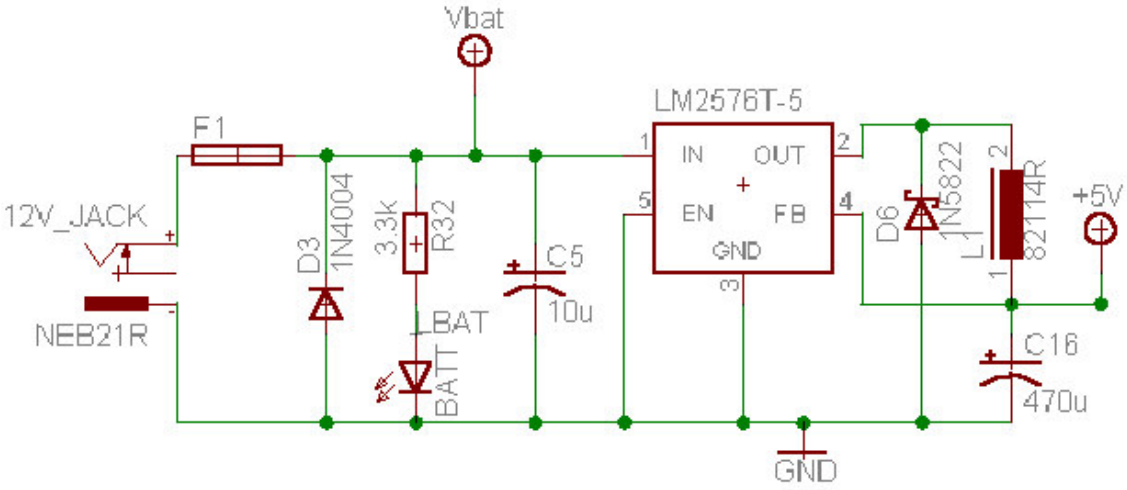


Figura 4.5.1 Circuito de alimentación

Como toda fuente switching, este circuito integrado debe venir acompañado por otros tres accesorios para su correcto funcionamiento, un condesador, un diodo y una bobina, el manual nos dice que para que el regulador opere a su máxima potencia debemos usar un diodo de 3A, una bobina de 100uH (con alambre calibre 20) y un condensador de 1000uF a 50V. En este caso no superamos 1A de corriente por lo que se emplea los siguientes:

- Diodo 1N5822 (Rápido de 2 A).
- Bobina de 100uH con alambre de calibre 24.
- Condensador de 100uF a 25V.

Estos cambios se hicieron para hacer más pequeña la tarjeta puesto que estos componentes al ser de menor potencia también son más pequeños físicamente.

El calibre del alambre de la bobina puede ser menor en nuestro caso, por lo que también se podría usar el número 25 o 26 envuelto en una ferrita cilíndrica que se acomode al circuito impreso, en el caso que no se pueda conseguir una comercial con el tamaño y las características deseadas.

4.6. FABRICACIÓN Y MONTAJE DEL CIRCUITO IMPRESO

Para terminar con el hardware, se mando fabricar el circuito impreso en una placa de fibra de vidrio de doble cara con máscara antisoldante y diagrama de distribución de componentes (adjuntos en los anexos) que facilitan el montaje de los componentes a la hora de ubicarlos en la tarjeta para soldarlos, aparte de estas especificaciones se mandó hacer también un proceso de metalizado para conectar ambas caras y evitar con esto el uso de pasantes o vías de conexión.

En la figura 4.6.1 se puede apreciar el circuito de la versión 1.3.

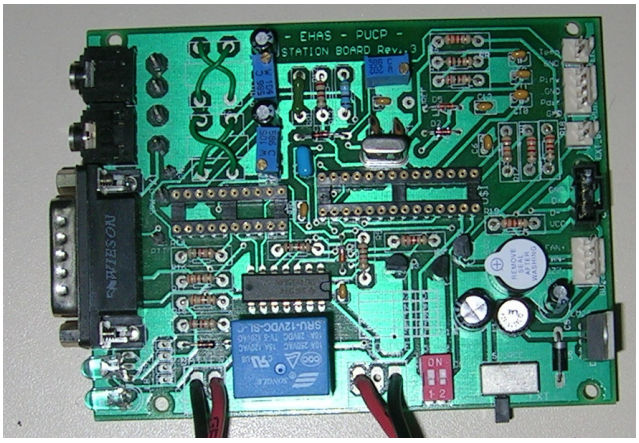


Figura 4.6.1- Vista del circuito final construido.

También se tiene un modelo usando tecnología de montaje superficial o SMD correspondiente a una versión posterior, con el cual se redujo en gran proporción el tamaño y los costos de fabricación de la Placa Estación, como se muestra en la figura 4.6.2

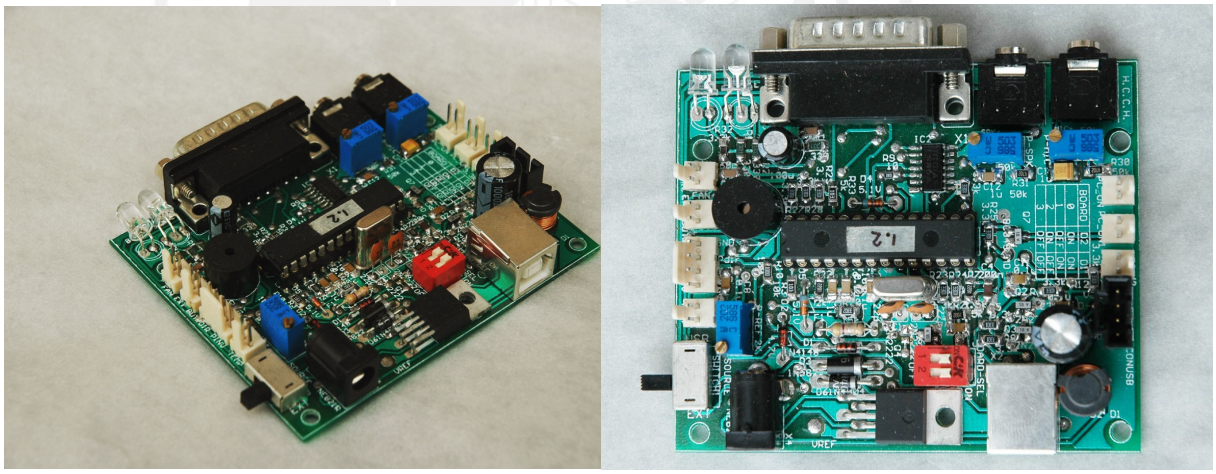


Figura 4.6.2 Vistas del modelo reducido

5. FUNCIONAMIENTO Y MANEJO

A continuación se presenta el funcionamiento y manejo de la placa estación para una estación HF y VHF.

Para tener un panorama de las capacidades de la placa estación listamos las funciones que se pueden realizar:

5.1. FUNCIONES PARA LA RADIO RADIO HF (KENWOOD TK-80)

FUNCIONES ELEMENTALES:
- Conexión de audio de la computadora a la radio
- Control del PTT

FUNCIONES DE CONTROL Y LECTURA DE LA RADIO
- Lectura y cambio de canal †
- Lectura y cambio de modo (LSB, USB, CW)
- Lectura y cambio de potencia de transmisión
- Lectura y cambio del silenciador (Squelch)
- Lectura y cambio del clarificador (Clarifier)
- Lectura y cambio de la ganancia (volumen de altavoz) †
- Control del encendido y apagado de la radio y lectura del estado

FUNCIONES DE MONITOREO EXTERNOS A LA RADIO
- Lectura de temperatura (Con indicador en el monitor)
- Lectura de voltaje del sistema (Al que esté conectada la interfaz)
- Lectura del ROE

ENTRADAS EXTERNAS
- Botón de emergencia para cortar transmisión de datos y usar voz

SALIDAS EXTERNAS

- Conexión para uso de ventilador externo de 12V
- Conexión para uso de buzzer externo

NOTA:

La radio ICOM IC-78 además de tener las funciones básicas tiene las que están con †

5.2. FUNCIONES PARA LA RADIO VHF (MOTOROLA PRO 3100)

FUNCIONES ELEMENTALES:

- Conexión de audio de la computadora a la radio
- Control del PTT

FUNCIONES DE CONTROL Y LECTURADE LA RADIO

- Lectura y cambio de canal (Dos estados)
- Control del encendido y apagado de la radio y lectura del estado

FUNCIONES DE MONITOREO EXTERNO A LA RADIO

- Lectura de temperatura (Con indicador en el monitor)
- Lectura de voltaje del sistema (Al que esté conectada la interfaz)
- Lectura del ROE

ENTRADAS EXTERNAS

- Botón de emergencia para cortar transmisión de datos y usar voz

SALIDAS EXTERNAS

- Conexión para uso de ventilador externo
- Conexión para uso de zumbador externo

5.3. PARÁMETROS DE FUNCIONAMIENTO

CARATERÍSTICAS ELÉCTRICAS:

PARÁMETRO	*MÍNIMO*	*TÍPICO*	*MÁXIMO*	
Voltaje de alimentación externa	**8**	**12**	**20**	**V**
Corriente de carga en el relay	**-**	**3**	**6**	**A**
Potencia consumida por la interfaz	**100**	**150**	**200**	**mW**
Temperatura de funcionamiento óptimo	**10**	**25**	**60**	°C
Voltaje en cualquier entrada de sensado	**0**	**-**	**4**	**V**
Corriente en el puerto del ventilador	**1**	**10**	**50**	**mA**
Corriente en el puerto del buzzer	**0.1**	**1**	**3**	**mA**

Tabla 5.3.1 Características eléctricas

RANGOS DE SENSADO:

PARÁMETRO	*MÍNIMO*	*TÍPICO*	*MÁXIMO*	
Sensado de temperatura	**-40**	**-**	**100**	°C
Sensado de voltaje	**0**	**-**	**16**	**V**
Sensado de ROE	**1**	**-**	**4**	

Tabla 5.3.1 Rangos de sensado

5.4. DESCRIPCIÓN

El Diagrama 5.4.1 muestra las conexiones en la tarjeta con la distribución real de elementos internos.

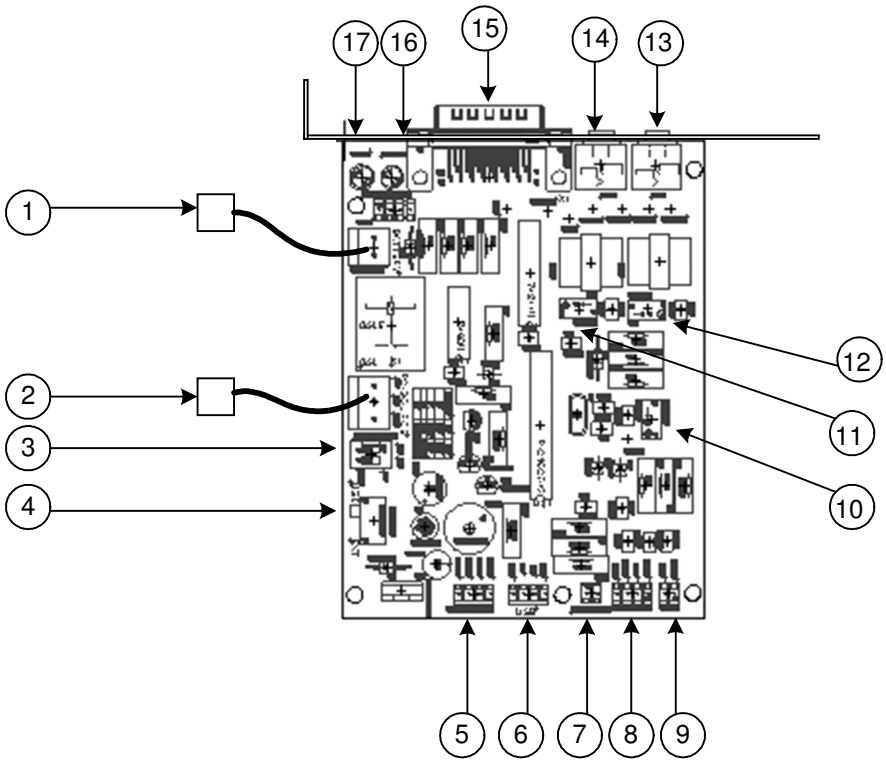


Diagrama 5.4.1 Descripción física del dispositivo

1. Alimentación externa.
2. Entrada de alimentación de la computadora para el control de encendido.
3. Switch selector de tarjeta.
4. Switch selector de tipo de alimentación.
5. Conexión al zumbador externo y al ventilador.
6. Entrada USB.
7. Conexión para la entrada del botón externo.
8. Conexión para sensado.de potencia directa (Pdir) y potencia inversa (Pinv).
9. Conexión para el sensado de temperatura.
10. Potenciómetro para calibrar **referencia** a 4V.
11. Potenciómetro para calibrar nivel de **salida (speaker)** de audio.
12. Potenciómetro para calibrar nivel de **entrada (line-in/mic)** de audio.
13. Conector Jack hembra de **salida (speaker)**.
14. Conector Jack hembra de **entrada (line-in/mic)**.

15. Conector DB25 Macho de salida de audio y control de las radios HF (Icom IC-78, Kenwood TK-80, Yaesu System 600) y VHF (Motorola Pro3100).
16. Led de configuración.
17. Led indicador de fuente externa.

5.5. MODALIDADES DE FUNCIONAMIENTO DE LA PLACA ESTACIÓN

A continuación se describe las modalidades de funcionamiento tanto para una transmisión HF como VHF.

5.5.1. MODALIDAD VHF

En la figura 5.5.1.1 se ve el la conexión de la radio VHF Motorola usando su puerto posterior de control, allí se encuentra la conexión de audio y los pines de control y lectura de estado.

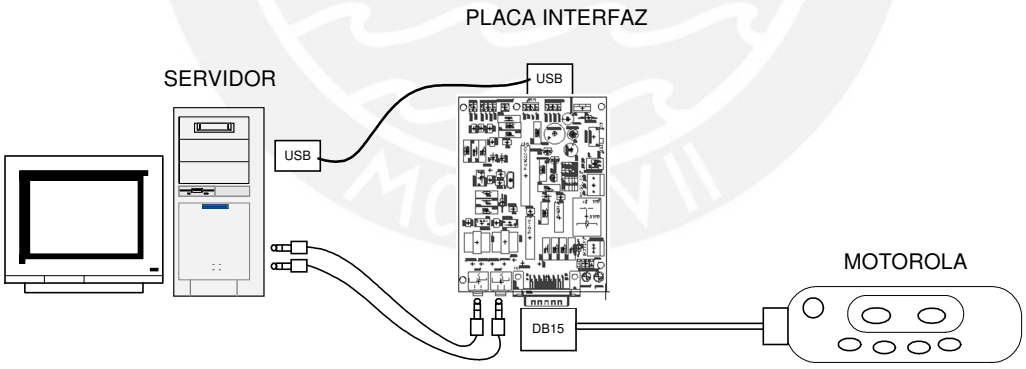


Figura 5.5.1.1 Modalidad VHF

5.5.2. MODALIDAD HF

En la figura 5.5.2.1 se ve la conexión de audio a la radio así como la conexión para el manejo serial de la radio (CAT).

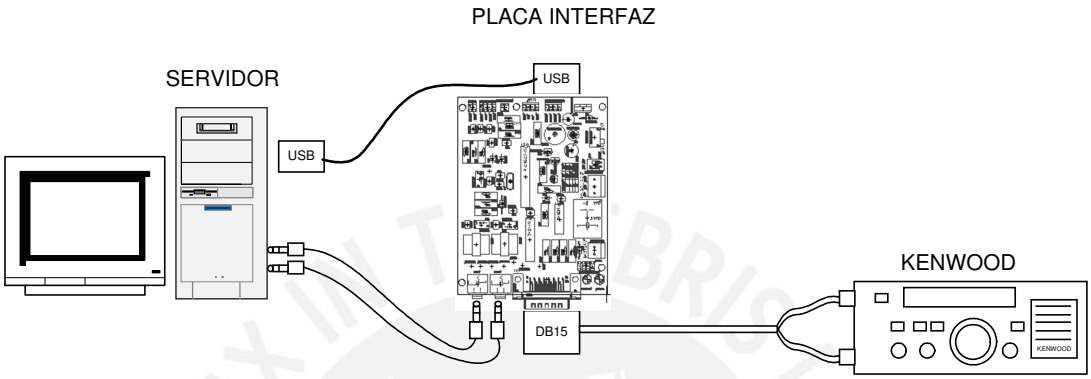


Figura 5.5.2.1 Modalidad HF

5.6. CABLEADO Y CONEXIONES DE LA PLACA.

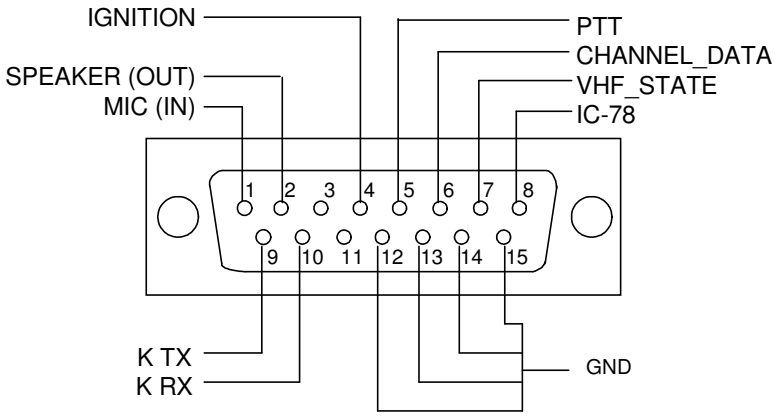


Figura 5.6 Vista frontal del conector DB15 macho del circuito impreso.

5.6.1. CABLE ICOM IC-78

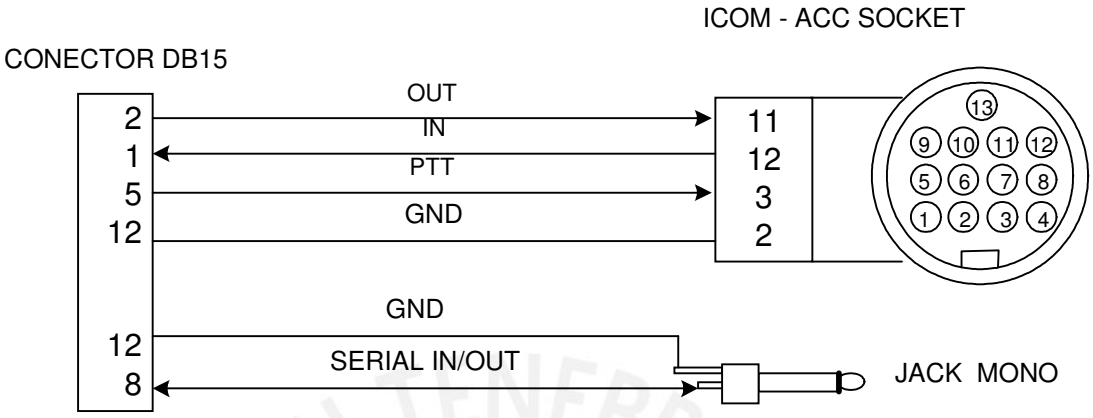


Figura 5.6.1.1 Conexión del dispositivo a la radio ICOM IC-78

5.6.2. CABLE MOTOROLA PRO3100

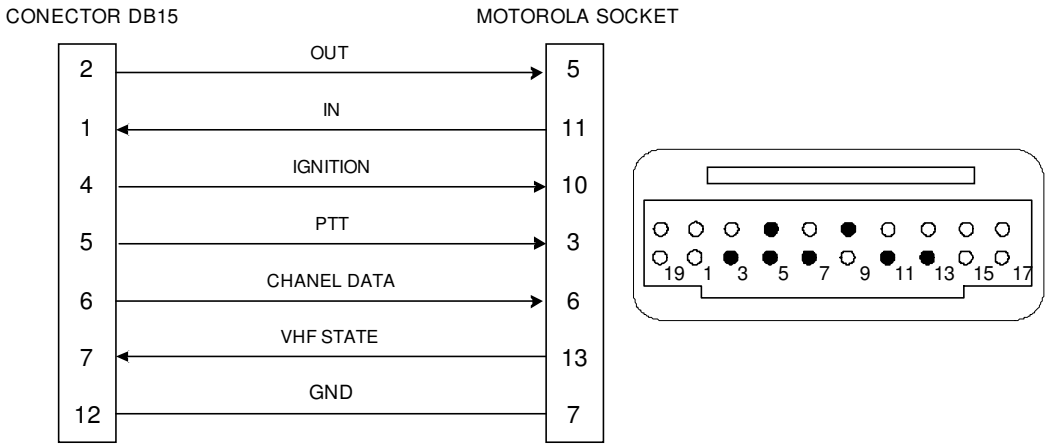


Figura 5.6.1.1 Conexión del dispositivo a la radio Motorota Pro 3100

5.6.3. CABLE KENWOOD TK-80

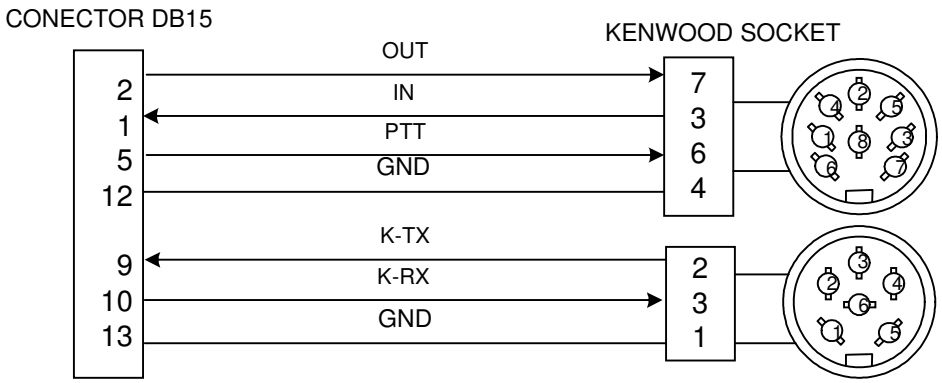


Figura 5.6.3.1 Conexión del dispositivo a la radio Kenwood TK-80

5.6.4. CONECTORES MOLEX DE SENSADO EXTERNO

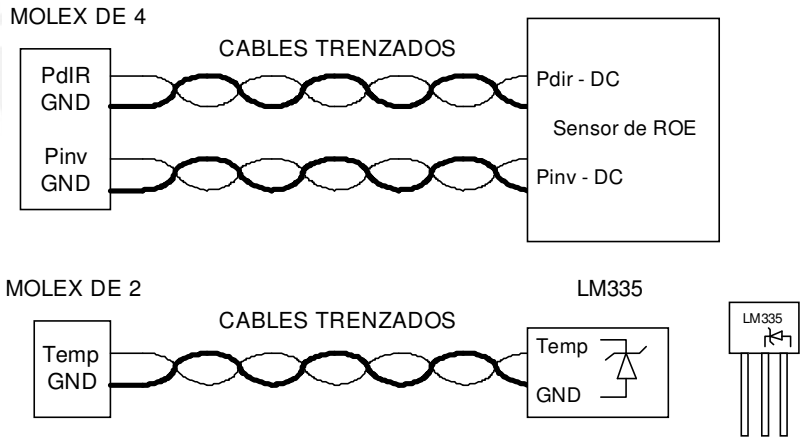


Figura 5.6.4.1 Conexión del dispositivo a la radio ICOM IC-78

El montaje se muestra en la figura 5.6.4.2.

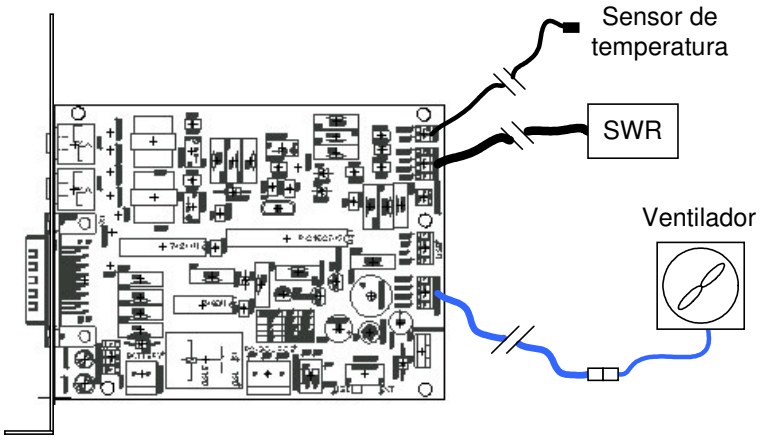


Figura 5.6.1.1 Conexión de los sensores del dispositivo

NOTAS:

- ***Distancia máxima = 2m.***
- ***Puede usarse un cable UTP flexible multihilo.o cables apantallados como los de audio mono o estereo.***
- ***La vista de los conectores de las radios son con respecto a las mismas.***

5.7. CONEXIÓN USB

La conexión de la interfaz USB a la placa madre de la computadora se ve en la figura 3.5.1, hay que indicar que éste esquema es sólo para las placas madre VIA EPIA, para los demás habrá que consultar el manual.

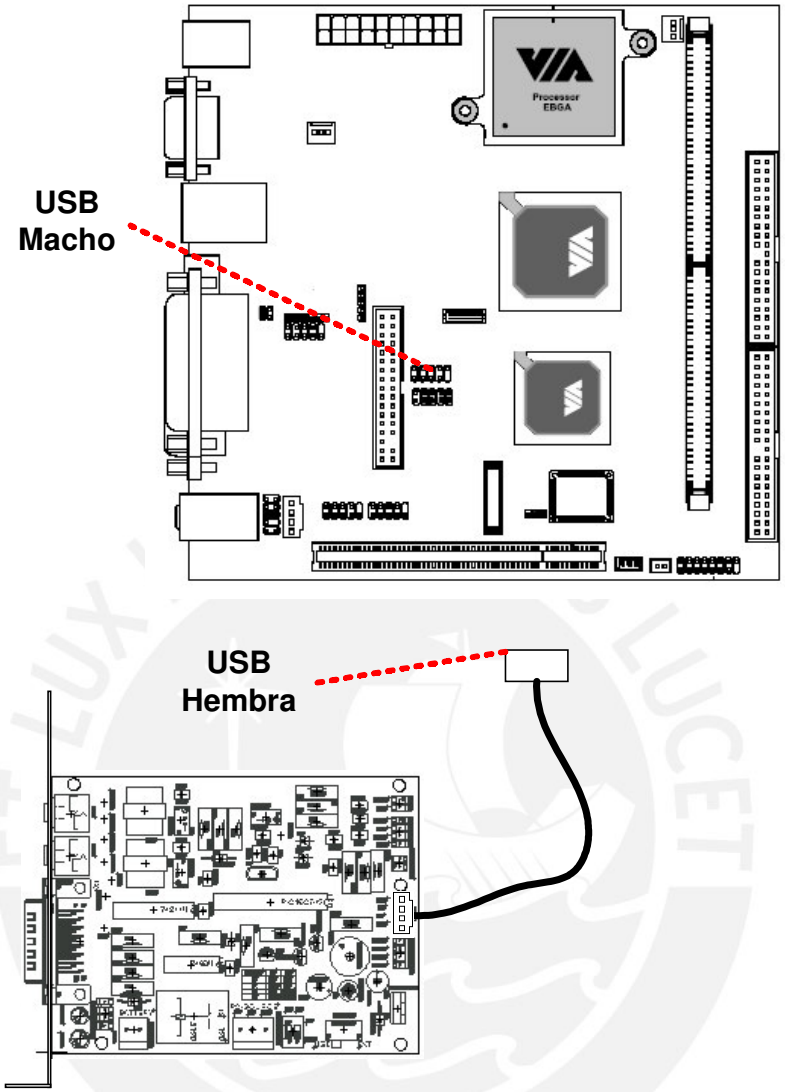


Figura 5.7.1 Conexión USB del dispositivo a la placa madre

NOTA:

Sólo hay una forma de conectar ambos dispositivos

5.8. CONEXIÓN DE ALIMENTACIÓN

Antes de empezar con el cableado de alimentación se deberá tener en cuenta si tiene fluido eléctrico permanente o si la fuente de energía es una batería, en cualquier caso se debe tener el interruptor Selector de Alimentación, mostrado en la figura 5.8.1, en la posición correcta.

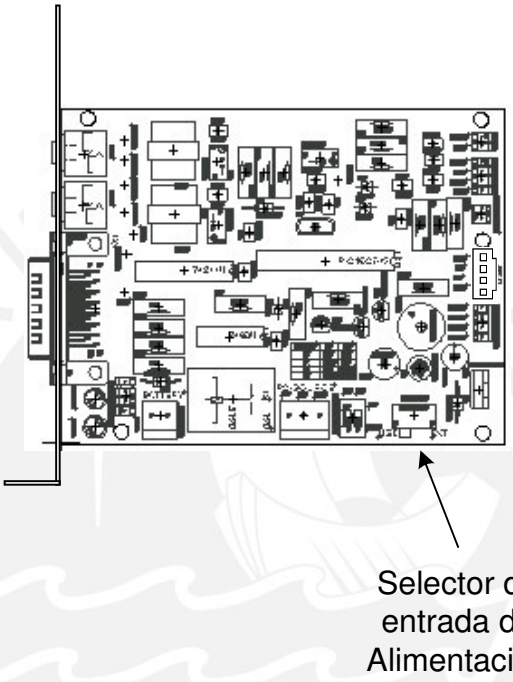


Figura 5.8.1 Vista del selector de alimentación del dispositivo

5.8.1. PARA LAS ESTACIONES CON FLUÍDO ELÉCTRICO

Si se cuenta con fluido eléctrico permanente bastará con tener el interruptor en la posición que indica la figura, lo que significa que la tarjeta se alimentará directamente del puerto USB.

Figura 5.8.1.1 Selector configurado para recibir alimentación del puerto USB

NOTA:

Cabe recordar que en esta opción no se deben configurar las opciones de control de voltaje, de encendido y apagado externo ni de control de temperatura (El ventilador se alimenta de la batería).

5.8.2. PARA LAS ESTACIONES SIN FLUIDO ELÉCTRICO

En este caso la Placa Estación tomará control del encendido de la computadora por lo que se deberá hacer el conexionado que se explica en adelante.

1. Antes de hacer cualquier conexión se debe tener en cuentas que los conectores estén en buen estado y los cables bien soldados a ellos, así como la posición correcta del interruptor de Selección de Alimentación como indica la figura.

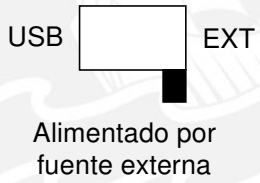


Figura 5.8.2.1 Selector configurado para recibir alimentación externa

2. La simbología de colores es ROJO para la línea positiva y NEGRO para la línea negativa y, en este caso, también es tierra como se ve en la figura 5.8.2.2.

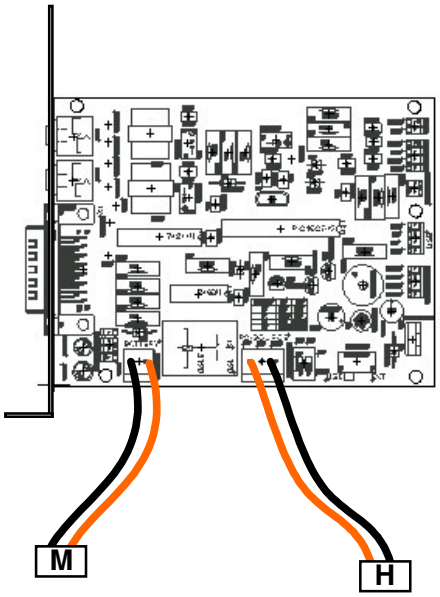


Figura 5.8.2.2 Vista de los cables de poder de la tarjeta.

Donde los conectores empleados son de tipo batería, macho M y hembra H.

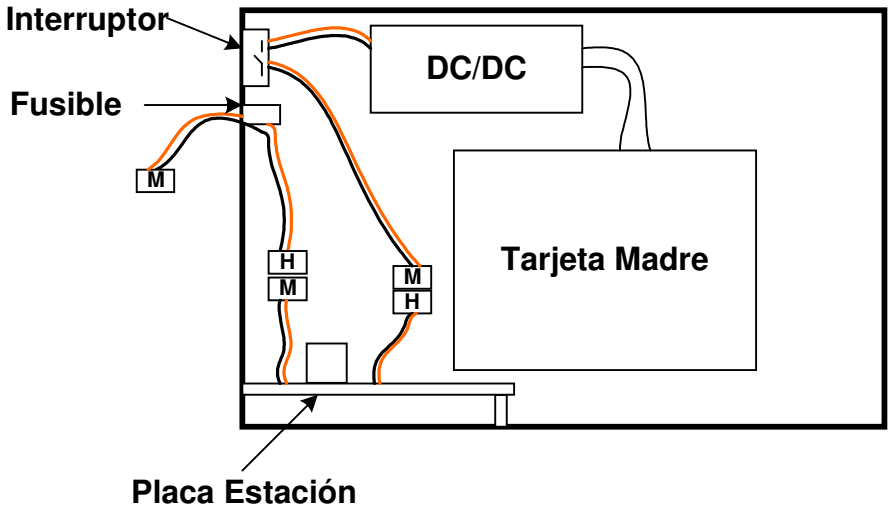


Figura 5.8.2.3 Conexiones de poder

Las conexiones mostradas en la Figura 5.8.2 están en la parte posterior de la caja de computadora y los bloques son referenciales no indican el lugar exacto de los componentes internos de la computadora.

5.9. CALIBRACIÓN DE LA TARJETA.

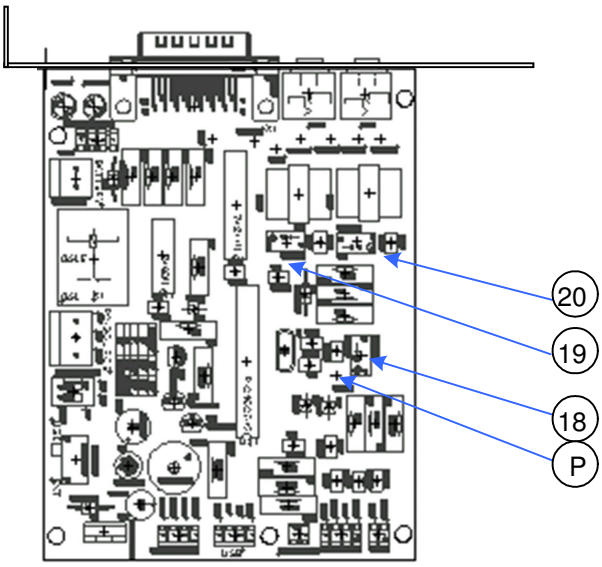


Figura 5.9.1 Puntos de calibración

P: Punto de prueba de 4V.
18: Potenciómetro para calibrar **referencia** a 4V.
19: Potenciómetro para calibrar nivel de **salida (speaker)** de audio.
20: Potenciómetro para calibrar nivel de **entrada (line-in/mic)** de audio.

El valor de voltaje de referencia aumenta en sentido antihorario.
Los valores de resistencia aumentan en sentido antihorario.

El siguiente diagrama muestra los siguientes puntos importantes para la calibración tanto del voltaje referencial para las entradas analógicas como para la parte de audio.

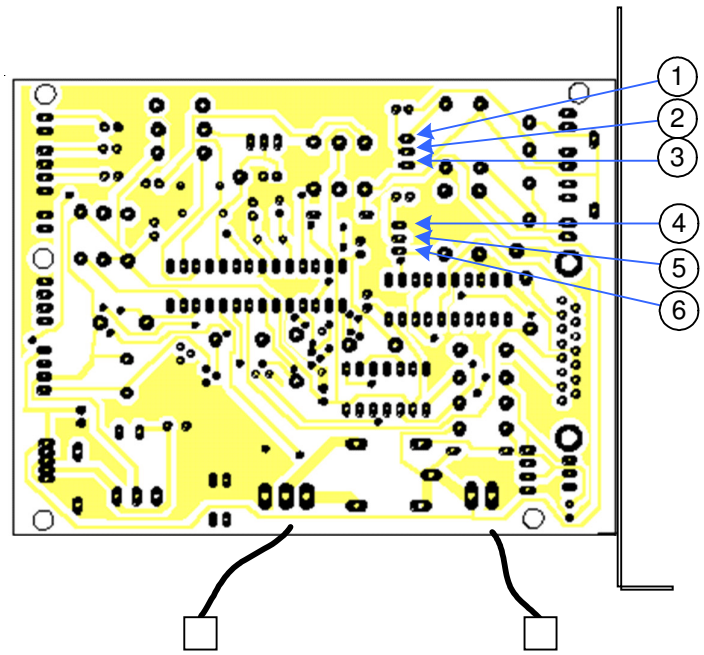


Figura 5.9.2 Puntos de prueba

En la figura 5.9.2 los puntos 1, 2 y 3 pertenecen al potenciómetro de Entrada y los valores aproximados que deben tener se muestran en la siguiente tabla:

Resistencia entre puntos del potenciómetro	***Conexión HF***	***Conexión VHF***
R 1 – 2	**0,5 KOhm**	**3 KOhm**
R 2 – 3	**99,5 KOhm**	**97 KOhm**
R 4 – 5	**35 KOhm**	**35 KOhm**
R 5 – 6	**65 KOhm**	**65 KOhm**

Tabla 5.9.1 Valores de calibración aproximados

NOTAS:

- *Los puntos 4, 5 y 6 pertenecen al potenciómetro de Entrada.*
- *Estos valores son referenciales, la calibración final se debe hacer con una prueba de transmisión, que junto con un mezclador como puede ser el Rexima nos den los*

valores finales. Los valores con los que debe empezar las pruebas en el programa antes mencionado son los siguientes:

```
                                    rexima

                    min  .    .    .    .    :    .    .    .    .  max
     Vol            [=========================|-------------------------]
50% [ ]
     Pcm            [=========================|-------------------------]
50%
     Line           [|-----------------------------------------------------]
0% [ ]
     Mic            [|-----------------------------------------------------]
0% [R]
     CD             [|-----------------------------------------------------]
0% [ ]
     Pcm2           [=========================|-------------------------]
50%
     IGain          [=========================|-------------------------]
50%
     Line1          [|-----------------------------------------------------]
0% [ ]
     PhoneIn        [|-----------------------------------------------------]
0% [ ]
```

Figura 5.9.3: Vista de la consola de control de audio

- *El rexima es parte del paquete ALSA-MIXER que se puede descargar libremente de la red.*

5.10. MONTAJE

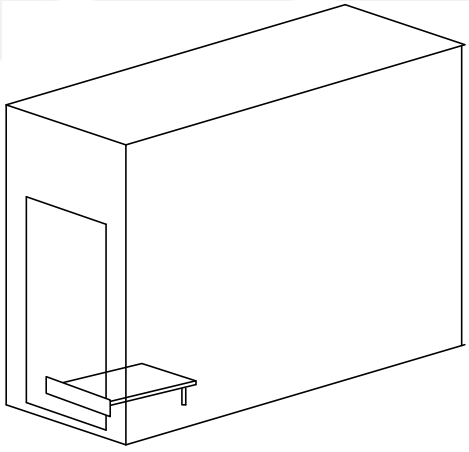


Figura 5.10.1 Montaje en una caja de computadora comercial

La Figura 5.10.1 muestra la posición de montaje en la parte posterior de una caja de computadora el cual ya vendrá con dos soportes que coinciden con los agujeros mostrados en la figura 5.10.2

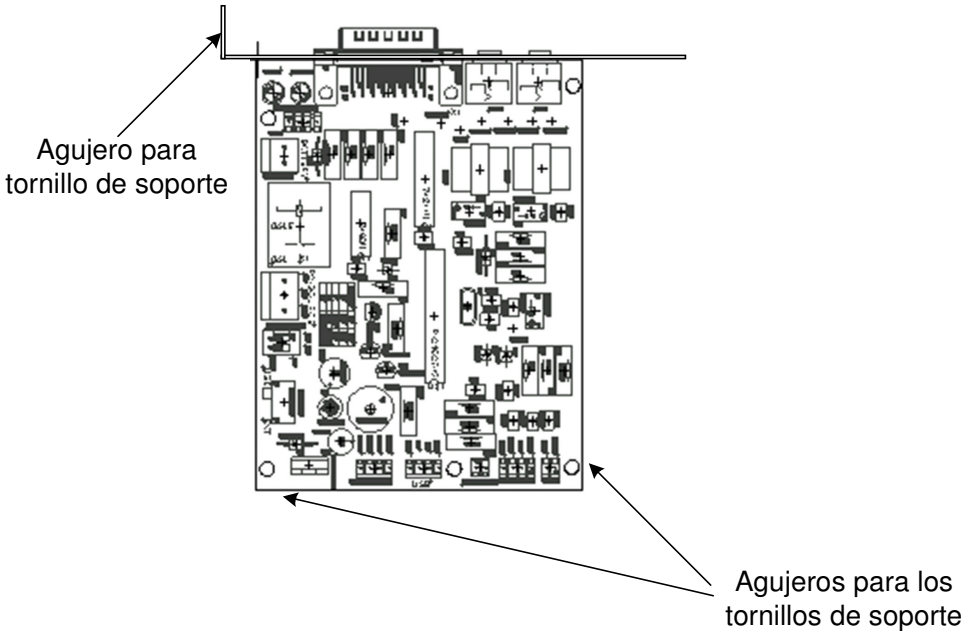


Figura 5.10.2 Puntos de soporte

5.11. CONFIGURACIÓN DE LA PLACA ESTACIÓN

Para que la placa estación funcione en conjunto con el software modem “ehas-station” se debe configurar dentro de la interfaz “config-ehas”de la siguiente manera:

config-ehas

Figura 5.11.1 Pantalla principal de la interfaz

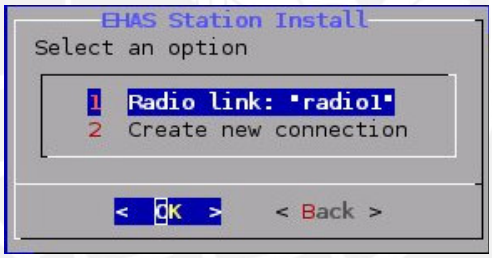


Figura 5.11.1 Se crea una conexión nueva, en nuestro caso “radio1” el cual puede ser para radios HF o VHF.

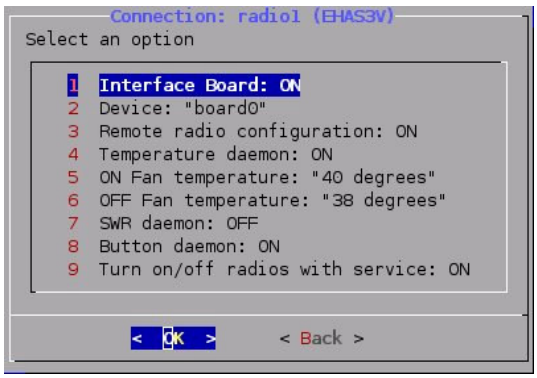


Figura 5.11.2 Activación del sensado de temperatura y los demás parámetros de conexión HF o VHF.

Figura 5.11.3 Configuración del dispositivo USB.

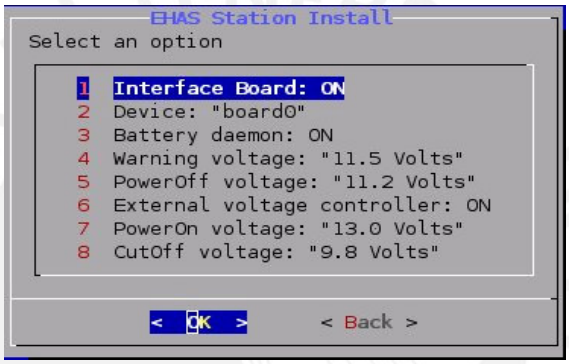


Figura 5.11.4 Activación del sensado de voltaje:

Al terminar de configurar se sale de la interfaz de configuración y se escribe sobre los cambios realizados con la siguiente instrucción:

```
# write config-ehas
```

5.12. COMANDOS DE MANEJO DE LA PLACA INTERFAZ

Los comandos para manejar la placa interfaz vienen en el paquete "ehas-board" que es complemento del paquete “ehas-station”, y son:

eboard [OPCIONES] [COMANDO] [PUERTO]

Comando generado directamente por el driver "eboard"

OPCIONES:

-v: Muestra cada envío que hace el driver eboard al PIC

-q: quita comentarios y argumentos y sólo muestra el valor en viado o recibido

-b: Selecciona la tarjeta a la que va la orden: Board (0/1/2/3) (No se usa si la tarjeta es Board0)

-l: Muestra la lista de las tarjetas conectadas
Es el primer comando que se usa para ver si la(s) station Board(s) enumera(n)

Ejemplo:

 # eboard -l
EHAS Board found (version 1.3) ID: 0

-p: Abre comando y sólo espera opciones (no es muy útil, mejor se hace con el "etestboard")

COMANDOS:

version: muestra la versión de Station Board

set / **unset** PUERTO:

pone a 1 o cero un PUERTO de salida).

Ejemplo:

 # eboard unset led
Gate <led>: 0 *(Pone led a color verde)*
 # eboard set led
Gate <led>: 1 *(Pone led a color rojo)*

get PUERTO:

Lee el valor que tiene un PUERTO, se usa de preferencia para un PUERTO de entrada pero también se puede leer en qué valor quedó un PUERTO de salida.

Ejemplo:

 # eboard get radio_state
Gate <radio_state>: 0 *(radio vhf está en 0, apagada)*
 # eboard get button
Gate <button>: 1 *(botón externo está en 1, no activado)*
 # eboard get led
Gate <led>: 1 *(el led quedo en 1, verde)*

analog PUERTO:

Sólo para leer PUERTOS analógicos indicados abajo.

Ejemplo:

 # eboard analog battery
Battery (V): 12.36
 # eboard analog temperature
Temperature (Celsius): 25.04

hysteresis Von Voff:

Para poner valores de histéresis de control de voltaje externo que maneja la Station Board con el relé, mejor hacerlo con "config-ehas"

txserial STRING:

para trasmisión serial en radios HF, para enviar una orden completa este formato no es muy legible se recomienda usar el comando "catradio" explicado más abajo, que usa este comando de eboard.

reset

=ATENCIÓN= reinicia la Placa Estación, si se usa reiniciar también la PC para que la placa se vuelva a enumerar sino no funciona nada.

Ejemplo:

 # eboard reset
... reset successful
 # reboot

PUERTOS:

Estos son los PUERTOS disponibles en la Placa Estación de entradas y salidas digitales y entradas analógicas.

Salida: led, buzzer, fan, ptt, invert_rs232, data_channel, ignition, pc_plug.
Entrada: radio_state, button.
Entrada Analógica: battery, temperature, swr.

etesboard -c [conexión]

Comando que realiza la prueba de funcionamiento de la Placa Estación, para que funcione debidamente se configura primero en “config-ehas” el número de tarjeta que se esté usando porque las pruebas servirán solamente para ella:

una vez configurado se procede con las pruebas.

Ejemplo:
 # etesboard -c radio1

Con esta opción hace el test completo de la tarjeta que consiste en probar lo siguiente:

PARA RADIOS VHF Y HF

SALIDAS: led, buzzer, fan.
ENTRADAS: temperature, swr.

PARA LA RADIO VHF

SALIDAS: ptt, data_channel, ignition.
ENTRADAS: radio_state, button.

PARA LA RADIO HF

SALIDAS: ptt, SET_MEMCHAN 1/2 (toggle entre el canal 1 y 2).
ENTRADAS: button.

NOTAS:

- ***Para probar los puertos digitales se cambia de valor pulsando cualquier tecla y para pasar al siguiente test se presiona ENTER.***
- ***Para cancelar cualquier test se hace con crtl+c.***

para hacer una prueba por separado sólo hay que indicar el puerto que se quiere probar.

Ejemplo:

 # etestboard -c radio1 data_channel (para cambiar de canal en la radio vhf)
Connection: radio1
Board: board0
Selected tests: data_channel

***** DATA_CHANNEL digital output test *****
Gate <data_channel>: 1
DATA_CHANNEL on
Press Enter to skip the test, any other key to toggles
Gate <data_channel>: 0
DATA_CHANNEL off
Press Enter to skip the test, any other key to toggle

 # etestboard -c radio1 temperature (para ver la temperatura en tiempo real)
Connection: radio1
Board: board0
Selected tests: temperature
***** TEMPERATURE analog input test *****
Press Enter to skip the test
TEMPERATURE: 23.47
TEMPERATURE: 23.47
TEMPERATURE: 23.47
.....

En todos los casos para ver el test de voltaje de la batería NO se necesita indicar la conexión “radio1”.

Ejemplo:

 # etestboard battery
No connection given, defaulting to global connection
Connection: global
Board: board0
Selected tests: battery
***** BATTERY analog input test *****
Press Enter to skip the test
BATTERY: 12.30
BATTERY: 12.30
BATTERY: 12.30
BATTERY: 12.36
.....

epradio [-b device_board] [-r radio] on | off

Comando de encendido y apagado de radio.

Ejemplo:

 # epradio -r vhf on
Using device board: board0
Radio type: vhf
New state: on
Radio is off. Turning it on... done

enciende la radio vhf

catradio [-q] [-d device_port] [-m radio_model] COMANDO [parm]

Comando de transmisión serial para radios HF

COMANDOS:
salidas:
SET_POWER SET_TXPOWER SET_MODE SET_SCAN SET_MEMCHAN SET_GAIN CLEAR_CLARIFIER SET_SQUELCH RESET_LOCAL

entradas:
GET_INFO GET_TXPOWER GET_MODEL GET_MEMCHAN GET_METER GET_MODE GET_PTT GET_SQUELCH GET_SCAN GET_GAIN GET_BUSY

Ejemplo:

limahf:~# catradio -d board0 -m kenwood SET_MEMCHAN 27

- Cambia al canal 27, que previamente se había programado en la radio con los siguientes parámetros:

Frecuencia de transmisión y recepción: 19,600MHz
Banda de transmisión: LSB.

limahf:~# catradio -d board0 -m kenwood GET_INFO

00009160000	*00000*	*27_01*	*00*
frecuencia	squelch	mem_mode	gain

5.13. PRUEBAS DE LA CONEXIÓN DE AUDIO

Estas pruebas ya están incluidas en el software MODEM “Ehas-Station” por lo que no se detallará el funcionamiento pero sí se usará para verificar el correcto funcionamiento de la conexión de audio y la correcta calibración de la placa estación en la parte de adaptación de impedancias.

Con el comando “etestax25 –c radio1” se envían paquetes de prueba a otro terminal en donde allí se usará el comando “ tail –f /var/log/daemon.log” que muestra los logs de recepción en donde se salen los resultados de conexión. La figura 5.13.1 muestra la disposición de elementos para la prueba.

Figura 5.13.1 Disposición de pruebas de conexión

Con esta disposición se está probando la salida del Terminal 1 (Ruta OUT) y la entrada del Terminal 2 (Ruta IN).

Una prueba satisfactoria en HF mostrará los siguientes resultados:

tail –f /var/log/daemon.log

```
        soundmodem[5046]: Sync: 10 (inter-symbol)
        soundmodem[5046]: Preamble at: +0.20Hz
        soundmodem[5046]: RX: Control block: l: 58, FECL: 4, FEC = 31, reqFEC
        = 20
        soundmodem[5046]: newqpskrx[CHUPH]: BCH. Errors = 0/1728
        soundmodem[5046]: Packet done at: +0.20Hz, final path metric: 0
        soundmodem[5046]: S/N ratio: 22 31 21 25 25 23 24 15 23 31 21 25 24
        25 17  dB
soundmodem[5046]: newqpskrx[CHUPH]: S/N ratio = 21.6
```

En donde la relación señal a ruido S/N promedio es de 21.6 lo cual es muy aceptable.

Una prueba satisfactoria en VHF mostrará los siguientes resultados:

tail –f /var/log/daemon.log

```
        localhost soundmodem[5129]: newfsk: (64/64)-(120/120) rate = 6, req =
        1, l: 75. OK (ts: 151.56 mseg)
        localhost soundmodem[5129]: Frame 01. DATA_OK. Errors = 0/600
        localhost soundmodem[5129]: newfsk: Window OK: RX-Frames: 1/1. Errors
        = 0/600 (0.00%)
```

En los resultados obtenidos podemos ver que la pérdida de paquetes es 0 y es lo que buscamos.

Invirtiendo el orden de prueba se terminará de probar ambas salidas y ambas entradas de la tarjeta. Cualquier falla de conexión causado por la tarjeta se debe regresar a la etapa de calibración explicado en el punto 5.9.

Todas las pruebas de calibración se hacen en el laboratorio en donde las distorsiones por el canal se puedan despreciar. Ya en el campo si se ve alguna variación se puede mover algún parámetro lo menos posible pero sólo para mejorar transmisión de datos.

6. COSTOS DEL CIRCUITO

A continuación se detalla el costo de cada componente electrónico y materiales empleados así como los servicios de mano de obra para el montaje de la placa estación. Los costos están en referencia al mercado local.

Placa Estación

Cantidad	Descripción	P/Unitario	P/Total S/.	Total $
1	Socket de integrado de 28 (14x2 redondo) Tipo DIL	S/. 5.00	S/. 5.00	$ 1.54
1	Buffer 74LS244	S/. 3.00	S/. 3.00	$ 0.93
1	XOR 7486	S/. 2.00	S/. 2.00	$ 0.62
1	Cristal 6MHz	S/. 5.00	S/. 5.00	$ 1.54
1	Regulador LM2576	S/. 15.00	S/. 15.00	$ 4.63
1	Bobina 100uH	S/. 7.00	S/. 7.00	$ 2.16
1	Diodo 1N5822	S/. 3.50	S/. 3.50	$ 1.08
2	Diodo 1N4148	S/. 0.60	S/. 1.20	$ 0.37
1	Diodo Zener 4.7V	S/. 0.70	S/. 0.70	$ 0.22
2	Diodo Zener 5.1V	S/. 0.70	S/. 1.40	$ 0.43
1	Led azul	S/. 1.50	S/. 1.50	$ 0.46
1	Led bicolor rojo-verde	S/. 1.50	S/. 1.50	$ 0.46
1	Termistor LM335	S/. 7.00	S/. 7.00	$ 2.16
3	Transistores 2N2222	S/. 0.70	S/. 2.10	$ 0.65
1	Relay de 10A @ 12V	S/. 4.00	S/. 4.00	$ 1.23
1	Buzzer 5V	S/. 2.80	S/. 2.80	$ 0.86
1	DIP Switch de 2	S/. 1.50	S/. 1.50	$ 0.46
1	Switch de montaje de 3 pines (Source Select)	S/. 0.70	S/. 0.70	$ 0.22
1	Conector DB15 MACHO de tarjeta	S/. 3.80	S/. 3.80	$ 1.17
3	Conector MOLEX de 4 pines	S/. 1.00	S/. 3.00	$ 0.93
2	Conector MOLEX de 2 pines	S/. 0.60	S/. 1.20	$ 0.37
2	Conector JACK HEMBRA STEREO de tarjeta	S/. 1.50	S/. 3.00	$ 0.93
2	Conector aéreo de bateria con cable	S/. 7.00	S/. 14.00	$ 4.32
7	Pin espadilla de test point	S/. 3.00	S/. 21.00	$ 6.48
1	Resistencia 100Ω ¼W	S/. 0.10	S/. 0.10	$ 0.03
2	Resistencia 330 Ω¼W	S/. 0.10	S/. 0.20	$ 0.06
1	Resistencia 1kΩ¼W	S/. 0.10	S/. 0.10	$ 0.03

Cantidad	Descripción	P/Unitario	P/Total S/.	Total $
1	Resistencia 1.5kΩ ¼W	S/. 0.10	S/. 0.10	$ 0.03
3	Resistencia 2kΩ ¼W	S/. 0.10	S/. 0.30	$ 0.09
5	Resistencia 3.3kΩ ¼W	S/. 0.10	S/. 0.50	$ 0.15
1	Resistencia 5kΩ ¼W	S/. 0.10	S/. 0.10	$ 0.03
1	Resistencia 10kΩ ¼W	S/. 0.10	S/. 0.10	$ 0.03
1	Resistencia de precisión 5kΩ ¼W	S/. 0.60	S/. 0.60	$ 0.19
1	Resistencia de precisión 15kΩ ¼W	S/. 0.60	S/. 0.60	$ 0.19
1	Potenciómetro vertical de precisión 2kΩ	S/. 4.50	S/. 4.50	$ 1.39
2	Potenciómetro vertical de precisión 100kΩ	S/. 5.00	S/. 10.00	$ 3.09
2	Condensador monolítico 22pF	S/. 0.20	S/. 0.40	$ 0.12
9	Condensador monolítico 0.1uF	S/. 0.20	S/. 1.80	$ 0.56
1	Condensador monolítico 0.3uF	S/. 0.40	S/. 0.40	$ 0.12
1	Condensador electrolítico 100uF / 25V	S/. 0.80	S/. 0.80	$ 0.25
1	Condensador electrolítico 1000uF / 16V	S/. 1.00	S/. 1.00	$ 0.31
1	Estaño y pasta de soldadura	S/. 1.60	S/. 1.60	$ 0.49
1	Circuito impreso metalizado	S/. 55.00	S/. 55.00	$ 16.98
1	Servicio de montado y soldado	S/. 40.00	S/. 40.00	$ 12.35
			Total Placa Estación	**$ 70.71**

Tipo de Cambio S/. 3.24

Cable de la Estación HF

Cantidad	Descripción	P/Unitario	P/Total S/.	Total $
1	Conector DB15 Hembra Aéreo (con capucha)	S/. 3.50	S/. 3.50	$ 1.08
1	Conector DIN de 6 pines	S/. 3.50	S/. 3.50	$ 1.08
1	Conector DIN de 8 pines	S/. 4.00	S/. 4.00	$ 1.23
2	Metros de cable multifilar de 9 hilos flexible blindada	S/. 2.80	S/. 5.60	$ 1.73
1.5	Metros de cable UTP multifilar (flexible)	S/. 0.50	S/. 0.75	$ 0.23
1	Estaño y pasta de soldar	S/. 1.00	S/. 1.00	$ 0.31
1	Termorretractil 1.5"	S/. 2.50	S/. 2.50	$ 0.77
1	Pegamento y cinta aislante	S/. 0.50	S/. 0.50	$ 0.15
1	Servicio de soldadura y ensamble	S/. 10.00	S/. 10.00	$ 3.09
			Total Cable HF	**$ 9.68**

Tipo de Cambio S/. 3.24

Cable de la Estación VHF

Cantidad	Descripción	P/Unitario	P/Total S/.	Total $
1	Conector DB15 Hembra Aéreo (con capucha)	S/. 3.50	S/. 3.50	$ 1.08
1	Conector 10x2 hembra tipo espadines	S/. 3.50	S/. 3.50	$ 1.08
1.5	Metros de cable multifilar de 9 hilos flexible blindada	S/. 2.80	S/. 4.20	$ 1.30
1	Estaño y pasta de soldar	S/. 1.00	S/. 1.00	$ 0.31
1	Termorretractil 1.5"	S/. 2.50	S/. 2.50	$ 0.77
1	Termorretractil 2"	S/. 2.80	S/. 2.80	$ 0.86
1	Pegamento y cinta aislante	S/. 0.70	S/. 0.70	$ 0.22
1	Servicio de soldadura y ensamble	S/. 10.00	S/. 10.00	$ 3.09
			Total Cable VHF	**$ 8.70**

Tipo de Cambio S/. 3.24

De los cuadros mostrados se ve que el costo de una tarjeta completa es de casi $ 80 dólares con el tipo de cambio correspondiente a la fecha. Los costos del desarrollo no se consideran.

CONCLUSIONES

Desde el punto de vista de la ingeniería en nuestro país, el desarrollo de un circuito impreso, en general, tiene un costo muy elevado, en la mayoría de casos conviene encontrar una solución comercial alternativa de tal manera que se cubran las necesidades de un proyecto. Adicionalmente con un producto comercial se asegura la reposición y el mantenimiento. Por lo tanto la parte económica no juega un papel crítico para decidir el desarrollo de una tarjeta para un uso específico que en nuestro caso tendrá más ventajas optando por la propia implementación, tales como: el conocimiento detallado del producto en todos sus aspectos, la mejor adaptación a los requerimientos buscados y la opción a mejoras constantes y cambios que se puedan presentar. Estas ventajas no se podrían dar con un producto comercial, como se mostró en nuestra aplicación.

En vista de que se atendió una necesidad real de comunicación rural se buscó que nuestro trabajo de ingeniería quede como un producto comercial permita la producción masiva, en donde los costos de fabricación disminuirán en gran medida, siendo éste uno de los objetivos de la presente tesis.

Con la experiencia de nuestra aplicación se tuvo opción a un manejo de herramientas valiosas tanto de hardware como de software que posibilitan una constante actualización y mejora del producto implementado, así como la oportunidad de brindar el soporte y mantenimiento necesarios para su vigencia en el mercado.

OBSERVACIONES

Este proyecto aún sigue en etapa de evaluación y se esperan muchas mejoras para futuro tales como la digitalización completa de los controles de nivel de audio usando potenciómetros digitales.

El sensor de ROE también sigue en etapa de experimentación pues para su fabricación es necesaria una alta precisión en la fabricación de sus partes mecánicas para lograr de esta manera una mayor simetría en ambos lados del sensado (directo e inverso) necesaria para lograr que la fórmula planteada se cumpla.

No se pudo detallar más el código del software tanto del firmware como del software de aplicación pues como se trata de un producto comercial los derechos le pertenecen al programa EHAS.

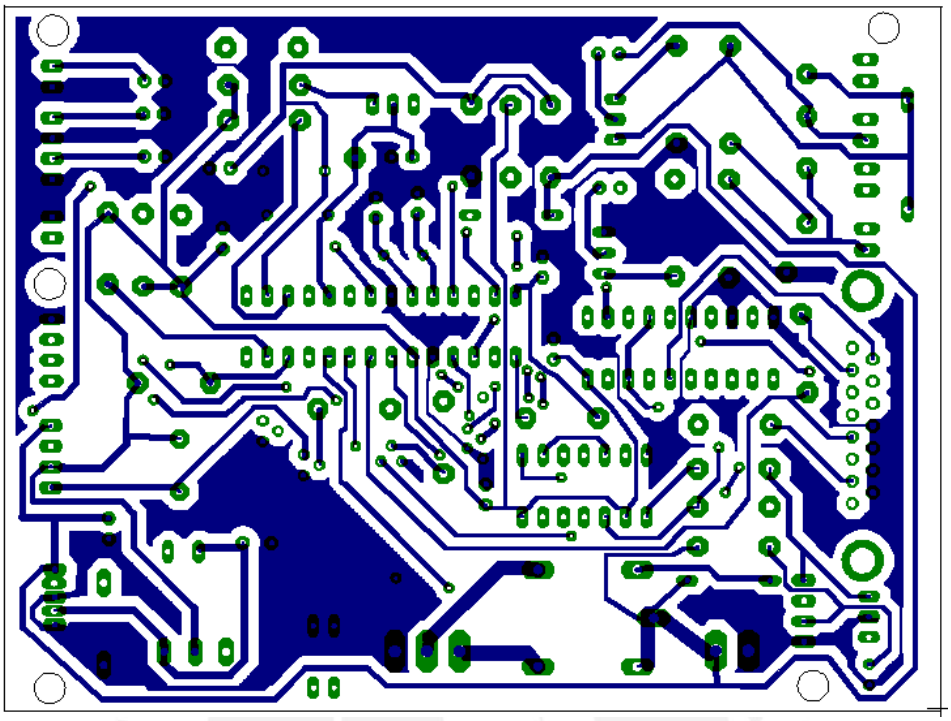
BIBLIOGRAFÍA

1. AXELSON, JAN. 2001. USB Complete 2nd Edition. Lakeview Research.
2. CHIPMAN, R. A. Líneas de Transmisión. Mc Graw-Hill, Inc.
3. ARRL HANDBOOK. 2002. Manual para los radio aficionados.

Páginas Web visitadas:

1. Manual del Microcontrolador 16C7X5 de Microchip, http://www.microchip.com
2. Manual del puerto USB 1.0 y USB 2.0, http://www.usb.org
3. Manual para el linux Debian http://www.debian.org
4. Páginas de desarrolladores de hardware USB http://libusb.sourceforge.net/doc
5. Herramientas de desarrollo de hardware en windows http://www.microsoft.com/winddk.

CARA INFERIOR (VISTA INVERTIDA)



CARA SUPERIOR

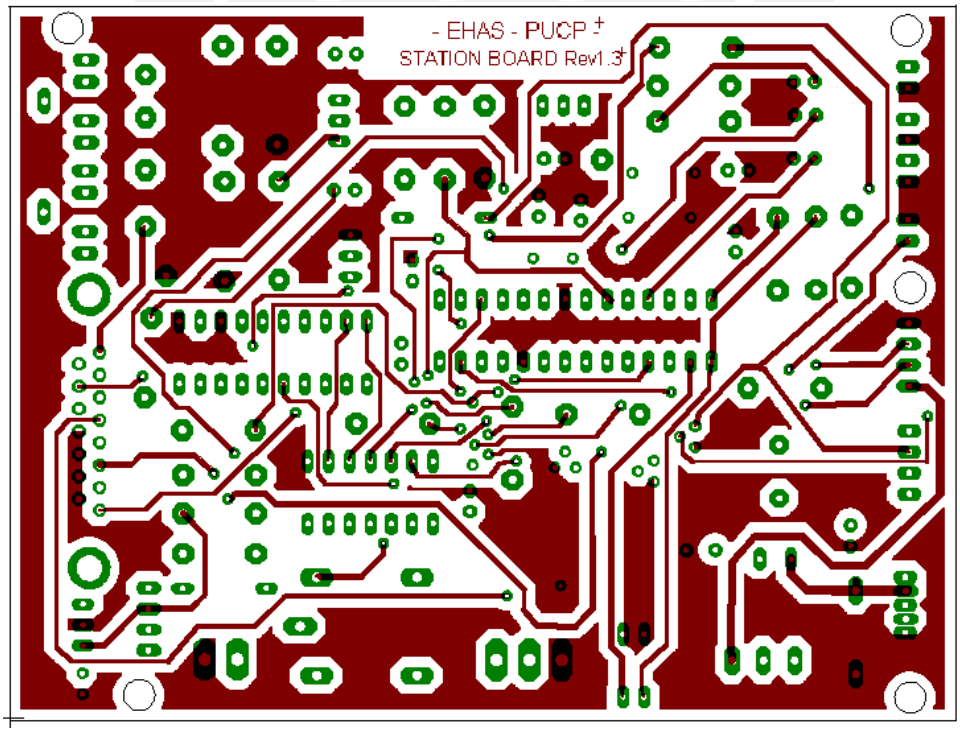
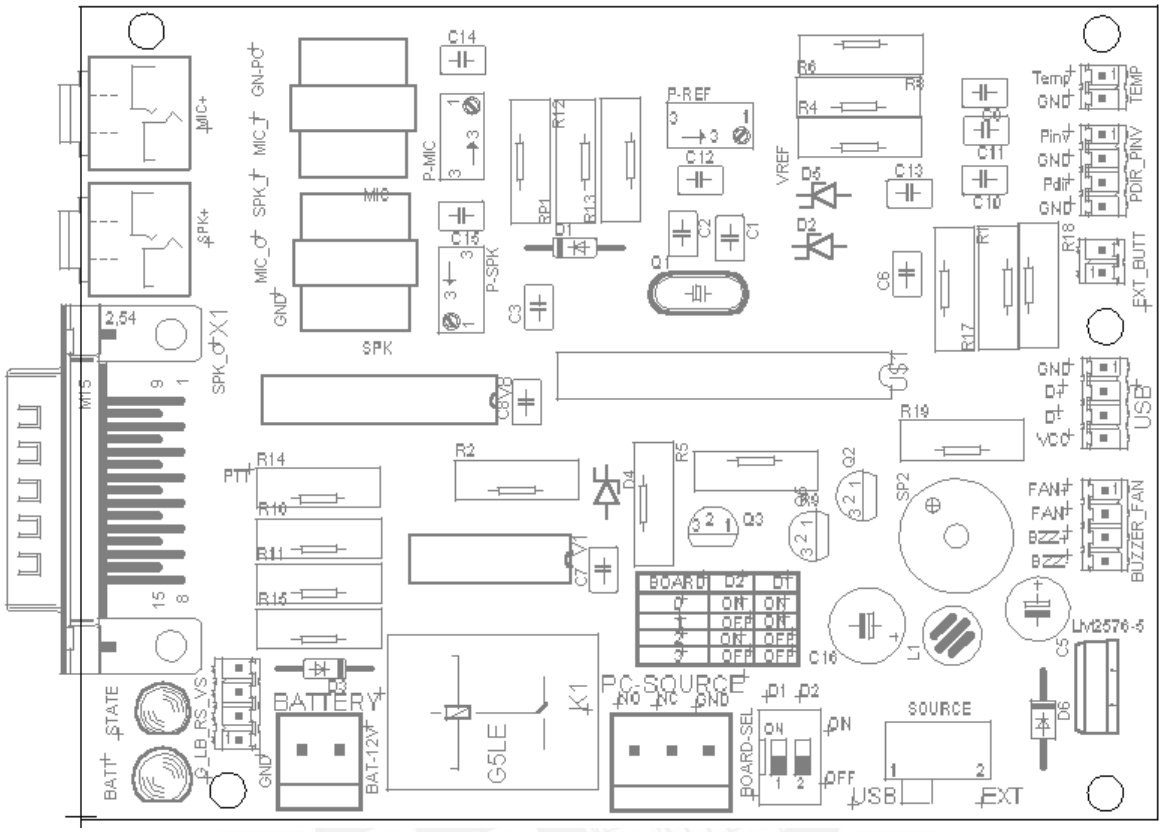
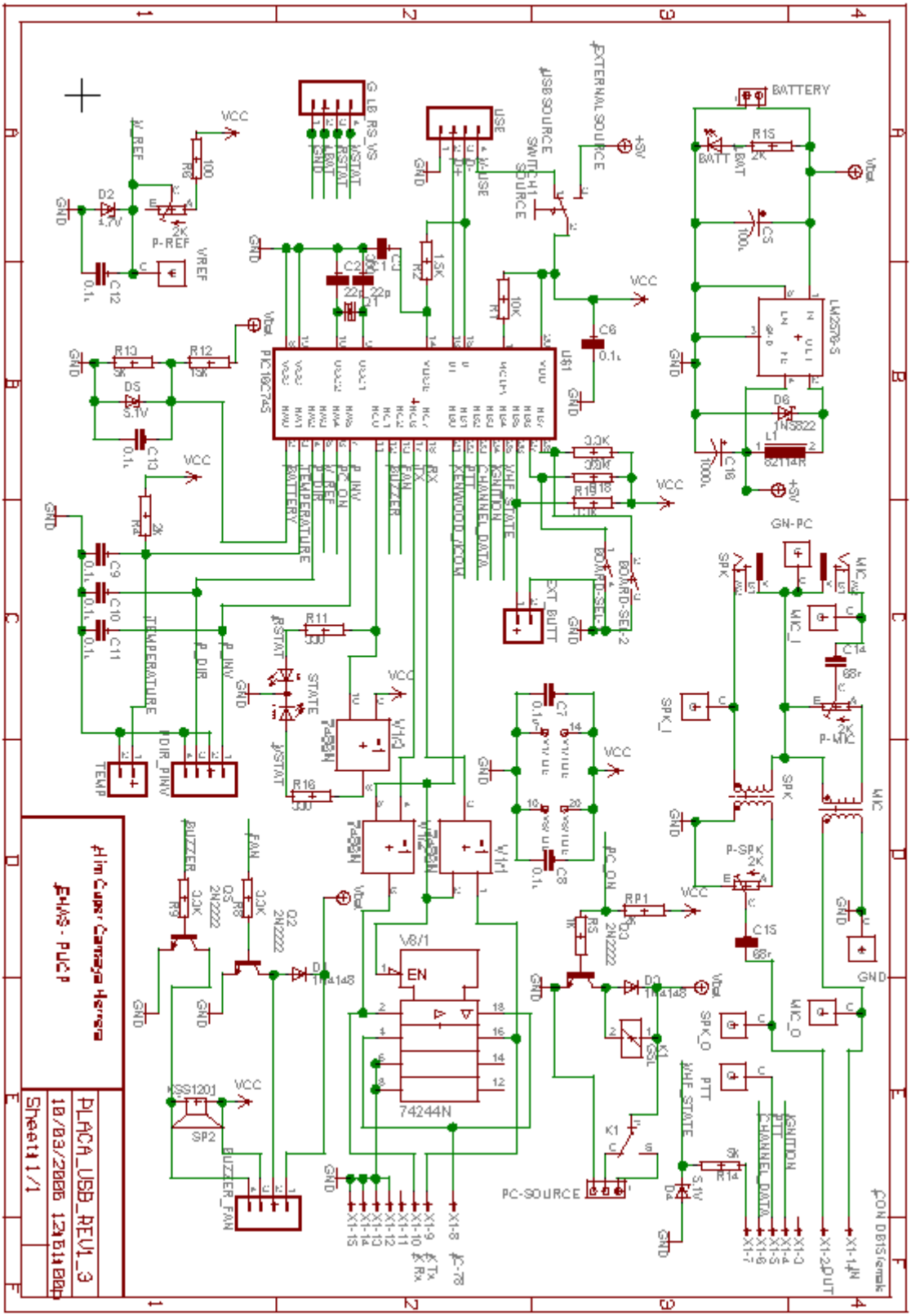


DIAGRAMA DE DISTRIBUCIÓN DE COMPONENTES



ESQUEMÁTICO DEL CIRCUITO



FIRMWARE DEL MICROCONTROLADOR

Programa principal usando el compilador "PCWH C COMPILER"

```
/* Device used: PIC16C745 */
/* Firmware developed by Him Cansaya and Arnau Sanchez- 2004 */
/* hcansaya@ehas.org, arnau@ehas.org */

#include <16c745.h>
#include "usbcom.h"

/* Version 1.1

      - USB packet length: 4

 Version 1.2

      - USB packet length: 8

 Version 1.3

      - Reference voltage: 4.0
      - Buzzer using a standard speaker

 Version 1.4

      - Reset function

*/

#define VERSION   1
#define SUBVERSION      4

#device *=16

/* FUSES ********************************/
/* H4: 6 MHz Crystal                    */
/* NOWDT: Watchdog disabled             */
/* NOPROTECT: Don't protect PIC code    */
/* PUT: Power-up-Timer enabled          */

#fuses H4,NOWDT,NOPROTECT,PUT

/* Although using an external 6 MHz clock, internal clock is always 24MHz
*/

#use delay(clock=24000000)

/* RS232 for HF-CAT. 9600bps 8N1. Tx: C6, Rx: C7 */

#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

/* Timeout for RS232 RX = 500 msec */
#define RS232_TIMEOUT          500
```

```
/* USB constants */

#define USB_EP1_TX_ENABLE  1
#define USB_EP1_TX_SIZE    8

#define USB_EP1_RX_ENABLE  1
#define USB_EP1_RX_SIZE    8

#include "pic_usb.h"
#include "usb.c"

/*********************************
 Input/Output map
*********************************/

/* Sensor input. Analog-to-digital */

#define BATTERY              PIN_A0
#define TEMPERATURE     PIN_A1
#define SWR_DIRECT           PIN_A2
#define SWR_INVERSE          PIN_A5

#define BATTERY_ADC               0
#define TEMPERATURE_ADC      1
#define SWR_DIRECT_ADC       2
#define SWR_INVERSE_ADC      4

#define ADC_READS 16

/* Misc digital outputs */

#define LED                  PIN_C0
#define BUZZER               PIN_C1
#define FAN                  PIN_C2

/* Misc inputs */

#define BUTTON               PIN_B5

/* Common radio */

#define PTT                  PIN_B1

/* HF radio */

#define INVERT_RS232    PIN_B0

/* VHF radio */

#define VHF_SET_DATA    PIN_B2
#define VHF_IGNITION    PIN_B3

#define RADIO_STATE          PIN_B4

/* Board Identifier (0, 1, 2, 3) */

#define BOARD_ID1       PIN_B7
```

```
#define BOARD_ID0       PIN_B6

#define PC_PLUG               PIN_A4


/* Global variables */

char timeout_error;
char rs232buffer[RS232_MAX_BUFFER];
char rxlength, txlength, txindex = 0;

float on_voltage, off_voltage;
char pc_state;

/* Output state */

char led_state, buzzer_state, fan_state, ptt_state, plug_state;
char invert_state, data_state, ignition_state;

/*****************************************************************/
/*****************************************************************/
/*****************************************************************/

/*************************************************************************
*/
/* reset_pic: Resets PIC
                        */
/*************************************************************************
*/

void reset_pic(void)
{
#asm
   bcf 03,6
   bcf 03,7
   movlw 020
   movwf 04
next:
   clrf 00
   incf 04
   btfss 04,7
   goto next
#endasm

reset_cpu();
}

/*************************************************************************
*/
/* timed_getc: Waits for incoming RS232 data byte, with timeout (msecs)
*/
/*************************************************************************
*/

char timed_getc(int timeout) {

      long elapsed = 0;
```

```
	timeout_error = FALSE;
	while( !kbhit() && (++elapsed < timeout) )
		delay_us(1000);
	if( kbhit() )
		return(getc());
	else
		timeout_error = TRUE;
	return 0;
}


/*************************************************************************
*/
/* rs232_send: Sends string to RS232
                                    */
/*************************************************************************
*/

void rs232_send(char *buffer, int length)
{
	int i;

	for(i=0; i<length; i++)
		putc(buffer[i]);
}


/*************************************************************************
*/
/* rs232_recv: Waits for incoming RS232 data string, with timeout
(msecs)*/
/*************************************************************************
*/

char rs232_recv(char *buffer, int max_length, int timeout) {

	int length = 0;
	char c;

	while (length < max_length) {
		c = timed_getc(timeout);
		if (timeout_error)
			break;
		buffer[length++] = c;
	}

	return length;

}

/*************************************************************************
*/
/* adc_read: Read ADC channel
                              */
/*************************************************************************
*/
```

```
char adc_read(channel)
{
      int i;
      int16 sum = 0;
      set_adc_channel( channel );
      for(i=0; i<ADC_READS; i++) {
            delay_us(10);
            sum += (int16)read_adc();
      }
      sum /= ADC_READS;
      return (char)sum;
}

/*************************************************************************
*/
/* flash_led: Flash the led (on/off) with a given delay
            */
/*************************************************************************
*/

void flash_led(int times, int delay)
{
      int i;

      for (i=0; i<times*2; i++) {
            output_low(LED);
            delay_ms(delay);
            output_high(LED);
            delay_ms(delay);
      }
      output_bit(LED, led_state);
}


/*************************************************************************
*/
/* buzzer: Activates/deactivates a speaker buzzer (square wave)
*/
/*************************************************************************
*/

void buzzer_speaker(char state)
{
      if (state) {
            setup_ccp2(CCP_PWM);                      // clk=6MHz, =>
F=2KHz
            setup_timer_2(T2_DIV_BY_16, 186, 1);      //
(1/6000000)*16*187*1= 498.67 us or 2 khz
            set_pwm2_duty(375);                       // 50%,
6000000(1/2)*(1/2000)*(1/4)=375
      }
      else {
            setup_ccp2(CCP_OFF);
            output_low(BUZZER);
    }
}
```

```
/*************************************************************************
*/
/* rs232_init: Init the Serial Device
                  */
/*************************************************************************
*/

void rs232_init(void)
{
      /* Send a test char to RS-232 port */
      int ninit = 10;

      strcpy(rs232buffer, "init;");
      rs232_send(rs232buffer, 5);
      while (ninit-- > 0) {
            if ( rs232_recv(rs232buffer, RS232_MAX_BUFFER, RS232_TIMEOUT)
== 0 )
                  break;
      }
}

/*************************************************************************
*/
/* board_init: Initialize EHAS board. Outputs and LED flashing
*/
/*************************************************************************
*/

void board_init(void)
{
      /* Initialize global variables */

      pc_state = TRUE;
      on_voltage = off_voltage = -1;
      flash_led(2, 200);

      /* Initialize digital outputs */

      output_high(LED);
      output_low(BUZZER);
      output_low(FAN);
      output_high(INVERT_RS232);
      output_high(VHF_SET_DATA);

      /* VHF_IGNITION remains in high impedance */
      /* output_high(VHF_IGNITION); */

      output_high(PC_PLUG);
      output_high(PTT);

      /* Initialize global variables related with digital outputs*/

      led_state = 1;
      buzzer_state = fan_state = invert_state = ignition_state = 0;
      ptt_state = data_state = 1;
```

```
	/* Initialize ADC */

	setup_adc(ADC_CLOCK_INTERNAL);
	setup_adc_ports(ANALOG_RA3_REF);

	rs232_init();
}


/************************************************************************
*/
/* usb_read: Try to read a USB command from PC
             */
/************************************************************************
*/

char usb_read(char *buffer)
{
	if (usb_kbhit(1)) {
		usb_gets(1, buffer, USB_EP1_RX_SIZE);
		return TRUE;
	}
	return FALSE;
}

/************************************************************************
*/
/* usb_write: Write a USB command to PC
             */
/************************************************************************
*/

char usb_write(char *buffer)
{
	if ( usb_put_packet(1, buffer, USB_EP1_TX_SIZE, TOGGLE) )
		return TRUE;
	return FALSE;
}

/************************************************************************
*/
/* set_bit: Set an output bit with USB Gate Identifier
             */
/************************************************************************
*/

char set_bit(char pin, char value)
{
	switch(pin) {
	case USBC_GATE_LED:	output_bit(LED, value); led_state = value;
return 0;
	case USBC_GATE_FAN:		output_bit(FAN, value); fan_state =
value; return 0;
	case USBC_GATE_PTT:		output_bit(PTT, value); ptt_state =
value; return 0;
```

```
      case USBC_GATE_INVERT:  output_bit(INVERT_RS232, value);
invert_state = value; return 0;
      case USBC_GATE_DATAC:   output_bit(VHF_SET_DATA, value); data_state
= value; return 0;
      case USBC_GATE_IGNITION:output_bit(VHF_IGNITION, value);
ignition_state = value; return 0;
      case USBC_GATE_PLUG:    output_bit(PC_PLUG, value); plug_state =
value; return 0;
      case USBC_GATE_BUZZER:  buzzer_speaker(value); buzzer_state =
value; return 0;
      }
      return 0xFF;
}


/*************************************************************************
*/
/* get_bit: Get an input bit value with USB Gate Identifier
            */
/*************************************************************************
*/

char get_bit(char pin)
{
      switch(pin) {
      /* Input gates */
      case USBC_GATE_RADIO_STA: return ( input(RADIO_STATE));
      case USBC_GATE_BUTTON: return ( input(BUTTON));

      /* Output gates */
      case USBC_GATE_LED:     return led_state;
      case USBC_GATE_BUZZER:  return buzzer_state;
      case USBC_GATE_FAN:           return fan_state;
      case USBC_GATE_PTT:           return ptt_state;
      case USBC_GATE_INVERT:  return invert_state;
      case USBC_GATE_DATAC:   return data_state;
      case USBC_GATE_IGNITION:return ignition_state;
      case USBC_GATE_PLUG:    return plug_state;
      }
      return 0xFF;
}

/*************************************************************************
*/
/* get_adc: Get an ADC value usign channel from USB Gate Identifier
       */
/*************************************************************************
*/

int16 get_adc(char channel)
{
      char chan = 0xFF;

      switch(channel) {
      case USBC_GATE_BATTERY: chan = BATTERY_ADC; break;
      case USBC_GATE_TEMP:    chan = TEMPERATURE_ADC; break;
      case USBC_GATE_SWR_DIR: chan = SWR_DIRECT_ADC; break;
```

```
	case USBC_GATE_SWR_INV: chan = SWR_INVERSE_ADC; break;
	}

	if (chan != 0xFF)
		return (adc_read(chan) & 0xFF);

	return 0xFFFF;
}


/*************************************************************************
*/
/* fill_error: Fill answer error
                */
/*************************************************************************
*/

void fill_error(char *in_buffer, char *out_buffer, char error)
{
	out_buffer[0] = 0xFF;
	out_buffer[1] = in_buffer[0];
	out_buffer[2] = error;
	out_buffer[3] = 0;
}

/*************************************************************************
*/
/* process_command: Process an USB command and prepares the answer
	*/
/*************************************************************************
*/

void process_command(char *in_data, char *out_data)
{
	int16 adc_value;
	char retval, rxindex, flash;

	memcpy(out_data, in_data, 4);

	flash = 0;

	switch(in_data[0]) {

	case USBC_GET_VERSION:
		flash = 1;
		out_data[1] = SUBVERSION;
		out_data[2] = VERSION;
		out_data[3] = 0;
		break;

	case USBC_GET_DIGITAL:
		retval = get_bit( in_data[1] );
		out_data[2] = retval;
		out_data[3] = 0;
		if (retval == 0xFF)
			fill_error(in_data, out_data, GATE_UNKNOWN);
		break;
```

```
	case USBC_GET_ANALOG:
		adc_value = get_adc( in_data[1] );
		out_data[2] = adc_value & 0xFF;
		out_data[3] = 0;
		if (adc_value == 0xFFFF)
			fill_error(in_data, out_data, GATE_UNKNOWN);
		break;

	case USBC_SET_DIGITAL:
		retval = set_bit( in_data[1], in_data[2] );
		if (retval == 0xFF)
			fill_error(in_data, out_data, GATE_UNKNOWN);
		break;

	case USBC_TX_SERIAL:

		/* If Packet Index is 0, reset TX state */
		if (in_data[1] == 0 && txindex > 0)
			txindex = 0;

		/* Check if command packet is the expected */
		if (in_data[1] != txindex) {
			fill_error(in_data, out_data, RS232_WRONG_INDEX);
			txindex = 0;
			break;
		}

		/* Check if it's the first packet, then creates first answer
*/
		if (txindex == 0) {
			txlength = in_data[2];
			txindex = 1;
			if (txlength > RS232_MAX_BUFFER) {
				fill_error(in_data, out_data, RS232_WRONG_LENGTH);
				txindex = 0;
			}
			break;
		}
		else {
			memcpy(rs232buffer + ((txindex-1)*2), in_data + 2, 2);

			/* Check if that was the last packet */
			if (2*txindex >= txlength) {
				usb_write(out_data);
				rs232_send(rs232buffer, txlength);
				txindex = 0;
			}
			else {
				txindex++;
				break;
			}
		}

		/* Last packet sent. Now fill RX RS232 with radio response */
		rxlength = rs232_recv(rs232buffer, RS232_MAX_BUFFER,
RS232_TIMEOUT);
```

```
        return;

case USBC_RX_SERIAL:

        /* First packet. Return length */

        if (in_data[1] == 0) {
              out_data[1] = 0;
              out_data[2] = rxlength;
              out_data[3] = 0;
              break;
        }

        /* Index cannot be greater than RS232_MAX_BUFFER/2 */

        rxindex = in_data[1];
        if (rxindex > RS232_MAX_BUFFER/2)
              rxindex = RS232_MAX_BUFFER/2;

        out_data[1] = rxindex;
        memcpy(out_data + 2, rs232buffer + ((rxindex-1)*2), 2);

        /* Last packet read, then reset buffer */

        if (2*rxindex >= rxlength)
              rxlength = 0;
        break;

case USBC_BOARD_ID:
        retval =  input(BOARD_ID0);
        retval += 2*input(BOARD_ID1);
        out_data[2] = retval;
        out_data[1] = out_data[3] = 0;
        break;

case USBC_SET_HYSTERESIS:
        on_voltage  = ((float)in_data[1])/10.0;
        off_voltage = ((float)in_data[2])/10.0;
        break;

case USBC_NOP:
        return;


case USBC_RESET:
        reset_pic();

default:
        fill_error(in_data, out_data, COMMAND_UNKNOWN);
        break;

}
usb_write(out_data);

/* Flash LED to indicate command received */
if (flash)
        flash_led(2, 20);
```

```
}


/************************************************************************
****/
/* check_battery: Gets voltage and plugs/unplugs PC (with hysteresis
cycle) */
/************************************************************************
****/

void check_battery()
{
      char val;
      float voltage, on, off;

      val = adc_read(BATTERY_ADC);
      voltage = ((float)val * BATTERY_FACTOR * ADC_REFERENCE) / 255.00;

      on  = (on_voltage  > 0)?on_voltage :DEFAULT_ON_VOLTAGE;
      off = (off_voltage > 0)?off_voltage:DEFAULT_OFF_VOLTAGE;

      if (pc_state == TRUE && voltage < off) {
            pc_state = FALSE;
            output_bit(PC_PLUG, 0);
      }
      else if (pc_state == FALSE && voltage > on) {
            pc_state = TRUE;
            output_bit(PC_PLUG, 1);
      }
}



/************************************************
********** MAIN ***********************************
************************************************/

void main() {

      /* USB buffer */
      int8 out_data[USB_EP1_TX_SIZE];
      int8 in_data [USB_EP1_RX_SIZE];

      board_init();
      usb_init();

      flash_led(2, 150);

      /* Main loop */

      for(;;) {
            if (usb_enumerated())
                  if( usb_read(in_data) )
                        process_command(in_data, out_data);

            if (!usb_enumerated())
                  flash_led(2, 100);
```

```
            check_battery();
        }

        /* oops, USB port reset or disconnected. Let's reset the PIC */
        reset_cpu();
}
```

CABECERA “com.h”

```
/*********************************************************************/
/* USB Commands                                                      */

/*      0: Get Board Version.       00 XX XX XX              -> 00 SUBVERSION VERSION 00  */

/*      1: Get Digital Inputs.      01 GATE XX XX            -> 01 GATE VALUE 00                          */

/*      2: Get Analog Inputs.       02 AN_GATE XX XX   -> 02 AN_GATE VALUE 00                    */

/*      3: Set Digital Outputs. 03 GATE VALUE XX       -> 03 GATE VALUE 00                          */

/*      4: RS232 TX + Init RX.  04 00 LENGTH XX        -> 04 00 LENGTH 00                           */
/*                                                        04 01 D1 D2                  -> 04 01 D1 D2
                                                                                                    */
/*                                                        04 02 D3 D4                  -> 04 02 D3 D4
                                                                                                    */
/*                                                                                        ...
                                                                                                    */
/*                                                        04 NN DL-1 DL          -> 04 NN DL-1 DL
                                                                                                    */

/*      5: RS232 RX.                    05 00 XX XX                        -> 05 00 LENGTH 00
                                                                                                    */
/*                                                        05 01 XX XX                  -> 05 01 D1 D2
                                                                                                    */
/*                                                        05 02 XX XX                  -> 05 02 D3 D4
                                                                                                    */
/*                                                                                        ...
                                                                                                    */
/*                                                        05 NN XX XX                  -> 05 NN DL-1
DL                                                                                                  */

/*      6: Radio Model                  06 X XX XX                         -> 06 00 ID 00
                                                                                                    */

/*      7: Hysteresis voltage       07 VON VOFF XX          -> 07V ON VOFF 00                       */
/*                                                         (V = VALUE * 10)
                                                                                                    */

/*      On error returns: FF COMMAND ERROR 00
                                                                                                    */

/*********************************************************************/
```

```
/* Constants */

#define BATTERY_FACTOR 4.0
#define ADC_REFERENCE 4.0
#define DEFAULT_ON_VOLTAGE 12.0
#define DEFAULT_OFF_VOLTAGE 9.5

/* Define USB Errors */

#define COMMAND_UNKNOWN       1
#define GATE_UNKNOWN          2
#define RS232_WRONG_INDEX     3
#define RS232_WRONG_LENGTH    4

/* RS232 Buffer size */
#define RS232_MAX_BUFFER      64
```

```
#define USBC_GET_VERSION      0
#define USBC_GET_DIGITAL      1
#define USBC_GET_ANALOG       2
#define USBC_SET_DIGITAL      3
#define USBC_TX_SERIAL        4
#define USBC_RX_SERIAL        5
#define USBC_BOARD_ID         6
#define USBC_SET_HYSTERESIS   7
#define USBC_RESET                  8

#define USBC_NOP              100

/* Output */

#define USBC_GATE_LED         0
#define USBC_GATE_BUZZER      1
#define USBC_GATE_FAN         2
#define USBC_GATE_PTT         3
#define USBC_GATE_INVERT      4
#define USBC_GATE_DATAC       5
#define USBC_GATE_IGNITION    6
#define USBC_GATE_PLUG        7

/* Analog Input */

#define USBC_GATE_BATTERY     16
#define USBC_GATE_TEMP        17
#define USBC_GATE_SWR_DIR     18
#define USBC_GATE_SWR_INV     19

/* Digital Input */

#define USBC_GATE_RADIO_STA   32
#define USBC_GATE_BUTTON      33
```

CONTROLADOR LINUX

PROGRAMA PRINCIPAL "main.c"

```
#include <stdio.h>
#include <string.h>
#include <usb.h>
#include "eboard_ctr.h"

/* eboard: Controls an EHAS USB Board */

/* Driver for EHAS BOARD V1.1 */
/* Program developed by Arnau Sanchez and Him Cansaya – 2004 */

/* eboard COMMAND [OPTIONS]

   COMANDS: version
                set         GATE
                unset    GATE
                get         GATE
                analog      GATE
                txserial STRING (or write to stdin)
                rxserial (written to stderr)

   GATE:    output: led, buzzer, fan, ptt, invert_rs232, data_channel
                input:      radio_state button
                analog: battery, temperature, swr
*/


/***********************************************
* fill_command
***********************************************/


void fill_command(char *command, char c1, char c2, char c3, char c4)
{
      *command++ = c1;
      *command++ = c2;
      *command++ = c3;
      *command++ = c3;
}


/***********************************************
* get_gate
***********************************************/

int get_gate(char *strgate)
{
      char *gate_table[12] = { "led", "buzzer", "fan", "ptt",
"invert_rs232", "data_channel", "ignition",
                                          "battery", "temperature", "swr",
"radio_state", "button"};
```

```
	int gate_table_com[12] = {USBC_GATE_LED, USBC_GATE_BUZZER,
USBC_GATE_FAN, USBC_GATE_PTT,
                                                USBC_GATE_INVERT,
USBC_GATE_DATAC, USBC_GATE_IGNITION,
                                                USBC_GATE_BATTERY,
USBC_GATE_TEMP, USBC_GATE_SWR, USBC_GATE_RADIO_STA, USBC_GATE_BUTTON};


	int i;

	for(i=0; i<12; i++)
		if (strcasecmp(strgate, gate_table[i]) == 0)
			return (gate_table_com[i]);
	return -1;
}

/*********************************************
* options
*********************************************/

void options(void)
{
	printf("Usage: eboard [OPTIONS] COMMAND\n");
	printf("OPTIONS: -v: Be verbose\n");
	printf("         -q: Quiet. Just return values\n");
	printf("         -b: Board (0/1/2/3)\n");
	printf("         -l: List conected boards\n\n");

	printf("COMANDS: version\n");
	printf("         set      GATE    VALUE\n");
	printf("         get      GATE\n");
	printf("         analog   GATE\n");
	printf("         txserial STRING (if null, read from stdin.
Received string to stderr)\n");
	printf("GATE:    output:  led, buzzer, fan, ptt, invert_rs232,
data_channel, ignition\n");
	printf("         input:   radio_state button\n");
	printf("         analog:  battery, temperature, swr\n");

	exit(1);

}


int parse_command(int argl, char *strcommand, char *stroption, char
*command, unsigned char *serial_buffer, int *serial_length)
{
	int command_length, gate, i, j, n, set = 0;

	memset(command, 0, 256);
	command_length = 4;

	if ( strcasecmp(strcommand, "version") == 0 ) {
		fill_command(command, USBC_GET_VERSION, 0, 0, 0);
	}

	else if ( strcasecmp(strcommand, "get") == 0 ) {
```

```
            if (argl < 2)
                  options();
            gate = get_gate( stroption );
            if ( gate < 0 )
                  return (-2);
            fill_command(command, USBC_GET_DIGITAL, (char)gate, 0, 0);
      }

      else if ( strcasecmp(strcommand, "set") == 0 ||
strcasecmp(strcommand, "unset") == 0) {
            if (argl < 2)
                  options();
            gate = get_gate( stroption );
            if ( gate < 0 )
                  return (-2);

            if ( strcasecmp(strcommand, "set") == 0 )
                  set = 1;

            fill_command(command, USBC_SET_DIGITAL, (char)gate, set, 0);

      }

      else if ( strcasecmp(strcommand, "analog") == 0 ) {
            if (argl < 2)
                  options();
            gate = get_gate( stroption );
            if ( gate < 0 )
                  return (-2);

            if (gate == USBC_GATE_SWR) {
                  fill_command(command, USBC_GET_ANALOG,
USBC_GATE_SWR_DIR, 0, 0);
                  fill_command(command+4, USBC_GET_ANALOG,
USBC_GATE_SWR_INV, 0, 0);
                  command_length = 8;
            }
            else {
                  fill_command(command, USBC_GET_ANALOG, (char)gate, 0,
0);
            }
      }

      else if ( strcasecmp(strcommand, "txserial") == 0 ) {
            if (argl < 2) {
                  *serial_length = read(0, serial_buffer,
RS232_MAX_BUFFER);
            }
            else {
                  *serial_length = strlen(stroption);
                  strncpy(serial_buffer, stroption, RS232_MAX_BUFFER);
            }

            i = j = n = 0;

            fill_command(command, USBC_TX_SERIAL, i, *serial_length, 0);
            i++;
```

```
            n+=4;

            while (j < *serial_length) {
                  fill_command(command+n, USBC_TX_SERIAL, i,
serial_buffer[j], serial_buffer[j+1]);
                  i++;
                  j+=2;
                  n+=4;
            }
            fill_command(command+n, USBC_RX_SERIAL, 0, 0, 0);
            command_length = n + 4;
      }

      else {
            return -1;
      }
      return command_length;
}


/*********************************************
* MAIN
*********************************************/

int main(int argc, char **argv)
{
      /* USB pointers */
      usb_dev_handle *udev = NULL;

      /* Options */
      int board = 0, verbose = 0, quiet = 0, list=0;

      int n, argl, retval, icommand;
      int command_length, serial_length;
      double direct = 0.0, inverse = 0.0, vdirect, vinverse;

      unsigned char serial_buffer[RS232_MAX_BUFFER+1];
      unsigned char serial_txbuffer[RS232_MAX_BUFFER+1];
      unsigned char command[256];
      char err_buffer[256];

      char *stroption, *strcommand;

      /* Get options */

      while ((n = getopt(argc, argv, "b:vql")) != -1) {
            switch (n) {
                  case 'b': /* Board identifier (0 or 1) */
                        board = atoi(optarg);
                        break;
                  case 'v': /* Verbose */
                        verbose = 1;
                        break;
                  case 'q': /* Quiet, just return values */
                        quiet = 1;
                        break;
                  case 'l': /* List connected boards */
```

```
                        list = 1;
                        break;
                default:
                        options();
                        break;
        }
    }

    if (list) {
        for(board = 0; board < 4; board++) {
            udev = ehas_board_init(board, verbose, err_buffer);
            if (udev) {
                printf("EHAS Board found with ID: %x\n", board);
                ehas_board_close(udev);
            }
        }
        exit(0);
    }


    /* Remove options from parametrs */
    argl = argc - optind;

    /* At least we need a command */
    if (!list && argl < 1)
        options();

    strcommand = argv[optind];
    stroption  = argv[optind+1];

    command_length = parse_command(argl, strcommand, stroption,
command, serial_buffer, &serial_length);

    memcpy(serial_txbuffer, serial_buffer, RS232_MAX_BUFFER);

    if (command_length == -1) {
        printf("Error: Command unknown: %s\n\n", strcommand);
        options();
    }
    else if (command_length == -2) {
        printf("Error: Gate unknown: %s\n\n", stroption);
        options();
    }

    icommand = command[0];

    /* Init USB */

    udev = ehas_board_init(board, verbose, err_buffer);

    if (udev == NULL) {
        printf("Error: EHAS Board not found with id #%d\n(returned
message: %s).\n", board, err_buffer);
        printf("Check that USB cable is connected and ID board switch
is correct (-b option).\n");
        exit(1);
    }
```

```
	retval = ehas_board_send(udev, command, command_length, verbose,
serial_buffer, err_buffer);
	ehas_board_close(udev);

	if (retval < 0) {
		printf("Error: %s\n", err_buffer);
		exit(1);
	}

	switch (icommand) {

	case USBC_GET_VERSION:
		if (!quiet)
			printf("EHAS Board version: ");
		printf("%d.%d\n", (retval >> 8) & 0xFF, retval & 0xFF);
		break;

	case USBC_SET_DIGITAL:
	case USBC_GET_DIGITAL:
		if (!quiet)
			printf("Gate <%s>: ", stroption);
		printf("%d\n", retval);
		break;

	case USBC_GET_ANALOG:
		if (!quiet) {
		    if (strcmp(stroption, "battery") == 0)
			printf("Battery (V): ");
		    else if (strcmp(stroption, "temperature") == 0)
			printf("Temperature (Celsius): ");
		    else if (strcmp(stroption, "swr") == 0)
			printf("SWR: ");
		}

		if (strcmp(stroption, "battery") == 0) {
			printf("%0.2f\n", 3.0*5.0*(float)retval/255.0);
		}

		else if (strcmp(stroption, "temperature") == 0) {
			if (retval == 0) {
				printf("Temperature sensor error or not
connected\n");
				exit(1);
			}
			printf("%0.2f\n", ((5.0*(float)retval/255.0) * 100.0) -
273.0);
		}
		else if (strcmp(stroption, "swr") == 0) {

			if (serial_buffer[0] == serial_buffer[1]) {
				printf("SWR sensor error: SWR_direct = SWR_invert
= %f\n", 5.0*(float)serial_buffer[0]/255.0 );
				exit(1);
			}
```

```
                direct  = (double)serial_buffer[0];
                inverse = (double)serial_buffer[1];

                vdirect = 5.0*serial_buffer[0]/255.0;
                vinverse = 5.0*serial_buffer[1]/255.0;

                printf("%0.2f (direct=%0.2f, inverse=%0.2f)\n",
(direct+inverse)/(direct-inverse), vdirect, vinverse);
            }


            break;

      case USBC_TX_SERIAL:
            printf("TX RS232 command sent: ");
            for(n=0; n<serial_length; n++)
                  printf("%X ", serial_txbuffer[n]);
            printf("\n");

            printf("RX RS232. Bytes Read: %d\n", retval);
            write(2, serial_buffer, retval);
            break;

      }


      exit(0);
}
```

PROGRAMA DE CONTROL DEL PUERTO “eboard_ctr.c”

```
#include "eboard_ctr.h"

#include <stdio.h>
#include <usb.h>
#include <string.h>

#define MANUFACTURER "EHAS"
#define PRODUCT    "Board"

#define USB_READ_TIMEOUT  2000
#define USB_RESPONSE_ERROR 0xFF


/****************/

int usb_response_read(usb_dev_handle *udev, unsigned char *buffer, char
*err_buffer)
{

      if ( usb_interrupt_read(udev, 1, buffer, 4, USB_READ_TIMEOUT) < 4)
{
            sprintf(err_buffer, "Error: USB read");
            return 1;
```

```
	}

	if (buffer[0] == USB_RESPONSE_ERROR) {
		sprintf(err_buffer, "Error: USB response: ");
		switch(buffer[2]) {
			case COMMAND_UNKNOWN: sprintf(err_buffer, "Command
unknown"); break;
			case GATE_UNKNOWN: sprintf(err_buffer, "Gate unknown");
break;
			case RS232_WRONG_INDEX: sprintf(err_buffer, "Wrong
RS232 packet index"); break;
			case RS232_WRONG_LENGTH: sprintf(err_buffer, "Wrong
RS232 packet length"); break;
		}
		return 1;
	}

	return 0;

}

void print_command(char *command, int verbose, char *str)
{
	int j;

	if (!verbose)
		return;
	printf("%s", str);
	for(j=0; j<4; j++)
		printf("%X ", (unsigned char)command[j]);
	printf("\n");
}


/***********************************************
* usb_check_board
***********************************************/

usb_dev_handle *usb_check_board(struct usb_device *dev)
{
	usb_dev_handle *udev;
	char manufacturer[256];
	char product[256];

	udev = usb_open(dev);
	if (!udev || !dev->descriptor.iManufacturer) {
		usb_close (udev);
		return NULL;
	}

	if ( usb_get_string_simple(udev, dev->descriptor.iManufacturer,
manufacturer, sizeof(manufacturer)) < 0) {
		usb_close (udev);
		return NULL;
	}
```

```
	if ( usb_get_string_simple(udev, dev->descriptor.iProduct, product,
sizeof(product)) < 0) {
		usb_close (udev);
		return NULL;
	}

	if (strcmp(manufacturer, MANUFACTURER) != 0 || strcmp(product,
PRODUCT) != 0) {
		usb_close (udev);
		return NULL;
	}
	return udev;
}

/***********************************
* ehas_board_init
***********************************/

usb_dev_handle *ehas_board_init(int board, int verbose, char *err_buffer)
{
	struct usb_bus *bus;
	struct usb_device *dev;
	usb_dev_handle *udev;
	int n, claim;
	char buffer[256];
	char command_id[4] = {USBC_BOARD_ID, 0, 0, 0};

	usb_init();
	usb_find_busses();
	usb_find_devices();

	/* Seek EHAS Boards */
	for (bus = usb_busses; bus; bus = bus->next) {
		for (dev = bus->devices; dev; dev = dev->next) {

			if ( (udev = usb_check_board(dev)) == NULL )
				continue;

			/* Try to claim the inferface. 1 sec before timeout */

			for(n = 0; n < 100; n++) {
				if ( ( claim = usb_claim_interface(udev, 0)) >= 0)
					break;
				usleep(10000);
			}

			if (claim < 0) {
					sprintf(err_buffer, "usb_claim_interface
error");
					return NULL;
			}

			/* Empty USB incoming packets */
			while (usb_interrupt_read(udev, 1, buffer, 256, 1) >=
4);
```

```
                  if ( (n = ehas_board_send(udev, command_id, 4, 0,
buffer, buffer)) < 0) {
                        sprintf(err_buffer, "ID command rejected");
                        return NULL;
                  }

                  if (verbose)
                        printf("id #%d found\n", n);

                  if (board != n) {
                        ehas_board_close(udev);
                        continue;
                  }

                  if (verbose)
                        printf("id #%d selected\n", n);

                  return udev;
            }
      }
      sprintf(err_buffer, "board not found");
      return NULL;
}


/************************************
* ehas_board_send
************************************/

int ehas_board_send(usb_dev_handle *udev, char *command, int
command_length, int verbose, char *out_buffer, char *err_buffer)
{
      int i, k, n, retval = 0, serial_length;
      unsigned char buffer[256];
      unsigned char global_buffer[256];
      unsigned char serial_buffer[256];

      for(i=0; i<command_length; i+=4) {

            print_command(command + i, verbose, "send: ");

            if ( (n = usb_interrupt_write(udev, 1, command+i, 4, 5000)) <
4) {
                  sprintf(err_buffer, "write error: %d", n);
                  return -1;
            }

            if (usb_response_read(udev, buffer, err_buffer) )
                  return -1;

            memcpy(global_buffer + i, buffer, 4);

            print_command(buffer, verbose, "recv: ");
      }

      if (command[0] == USBC_GET_VERSION) {
            retval = (buffer[2] << 8) + buffer[1];
```

```
	}

	else if (command[0] == USBC_SET_DIGITAL || command[0] ==
USBC_GET_DIGITAL || command[0] == USBC_GET_ANALOG || command[0] ==
USBC_BOARD_ID) {
		if (command[0] == USBC_GET_ANALOG && command[1] ==
USBC_GATE_SWR_DIR && command[1+4] == USBC_GATE_SWR_INV) {
			out_buffer[0] = global_buffer[2];
			out_buffer[1] = global_buffer[2+4];
			retval = 0;
		}
		else
			retval = buffer[2];
	}

	else if (command[0] == USBC_TX_SERIAL) {
		serial_length = buffer[2];
		k = 0;
		i = 1;
		while (serial_length > 0)  {

			command[0] = USBC_RX_SERIAL;
			command[1] = i++;
			command[2] = command[3] = 0;

			print_command(command, verbose, "send: ");
			if ( (n = usb_interrupt_write(udev, 1, command, 4,
1000)) < 4) {
			    sprintf(err_buffer, "write error: %d", n);
			    return -1;
			}

			if (usb_response_read(udev, buffer, err_buffer) )
			    return -1;

			print_command(buffer, verbose, "recv: ");
			serial_buffer[k++] = buffer[2];
			serial_buffer[k++] = buffer[3];
			serial_length -= 2;
		}
		retval = k;
		memcpy(out_buffer, serial_buffer, k);
	}
	return (retval);
}

/************************************
* ehas_board_close
************************************/

void ehas_board_close(usb_dev_handle *udev)
{
	usb_release_interface(udev, 0);
	usb_close (udev);
}
```

CABECERA "usb.h"

```
#include <usb.h>

/************************************************************************/
/* USB Commands
                                        */

/*        0: Get Board Version.         00 XX XX XX                  -> 00 SUBVERSION VERSION 00   */

/*        1: Get Digital Inputs.        01 GATE XX XX                -> 01 GATE VALUE 00                                */

/*        2: Get Analog Inputs.         02 AN_GATE XX XX    -> 02 AN_GATE VALUE 00                          */

/*        3: Set Digital Outputs. 03 GATE VALUE XX          -> 03 GATE VALUE 00                             */

/*        4: RS232 TX + Init RX.  04 00 LENGTH XX           -> 04 00 LENGTH 00                              */
/*                                                                04 01 D1 D2                             -> 04 01 D1 D2
                                        */
/*                                                                04 02 D3 D4                             -> 04 02 D3 D4
                                        */
/*                                                                        ...
                                                                        */
/*                                                                04 NN DL-1 DL                -> 04 NN DL-1 DL
                          */

/*        5: RS232 RX.                  05 00 XX XX                           -> 05 00 LENGTH 00
                */
/*                                                                05 01 XX XX                             -> 05 01 D1 D2
                                        */
/*                                                                05 02 XX XX                             -> 05 02 D3 D4
                                        */
/*                                                                        ...
                                                                        */
/*                                                                05 NN XX XX                             -> 05 NN DL-1
DL                                      */

/*        6: Board ID (0,1,2,3)         06 XX XX XX                           -> 06 00 ID 00
                */

/*        On error returns: FF COMMAND ERROR 00
                */

/************************************************************************/

/* C functions */

usb_dev_handle *ehas_board_init(int board, int verbose, char
*err_buffer);
int ehas_board_send(usb_dev_handle *udev, char *command, int
command_length, int verbose, char *out_buffer, char *err_buffer);
void ehas_board_close(usb_dev_handle *udev);


/* Define USB Errors */

#define COMMAND_UNKNOWN       1
#define GATE_UNKNOWN          2
#define RS232_WRONG_INDEX     3
#define RS232_WRONG_LENGTH    4

/* RS232 Buffer size */
#define RS232_MAX_BUFFER       64

#define USBC_GET_VERSION      0
#define USBC_GET_DIGITAL      1
#define USBC_GET_ANALOG       2
#define USBC_SET_DIGITAL      3
#define USBC_TX_SERIAL        4
#define USBC_RX_SERIAL        5
#define USBC_BOARD_ID         6
```

```
/* Output */

#define USBC_GATE_LED         0
#define USBC_GATE_BUZZER      1
#define USBC_GATE_FAN         2
#define USBC_GATE_PTT         3
#define USBC_GATE_INVERT      4
#define USBC_GATE_DATAC       5
#define USBC_GATE_IGNITION    6

/* Analog Input */

#define USBC_GATE_BATTERY     16
#define USBC_GATE_TEMP        17
#define USBC_GATE_SWR_DIR     18
#define USBC_GATE_SWR_INV     19
#define USBC_GATE_ADC_FREE    20

/* Digital Input */

#define USBC_GATE_RADIO_STA   32
#define USBC_GATE_BUTTON      33


/* Virtual */
#define USBC_GATE_SWR         64
```

APLICACIÓN EN WINDOWS

PROGRAMA DESARROLLADO EN C++ usando el driver “testusb.sys”

```
// Windows console program to communicate with
// the Microchip USB PIC16C745/765
// Program developed by Him Cansaya based in Erik Center code

#include "windows.h"
#include "stdio.h"
int main(int argc, char* argv[])
      {
      unsigned char data[4];
      unsigned char data2[4];
      float data3[4];
      DWORD junk;
      char temp;
      HANDLE hdevice = CreateFile("\\\\.\\testusb2", GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
      if (hdevice == INVALID_HANDLE_VALUE)
            {
            printf("No se puede conectar con el dispositivo - error
%d\n", GetLastError());
            return 1;
            }
while(1){
            printf("encendido (H) y apagado (L) de los 4 leds: ");
            scanf("%c%c%c%c", &data[0],&data[1],&data[2],&data[3]);

            while (( temp = getchar() ) != '\n');
                  if (WriteFile(hdevice, data, 4, &junk, NULL))
                        printf("\n%d bytes fueron enviados al
dispositivo.\n", junk);
                  else
                  {
                        printf("Error %d intentando escribir los datos\n",
GetLastError());
                        break;
                  }

                  if (ReadFile(hdevice, data2, 4, &junk, NULL))
                  {

                        printf("bytes recibidos: ");
                        for (int j = 0; j < 4; j++)
                        {     printf("\nCanal %d: %d",j,data2[j]);
                        }

                        printf("\nEl valor de conversion es: ");
                        for (int i = 0; i < 4; i++)
                        {     data3[i] = (4.925 * data2[i]);
                              data3[i] = data3[i]/255;
                              printf("\nCanal %d: %3.3f
Voltios",i,data3[i]);
                        }
```

```
                        data3[3]=(325*data3[3]-925)/2;

                        printf("\nCanal 3: %3.2f Grados",data3[3]);

                        printf("\n\nCNTL C  para salir\n\n");
                }
                else
                {
                        printf("error\n\n");
                }
        }
        CloseHandle(hdevice);
        return 0;
}
```

CÓDIGOS FUENTE DEL DRIVER PARA SER COMPILADOS USANDO EL WINDDK EN C++

PROGRAMA “DriverEntry.cpp”

```
#include "stddcls.h"
#include "driver.h"
#include <initguid.h>
#include "guids.h"


NTSTATUS AddDevice(IN PDRIVER_OBJECT DriverObject, IN PDEVICE_OBJECT
pdo);
VOID DriverUnload(IN PDRIVER_OBJECT fdo);
NTSTATUS OnRequestComplete(IN PDEVICE_OBJECT fdo, IN PIRP Irp, IN PKEVENT
pev);

BOOLEAN IsWin98();
BOOLEAN win98 = FALSE;

UNICODE_STRING servkey;

#pragma INITCODE

extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
      IN PUNICODE_STRING RegistryPath)
      {                                       // DriverEntry

      // Insist that OS support at least the WDM level of the DDK we use

      if (!IoIsWdmVersionAvailable(1, 0))
            return STATUS_UNSUCCESSFUL;

      // See if we're running under Win98 or NT:

      win98 = IsWin98();


      // Save the name of the service key

      servkey.Buffer = (PWSTR) ExAllocatePool(PagedPool, RegistryPath-
>Length + sizeof(WCHAR));
      if (!servkey.Buffer)
            return STATUS_INSUFFICIENT_RESOURCES;
      servkey.MaximumLength = RegistryPath->Length + sizeof(WCHAR);
      RtlCopyUnicodeString(&servkey, RegistryPath);

      // Initialize function pointers

      DriverObject->DriverUnload = DriverUnload;
      DriverObject->DriverExtension->AddDevice = AddDevice;
      DriverObject->MajorFunction[IRP_MJ_CREATE] = DispatchCreate;
      DriverObject->MajorFunction[IRP_MJ_CLOSE] = DispatchClose;
      DriverObject->MajorFunction[IRP_MJ_READ] = DispatchReadWrite;
      DriverObject->MajorFunction[IRP_MJ_WRITE] = DispatchReadWrite;
```

```
      DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] =
DispatchInternalControl;
      DriverObject->MajorFunction[IRP_MJ_POWER] = DispatchPower;
      DriverObject->MajorFunction[IRP_MJ_PNP] = DispatchPnp;

      return STATUS_SUCCESS;
      }                                          // DriverEntry


#pragma PAGEDCODE

VOID DriverUnload(IN PDRIVER_OBJECT DriverObject)
      {                                          // DriverUnload
      PAGED_CODE();
      RtlFreeUnicodeString(&servkey);
      }                                          // DriverUnload


NTSTATUS AddDevice(IN PDRIVER_OBJECT DriverObject, IN PDEVICE_OBJECT pdo)
      {                                          // AddDevice
      PAGED_CODE();

      NTSTATUS status;

      // Create a functional device object to represent the hardware
we're managing.

      PDEVICE_OBJECT fdo;
      #define xsize sizeof(DEVICE_EXTENSION)
      UNICODE_STRING ifname;
      RtlInitUnicodeString(&ifname, L"\\DosDevices\\testusb2");
      status = IoCreateDevice(DriverObject, xsize, &ifname,
            FILE_DEVICE_UNKNOWN, FILE_DEVICE_SECURE_OPEN, FALSE, &fdo);
      if (!NT_SUCCESS(status))
            return status;              // can't create device object
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // From this point forward, any error will have side effects that
need to
      // be cleaned up. Using a try-finally block allows us to modify the
program
      // easily without losing track of the side effects.

      __try
            {                                    // finish initialization
            pdx->DeviceObject = fdo;
            pdx->Pdo = pdo;
            IoInitializeRemoveLock(&pdx->RemoveLock, 0, 0, 0);
            pdx->state = STOPPED;        // device starts in the stopped
state

            // Declare the buffering method we'll use for read/write
requests

            fdo->Flags |= DO_DIRECT_IO;

            // Link our device object into the stack leading to the PDO
```

```
            pdx->LowerDeviceObject = IoAttachDeviceToDeviceStack(fdo,
pdo);
            if (!pdx->LowerDeviceObject)
                  {                                     // can't attach
device
                  status = STATUS_DEVICE_REMOVED;
                  __leave;
                  }                                     // can't attach
device

            // Set power management flags in the device object

            fdo->Flags |= DO_POWER_PAGABLE;

            // Register a device interface

            status = IoRegisterDeviceInterface(pdo,
&GUID_INTERFACE_TESTUSB2, NULL, &pdx->ifname);
            if (!NT_SUCCESS(status))
                  __leave;                          // unable to register
interface

            // Indicate that our initial power state is D0 (fully on).
Also indicate that
            // we have a pagable power handler (otherwise, we'll never
get idle shutdown
            // messages!)

            pdx->syspower = PowerSystemWorking;
            pdx->devpower = PowerDeviceD0;
            POWER_STATE state;
            state.DeviceState = PowerDeviceD0;
            PoSetPowerState(fdo, DevicePowerState, state);

            // Clear the "initializing" flag so that we can get IRPs

            fdo->Flags &= ~DO_DEVICE_INITIALIZING;
            }                                       // finish initialization
      __finally
            {                                       // cleanup side effects
            if (!NT_SUCCESS(status))
                  {                                 // need to cleanup
                  if (pdx->ifname.Buffer)
                        RtlFreeUnicodeString(&pdx->ifname);
                  if (pdx->LowerDeviceObject)
                        IoDetachDevice(pdx->LowerDeviceObject);
                  IoDeleteDevice(fdo);
                  }                                 // need to cleanup
            }                                       // cleanup side effects

      return status;
      }


#pragma LOCKEDCODE
```

```
NTSTATUS CompleteRequest(IN PIRP Irp, IN NTSTATUS status, IN ULONG_PTR
info)
      {                                                 // CompleteRequest
      Irp->IoStatus.Status = status;
      Irp->IoStatus.Information = info;
      IoCompleteRequest(Irp, IO_NO_INCREMENT);
      return status;
      }
NTSTATUS CompleteRequest(IN PIRP Irp, IN NTSTATUS status)
      {                                                 // CompleteRequest
      Irp->IoStatus.Status = status;
      IoCompleteRequest(Irp, IO_NO_INCREMENT);
      return status;
      }

#pragma PAGEDCODE

NTSTATUS ForwardAndWait(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                                 // ForwardAndWait
      ASSERT(KeGetCurrentIrql() == PASSIVE_LEVEL);
      PAGED_CODE();

      KEVENT event;
      KeInitializeEvent(&event, NotificationEvent, FALSE);

      IoCopyCurrentIrpStackLocationToNext(Irp);
      IoSetCompletionRoutine(Irp, (PIO_COMPLETION_ROUTINE)
OnRequestComplete,
            (PVOID) &event, TRUE, TRUE, TRUE);

      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      IoCallDriver(pdx->LowerDeviceObject, Irp);
      KeWaitForSingleObject(&event, Executive, KernelMode, FALSE, NULL);
      return Irp->IoStatus.Status;
      }



#pragma LOCKEDCODE

NTSTATUS OnRequestComplete(IN PDEVICE_OBJECT fdo, IN PIRP Irp, IN PKEVENT
pev)
      {                                                 // OnRequestComplete
      KeSetEvent(pev, 0, FALSE);
      return STATUS_MORE_PROCESSING_REQUIRED;
      }



VOID EnableAllInterfaces(PDEVICE_EXTENSION pdx, BOOLEAN enable)
      {                                                 // EnableAllInterfaces
      IoSetDeviceInterfaceState(&pdx->ifname, enable);
      }



VOID DeregisterAllInterfaces(PDEVICE_EXTENSION pdx)
```

```
      {                                                 // DeregisterAllInterfaces
      IoSetDeviceInterfaceState(&pdx->ifname, FALSE);
      RtlFreeUnicodeString(&pdx->ifname);
      }



#pragma PAGEDCODE

VOID RemoveDevice(IN PDEVICE_OBJECT fdo)
      {                                                 // RemoveDevice
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      NTSTATUS status;
      if (pdx->LowerDeviceObject)
            IoDetachDevice(pdx->LowerDeviceObject);
      IoDeleteDevice(fdo);
      }


#pragma INITCODE

BOOLEAN IsWin98()
      {                                                 // IsWin98
#ifdef _X86_

      // Windows 98 (including 2d ed) supports WDM version 1.0, whereas
Win2K
      // supports 1.10.

      return !IoIsWdmVersionAvailable(1, 0x10);
#else // not _X86_
      return FALSE;
#endif // not _X86_
      }



#pragma PAGEDCODE
NTSTATUS DispatchInternalControl(PDEVICE_OBJECT fdo, PIRP Irp)
      {                                                 // DispatchInternalControl
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      IoSkipCurrentIrpStackLocation(Irp);
      return IoCallDriver(pdx->LowerDeviceObject, Irp);
      }

NTSTATUS DispatchPower(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                                 // DispatchPower
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      // Just pass to lower driver
      PoStartNextPowerIrp( Irp);
      IoSkipCurrentIrpStackLocation(Irp);
      return PoCallDriver( pdx->LowerDeviceObject, Irp);

      }
```

```
#pragma PAGEDCODE

VOID AbortPipe(PDEVICE_OBJECT fdo, USBD_PIPE_HANDLE hpipe)
      {                                          // AbortPipe
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      URB urb;
      urb.UrbHeader.Length = (USHORT) sizeof(_URB_PIPE_REQUEST);
      urb.UrbHeader.Function = URB_FUNCTION_ABORT_PIPE;
      urb.UrbPipeRequest.PipeHandle = hpipe;
      NTSTATUS status = SendAwaitUrb(fdo, &urb);
      }


#pragma PAGEDCODE

NTSTATUS DispatchCreate(PDEVICE_OBJECT fdo, PIRP Irp)
      {                                          // DispatchCreate
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);

      // Claim the remove lock in Win2K so that removal waits until the
      // handle closes. Don't do this in Win98, however, because this
      // device might be removed by surprise with handles open, whereupon
      // we'll deadlock in HandleRemoveDevice waiting for a close that
      // can never happen because we can't run the user-mode code that
      // would do the close.

      NTSTATUS status;
      if (win98)
            status = STATUS_SUCCESS;
      else
            status = IoAcquireRemoveLock(&pdx->RemoveLock, stack-
>FileObject);
      if (NT_SUCCESS(status))
            InterlockedDecrement(&pdx->handles);
      return CompleteRequest(Irp, status, 0);
      }


#pragma PAGEDCODE

NTSTATUS DispatchClose(PDEVICE_OBJECT fdo, PIRP Irp)
      {                                          // DispatchClose
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
      if (InterlockedDecrement(&pdx->handles) == 0)
            {                                    // no more open handles
            }                                    // no more open handles

      // Release the remove lock to match the acquisition done in
DispatchCreate

      if (!win98)
```

```
            IoReleaseRemoveLock(&pdx->RemoveLock, stack->FileObject);

      return CompleteRequest(Irp, STATUS_SUCCESS, 0);
      }



#pragma PAGEDCODE

NTSTATUS DispatchReadWrite(PDEVICE_OBJECT fdo, PIRP Irp)
      {                                         // DispatchReadWrite
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      NTSTATUS status = IoAcquireRemoveLock(&pdx->RemoveLock, Irp);
      if (!NT_SUCCESS(status))
            return CompleteRequest(Irp, status, 0);

      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
      BOOLEAN read = stack->MajorFunction == IRP_MJ_READ;

      USBD_PIPE_HANDLE hpipe = read ? pdx->endptwo : pdx->endpone;
      LONG haderr;
      if (read)
            haderr = InterlockedExchange(&pdx->inerror, 0);
      else
            haderr = InterlockedExchange(&pdx->outerror, 0);
      if (haderr && !NT_SUCCESS(ResetPipe(fdo, hpipe)))
            ResetDevice(fdo);



      USHORT UrbSize = sizeof(struct _URB_BULK_OR_INTERRUPT_TRANSFER);
      PURB urb = (PURB)ExAllocatePool(NonPagedPool, UrbSize);
      if( urb==NULL)
      {
            return STATUS_INSUFFICIENT_RESOURCES;
      }

      ULONG_PTR va = (ULONG_PTR) MmGetMdlVirtualAddress(Irp->MdlAddress);
      ULONG urbflags =  USBD_SHORT_TRANSFER_OK | (read ?
USBD_TRANSFER_DIRECTION_IN : USBD_TRANSFER_DIRECTION_OUT);
      UsbBuildInterruptOrBulkTransferRequest(urb,
sizeof(_URB_BULK_OR_INTERRUPT_TRANSFER),
            hpipe, NULL, Irp->MdlAddress, 4, urbflags, NULL);

      // Use the original Read or Write IRP as a container for the URB

      stack = IoGetNextIrpStackLocation(Irp);
      stack->MajorFunction = IRP_MJ_INTERNAL_DEVICE_CONTROL;
      stack->Parameters.Others.Argument1 = (PVOID) (PURB) urb;
      stack->Parameters.DeviceIoControl.IoControlCode =
IOCTL_INTERNAL_USB_SUBMIT_URB;

      IoSetCompletionRoutine(Irp, (PIO_COMPLETION_ROUTINE)
OnReadWriteComplete,
            (PVOID) urb, TRUE, TRUE, TRUE);
```

```
      IoMarkIrpPending(Irp);
      status = IoCallDriver(pdx->LowerDeviceObject, Irp);
      if (!NT_SUCCESS(status) && !NT_SUCCESS(ResetPipe(fdo, hpipe)))
            ResetDevice(fdo);
      return STATUS_PENDING;
}


#pragma LOCKEDCODE
      NTSTATUS OnReadWriteComplete(PDEVICE_OBJECT fdo, PIRP Irp, PURB
ctx)
      {                                          // OnReadWriteComplete
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      BOOLEAN read = (ctx->UrbBulkOrInterruptTransfer.TransferFlags &
USBD_TRANSFER_DIRECTION_IN) != 0;
      NTSTATUS status = Irp->IoStatus.Status;

      if (NT_SUCCESS(status))
            Irp->IoStatus.Information = ctx-
>UrbBulkOrInterruptTransfer.TransferBufferLength;
      else
            {                              // had an error
            if (read)
                  InterlockedIncrement(&pdx->inerror);
            else
                  InterlockedIncrement(&pdx->outerror);

            }                              // had an error
      ExFreePool(ctx);
      IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
      return status;
      }


#pragma PAGEDCODE

NTSTATUS GetStringDescriptor(PDEVICE_OBJECT fdo, UCHAR istring,
PUNICODE_STRING s)
      {                                          // GetStringDescriptor
      NTSTATUS status;
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      URB urb;

      UCHAR data[256];              // maximum-length buffer

      // If this is the first time here, read string descriptor zero and
arbitrarily select
      // the first language identifer as the one to use in subsequent
get-descriptor calls.

      if (!pdx->langid)
            {                                    // determine default
language id
            UsbBuildGetDescriptorRequest(&urb,
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST), USB_STRING_DESCRIPTOR_TYPE,
                  0, 0, data, NULL, sizeof(data), NULL);
            status = SendAwaitUrb(fdo, &urb);
```

```
            if (!NT_SUCCESS(status))
                  return status;
            pdx->langid = *(LANGID*)(data + 2);
            }                                       // determine default
language id

      // Fetch the designated string descriptor.

      UsbBuildGetDescriptorRequest(&urb,
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST), USB_STRING_DESCRIPTOR_TYPE,
            istring, pdx->langid, data, NULL, sizeof(data), NULL);
      status = SendAwaitUrb(fdo, &urb);
      if (!NT_SUCCESS(status))
            return status;

      ULONG nchars = (data[0] - 2) / 2;
      PWSTR p = (PWSTR) ExAllocatePool(PagedPool, data[0]);
      if (!p)
            return STATUS_INSUFFICIENT_RESOURCES;

      memcpy(p, data + 2, nchars*2);
      p[nchars] = 0;

      s->Length = (USHORT) (2 * nchars);
      s->MaximumLength = (USHORT) ((2 * nchars) + 2);
      s->Buffer = p;

      return STATUS_SUCCESS;
      }



#pragma PAGEDCODE

VOID ResetDevice(PDEVICE_OBJECT fdo)
      {                                             // ResetDevice
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      KEVENT event;
      KeInitializeEvent(&event, NotificationEvent, FALSE);
      IO_STATUS_BLOCK iostatus;

      PIRP Irp =
IoBuildDeviceIoControlRequest(IOCTL_INTERNAL_USB_RESET_PORT,
            pdx->LowerDeviceObject, NULL, 0, NULL, 0, TRUE, &event,
&iostatus);
      if (!Irp)
            return;

      NTSTATUS status = IoCallDriver(pdx->LowerDeviceObject, Irp);
      if (status == STATUS_PENDING)
            {
            KeWaitForSingleObject(&event, Executive, KernelMode, FALSE,
NULL);
            status = iostatus.Status;
            }
```

```
	}


#pragma PAGEDCODE

NTSTATUS ResetPipe(PDEVICE_OBJECT fdo, USBD_PIPE_HANDLE hpipe)
	{                                                  // ResetPipe
	PAGED_CODE();
	PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
	URB urb;

	urb.UrbHeader.Length = (USHORT) sizeof(_URB_PIPE_REQUEST);
	urb.UrbHeader.Function = URB_FUNCTION_RESET_PIPE;
	urb.UrbPipeRequest.PipeHandle = hpipe;

	NTSTATUS status = SendAwaitUrb(fdo, &urb);
	return status;
	}



NTSTATUS SendAwaitUrb(PDEVICE_OBJECT fdo, PURB urb)
	{                                                  // SendAwaitUrb
	PAGED_CODE();
	ASSERT(KeGetCurrentIrql() == PASSIVE_LEVEL);
	PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

	KEVENT event;
	KeInitializeEvent(&event, NotificationEvent, FALSE);

	IO_STATUS_BLOCK iostatus;
	PIRP Irp =
IoBuildDeviceIoControlRequest(IOCTL_INTERNAL_USB_SUBMIT_URB,
		pdx->LowerDeviceObject, NULL, 0, NULL, 0, TRUE, &event,
&iostatus);

	if (!Irp)
		return STATUS_INSUFFICIENT_RESOURCES;
	PIO_STACK_LOCATION stack = IoGetNextIrpStackLocation(Irp);
	stack->Parameters.Others.Argument1 = (PVOID) urb;
	NTSTATUS status = IoCallDriver(pdx->LowerDeviceObject, Irp);
	if (status == STATUS_PENDING)
		{
		KeWaitForSingleObject(&event, Executive, KernelMode, FALSE,
NULL);
		status = iostatus.Status;
		}
	return status;
	}



NTSTATUS StartDevice(PDEVICE_OBJECT fdo)
	{                                                  // StartDevice
	PAGED_CODE();
```

```
	NTSTATUS status;
	PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

	URB urb;                                // URB for use in this
subroutine

	// Read our device descriptor. The only real purpose to this would
be to find out how many
	// configurations there are so we can read their descriptors. In
this simplest of examples,
	// there's only one configuration.

	UsbBuildGetDescriptorRequest(&urb,
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST), USB_DEVICE_DESCRIPTOR_TYPE,
		0, 0, &pdx->dd, NULL, sizeof(pdx->dd), NULL);
	status = SendAwaitUrb(fdo, &urb);
	if (!NT_SUCCESS(status))
		return status;

	// Read the descriptor of the first configuration. This requires
two steps. The first step
	// reads the fixed-size configuration descriptor alone. The second
step reads the
	// configuration descriptor plus all imbedded interface and
endpoint descriptors.

	USB_CONFIGURATION_DESCRIPTOR tcd;
	UsbBuildGetDescriptorRequest(&urb,
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
USB_CONFIGURATION_DESCRIPTOR_TYPE,
		0, 0, &tcd, NULL, sizeof(tcd), NULL);
	status = SendAwaitUrb(fdo, &urb);
	if (!NT_SUCCESS(status))
		return status;

	ULONG size = tcd.wTotalLength;
	PUSB_CONFIGURATION_DESCRIPTOR pcd = (PUSB_CONFIGURATION_DESCRIPTOR)
ExAllocatePool(NonPagedPool, size);
	if (!pcd)
		return STATUS_INSUFFICIENT_RESOURCES;

	__try
		{
		UsbBuildGetDescriptorRequest(&urb,
sizeof(_URB_CONTROL_DESCRIPTOR_REQUEST),
USB_CONFIGURATION_DESCRIPTOR_TYPE,
			0, 0, pcd, NULL, size, NULL);
		status = SendAwaitUrb(fdo, &urb);
		if (!NT_SUCCESS(status))
			return status;

		// Locate the descriptor for the one and only interface we
expect to find

		PUSB_INTERFACE_DESCRIPTOR pid =
USBD_ParseConfigurationDescriptorEx(pcd, pcd,
			-1, -1, -1, -1, -1);
```

```
            ASSERT(pid);


            // Create a URB to use in selecting a configuration.

            USBD_INTERFACE_LIST_ENTRY interfaces[2] = {
                  {pid, NULL},
                  {NULL, NULL},           // fence to terminate the array
                  };

            PURB selurb = USBD_CreateConfigurationRequestEx(pcd,
interfaces);
            if (!selurb)
                  return STATUS_INSUFFICIENT_RESOURCES;

            __try
                  {

                  // Verify that the interface describes exactly the
endpoints we expect

                  if (pid->bNumEndpoints != 2)
                        return STATUS_DEVICE_CONFIGURATION_ERROR;

                  PUSB_ENDPOINT_DESCRIPTOR ped =
(PUSB_ENDPOINT_DESCRIPTOR) pid;
                  ped = (PUSB_ENDPOINT_DESCRIPTOR)
USBD_ParseDescriptors(pcd, tcd.wTotalLength, ped,
USB_ENDPOINT_DESCRIPTOR_TYPE);
                  if (!ped || ped->bEndpointAddress != 0x82 || ped-
>bmAttributes != USB_ENDPOINT_TYPE_INTERRUPT || ped->wMaxPacketSize != 4)
                        return STATUS_DEVICE_CONFIGURATION_ERROR;

                  ++ped;
                  ped = (PUSB_ENDPOINT_DESCRIPTOR)
USBD_ParseDescriptors(pcd, tcd.wTotalLength, ped,
USB_ENDPOINT_DESCRIPTOR_TYPE);
                  if (!ped || ped->bEndpointAddress != 0x1 || ped-
>bmAttributes != USB_ENDPOINT_TYPE_INTERRUPT || ped->wMaxPacketSize != 4)
                        return STATUS_DEVICE_CONFIGURATION_ERROR;

                  ++ped;
                  PUSBD_INTERFACE_INFORMATION pii =
interfaces[0].Interface;

                  // Initialize the maximum transfer size for each of the
endpoints
                  // TODO remove these statements if you're happy with
the default
                  // value provided by USBD.

                  pii->Pipes[0].MaximumTransferSize = 4;
                  pii->Pipes[1].MaximumTransferSize = 4;

                  // Submit the set-configuration request

                  status = SendAwaitUrb(fdo, selurb);
```

```
                  if (!NT_SUCCESS(status))
                        return status;


                  // Save the configuration and pipe handles

                  pdx->hconfig = selurb-
>UrbSelectConfiguration.ConfigurationHandle;
                  pdx->endptwo = pii->Pipes[0].PipeHandle;
                  pdx->endpone = pii->Pipes[1].PipeHandle;

                  // TODO If you have an interrupt endpoint, now would be
the time to
                  // create an IRP and URB with which to poll it
continuously

                  // Transfer ownership of the configuration descriptor
to the device extension

                  pdx->pcd = pcd;
                  pcd = NULL;
                  }
            __finally
                  {
                  ExFreePool(selurb);
                  }

            }
      __finally
            {
            if (pcd)
                  ExFreePool(pcd);
            }

      return STATUS_SUCCESS;
      }                                       // StartDevice


#pragma PAGEDCODE

VOID StopDevice(IN PDEVICE_OBJECT fdo, BOOLEAN oktouch /* = FALSE */)
      {                                       // StopDevice
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // If it's okay to touch our hardware (i.e., we're processing an
IRP_MN_STOP_DEVICE),
      // deconfigure the device.

      if (oktouch)
            {                                 // deconfigure device
            URB urb;
            UsbBuildSelectConfigurationRequest(&urb,
sizeof(_URB_SELECT_CONFIGURATION), NULL);
            NTSTATUS status = SendAwaitUrb(fdo, &urb);
            }
```

```
      if (pdx->pcd)
            ExFreePool(pdx->pcd);
      pdx->pcd = NULL;
      }                                          // StopDevice


NTSTATUS DefaultPnpHandler(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleCancelRemove(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleCancelStop(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleQueryCapabilities(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleQueryRemove(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleQueryStop(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleRemoveDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleStartDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleStopDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS HandleSurpriseRemoval(IN PDEVICE_OBJECT fdo, IN PIRP Irp);

//////////////////////////////////////////////////////////////////////////
//////

#pragma PAGEDCODE

NTSTATUS DispatchPnp(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                          // DispatchPnp
      PAGED_CODE();
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      NTSTATUS status = IoAcquireRemoveLock(&pdx->RemoveLock, Irp);
      if (!NT_SUCCESS(status))
            return CompleteRequest(Irp, status);

      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
      ASSERT(stack->MajorFunction == IRP_MJ_PNP);

      static NTSTATUS (*fcntab[])(IN PDEVICE_OBJECT fdo, IN PIRP Irp) = {
            HandleStartDevice,            // IRP_MN_START_DEVICE
            HandleQueryRemove,            // IRP_MN_QUERY_REMOVE_DEVICE
            HandleRemoveDevice,           // IRP_MN_REMOVE_DEVICE
            HandleCancelRemove,           // IRP_MN_CANCEL_REMOVE_DEVICE
            HandleStopDevice,       // IRP_MN_STOP_DEVICE
            HandleQueryStop,        // IRP_MN_QUERY_STOP_DEVICE
            HandleCancelStop,       // IRP_MN_CANCEL_STOP_DEVICE
            DefaultPnpHandler,            // IRP_MN_QUERY_DEVICE_RELATIONS
            DefaultPnpHandler,            // IRP_MN_QUERY_INTERFACE
            HandleQueryCapabilities,// IRP_MN_QUERY_CAPABILITIES
            DefaultPnpHandler,            // IRP_MN_QUERY_RESOURCES
            DefaultPnpHandler,            //
IRP_MN_QUERY_RESOURCE_REQUIREMENTS
            DefaultPnpHandler,            // IRP_MN_QUERY_DEVICE_TEXT
            DefaultPnpHandler,            //
IRP_MN_FILTER_RESOURCE_REQUIREMENTS
            DefaultPnpHandler,            //
            DefaultPnpHandler,            // IRP_MN_READ_CONFIG
            DefaultPnpHandler,            // IRP_MN_WRITE_CONFIG
            DefaultPnpHandler,            // IRP_MN_EJECT
            DefaultPnpHandler,            // IRP_MN_SET_LOCK
            DefaultPnpHandler,            // IRP_MN_QUERY_ID
            DefaultPnpHandler,            // IRP_MN_QUERY_PNP_DEVICE_STATE
```

```
            DefaultPnpHandler,            // IRP_MN_QUERY_BUS_INFORMATION
            DefaultPnpHandler,            //
IRP_MN_DEVICE_USAGE_NOTIFICATION
            HandleSurpriseRemoval,  // IRP_MN_SURPRISE_REMOVAL
            };

      ULONG fcn = stack->MinorFunction;
      if (fcn >= arraysize(fcntab))
            {                                     // unknown function
            status = DefaultPnpHandler(fdo, Irp); // some function we
don't know about
            IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
            return status;
            }
      status = (*fcntab[fcn])(fdo, Irp);
      if (fcn != IRP_MN_REMOVE_DEVICE)
            IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
      return status;
      }

NTSTATUS DefaultPnpHandler(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                       // DefaultPnpHandler
      IoSkipCurrentIrpStackLocation(Irp);
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      return IoCallDriver(pdx->LowerDeviceObject, Irp);
      }



NTSTATUS HandleCancelRemove(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                       // HandleCancelRemove
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_CANCEL_REMOVE_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      if (pdx->state == PENDINGREMOVE)
            {                                   // we succeeded earlier
query

            // Lower-level drivers are presumably in the pending-remove
state as
            // well, so we need to tell them that the remove has been
cancelled
            // before we start sending IRPs down to them.

            NTSTATUS status = ForwardAndWait(fdo, Irp); // wait for lower
layers
            if (NT_SUCCESS(status))
                  {                             // completed successfully
                  if ((pdx->state = pdx->prevstate) == WORKING)
                        {                       // back to working state
                        }                       // back to working state
                  }                             // completed successfully
            else
            {
            }
```

```
            return CompleteRequest(Irp, status);
            }                                       // we succeeded earlier
query

      return DefaultPnpHandler(fdo, Irp); // unexpected cancel
      }


NTSTATUS HandleCancelStop(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleCancelStop
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_CANCEL_STOP_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      if (pdx->state == PENDINGSTOP)
            {                                       // we succeeded earlier
query

            // Lower level drivers are presumably in the pending-stop
state as
            // well, so we need to tell them that the stop has been
cancelled
            // before we start sending IRPs down to them.

            NTSTATUS status = ForwardAndWait(fdo, Irp); // wait for lower
layers
            if (NT_SUCCESS(status))
                  {                                 // completed successfully
                  pdx->state = WORKING;
                  }                                 // completed successfully
            else
            {
            }

            return CompleteRequest(Irp, status);
            }                                       // we succeeded earlier
query

      return DefaultPnpHandler(fdo, Irp); // unexpected cancel
      }



NTSTATUS HandleQueryCapabilities(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleQueryCapabilities
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_QUERY_CAPABILITIES);
      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
      PDEVICE_CAPABILITIES pdc = stack-
>Parameters.DeviceCapabilities.Capabilities;
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // Check to be sure we know how to handle this version of the
capabilities structure
```

```
      if (pdc->Version < 1)
            return DefaultPnpHandler(fdo, Irp);

      NTSTATUS status = ForwardAndWait(fdo, Irp);
      if (NT_SUCCESS(status))
            {                                       // IRP succeeded
            stack = IoGetCurrentIrpStackLocation(Irp);
            pdc = stack->Parameters.DeviceCapabilities.Capabilities;

            // TODO Modify any capabilities that must be set on the way
back up

            pdc->SurpriseRemovalOK = TRUE;
            pdx->devcaps = *pdc;    // save capabilities for whoever needs
to see them
            }                                       // IRP succeeded

      return CompleteRequest(Irp, status);
      }



NTSTATUS HandleQueryRemove(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleQueryRemove
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_QUERY_REMOVE_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
      if (pdx->state == WORKING)
            {                                       // currently working

      #ifdef _X86_

            // Win98 doesn't check for open handles before allowing a
remove to proceed,
            // and it may deadlock in IoReleaseRemoveLockAndWait if
handles are still
            // open.

            if (win98 && pdx->DeviceObject->ReferenceCount)
                  return CompleteRequest(Irp, STATUS_DEVICE_BUSY);

      #endif
            }                                       // currently working

      // Save current state for restoration if the query gets cancelled.
      // (We can now be stopped or working)

      pdx->prevstate = pdx->state;
      pdx->state = PENDINGREMOVE;
      return DefaultPnpHandler(fdo, Irp);
      }



NTSTATUS HandleQueryStop(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
```

```
      {                                             // HandleQueryStop
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_QUERY_STOP_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // Boot devices may get this query before they even start, so check
to see
      // if we're in the WORKING state before doing anything.

      if (pdx->state != WORKING)
            return DefaultPnpHandler(fdo, Irp);

      pdx->state = PENDINGSTOP;
      return DefaultPnpHandler(fdo, Irp);
      }



NTSTATUS HandleRemoveDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleRemoveDevice
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_REMOVE_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // Cancel any queued IRPs and start rejecting new ones

      // Disable all device interfaces. This triggers PnP notifications
that will
      // allow apps to close their handles.

      DeregisterAllInterfaces(pdx);

      // Release our I/O resources

      StopDevice(fdo, pdx->state == WORKING);
      pdx->state = REMOVED;

      // Let lower-level drivers handle this request. Ignore whatever
      // result eventuates.

      NTSTATUS status = DefaultPnpHandler(fdo, Irp);

      // Wait for all claims against this device to vanish before
removing
      // the device object.

      IoReleaseRemoveLockAndWait(&pdx->RemoveLock, Irp);

      // Remove the device object

      RemoveDevice(fdo);
```

```
      return status;                        // lower-level completed
IoStatus already
      }


NTSTATUS HandleStartDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleStartDevice
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_START_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      NTSTATUS status = ForwardAndWait(fdo, Irp);
      if (!NT_SUCCESS(status))
            return CompleteRequest(Irp, status);

      PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      status = StartDevice(fdo);

      // While we were in the stopped state, we were stalling incoming
requests.
      // Now we can release any pending IRPs and start processing new
ones

      if (NT_SUCCESS(status))
            {                                       // started okay

            // Enable all registered device interfaces.

            EnableAllInterfaces(pdx, TRUE);

            pdx->state = WORKING;
            }                                       // started okay

      return CompleteRequest(Irp, status);
      }



NTSTATUS HandleStopDevice(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleStopDevice
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_STOP_DEVICE);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // We're supposed to always get a query before we're stopped, so
      // we should already be in the PENDINGSTOP state. There's a Win98
bug that
      // can sometimes cause us to get a STOP instead of a REMOVE, in
which case
      // we should start rejecting IRPs

      StopDevice(fdo, pdx->state == WORKING);
      pdx->state = STOPPED;
```

```
      return DefaultPnpHandler(fdo, Irp);
      }


NTSTATUS HandleSurpriseRemoval(IN PDEVICE_OBJECT fdo, IN PIRP Irp)
      {                                             // HandleSurpriseRemoval
      ASSERT(IoGetCurrentIrpStackLocation(Irp)->MinorFunction ==
IRP_MN_SURPRISE_REMOVAL);
      Irp->IoStatus.Status = STATUS_SUCCESS;    // flag that we handled
this IRP
      PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

      // Cancel any queued IRPs and start rejecting new ones
      EnableAllInterfaces(pdx, FALSE);

      BOOLEAN oktouch = pdx->state == WORKING;
      pdx->state = SURPRISEREMOVED;
      StopDevice(fdo, oktouch);

      return DefaultPnpHandler(fdo, Irp);
      }


#pragma PAGEDCODE

VOID InitializeRemoveLock(PREMOVE_LOCK lock, ULONG tag, ULONG minutes,
ULONG maxcount)
      {                                             // InitializeRemoveLock
      PAGED_CODE();
      KeInitializeEvent(&lock->evRemove, NotificationEvent, FALSE);
      lock->usage = 1;
      lock->removing = FALSE;
      }


#pragma LOCKEDCODE

NTSTATUS AcquireRemoveLock(PREMOVE_LOCK lock, PVOID tag)
      {                                             // AcquireRemoveLock
      LONG usage = InterlockedIncrement(&lock->usage);
      if (lock->removing)
            {                                       // removal in progress
            if (InterlockedDecrement(&lock->usage) == 0)
                  KeSetEvent(&lock->evRemove, 0, FALSE);
            return STATUS_DELETE_PENDING;
            }                                       // removal in progress
      return STATUS_SUCCESS;
      }


VOID ReleaseRemoveLock(PREMOVE_LOCK lock, PVOID tag)
      {                                             // ReleaseRemoveLock
      if (InterlockedDecrement(&lock->usage) == 0)
            KeSetEvent(&lock->evRemove, 0, FALSE);
```

```
      }


#pragma PAGEDCODE

VOID ReleaseRemoveLockAndWait(PREMOVE_LOCK lock, PVOID tag)
      {                                          //
ReleaseRemoveLockAndWait
      PAGED_CODE();
      lock->removing = TRUE;
      ReleaseRemoveLock(lock, tag);
      ReleaseRemoveLock(lock, NULL);
      KeWaitForSingleObject(&lock->evRemove, Executive, KernelMode,
FALSE, NULL);
      }
```

CABECERA “driver.h”

```
#ifndef DRIVER_H
#define DRIVER_H

#define DRIVERNAME "TESTUSB2"                  // for use in messages
#define LDRIVERNAME L"TESTUSB2"                     // for use in
UNICODE string constants

enum DEVSTATE {
      STOPPED,                                          // device
stopped
      WORKING,                                          // started and
working
      PENDINGSTOP,                                      // stop
pending
      PENDINGREMOVE,                                    // remove
pending
      SURPRISEREMOVED,                             // removed by
surprise
      REMOVED,                                          // removed
      };

typedef struct _DEVICE_EXTENSION {
      PDEVICE_OBJECT DeviceObject;             // device object this
extension belongs to
      PDEVICE_OBJECT LowerDeviceObject;        // next lower driver in
same stack
      PDEVICE_OBJECT Pdo;                               // the PDO
      IO_REMOVE_LOCK RemoveLock;                   // removal control
locking structure
      UNICODE_STRING ifname;                       // interface name
      DEVSTATE state;                                   // current
state of device
      DEVSTATE prevstate;                               // state prior
to removal query
```

```
      DEVICE_POWER_STATE devpower;              // current device power
state
      SYSTEM_POWER_STATE syspower;              // current system power
state
      DEVICE_CAPABILITIES devcaps;              // copy of most recent
device capabilities
      LONG handles;                                       // # open
handles
      USB_DEVICE_DESCRIPTOR dd;                        // device descriptor
      USBD_CONFIGURATION_HANDLE hconfig;        // selected configuration
handle
      PUSB_CONFIGURATION_DESCRIPTOR pcd;        // configuration
descriptor
      LANGID langid;                                      // default
language id for strings
      LONG inerror;                                       // nonzero if
had read error
      LONG outerror;                                      // nonzero if
had write error
      USBD_PIPE_HANDLE endptwo;
      USBD_PIPE_HANDLE endpone;

      } DEVICE_EXTENSION, *PDEVICE_EXTENSION;

VOID RemoveDevice(IN PDEVICE_OBJECT fdo);
NTSTATUS CompleteRequest(IN PIRP Irp, IN NTSTATUS status, IN ULONG_PTR
info);
NTSTATUS CompleteRequest(IN PIRP Irp, IN NTSTATUS status);
NTSTATUS ForwardAndWait(IN PDEVICE_OBJECT fdo, IN PIRP Irp);
NTSTATUS SendDeviceSetPower(PDEVICE_EXTENSION fdo, DEVICE_POWER_STATE
state, BOOLEAN wait = FALSE);
VOID SendAsyncNotification(PVOID context);
VOID EnableAllInterfaces(PDEVICE_EXTENSION pdx, BOOLEAN enable);
VOID DeregisterAllInterfaces(PDEVICE_EXTENSION pdx);
NTSTATUS SendAwaitUrb(PDEVICE_OBJECT fdo, PURB urb);
VOID AbortPipe(PDEVICE_OBJECT fdo, USBD_PIPE_HANDLE hpipe);
NTSTATUS ResetPipe(PDEVICE_OBJECT fdo, USBD_PIPE_HANDLE hpipe);
VOID ResetDevice(PDEVICE_OBJECT fdo);
NTSTATUS StartDevice(PDEVICE_OBJECT fdo);
VOID StopDevice(PDEVICE_OBJECT fdo, BOOLEAN oktouch = FALSE);

NTSTATUS DispatchCreate(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS DispatchClose(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS DispatchReadWrite(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS DispatchInternalControl(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS DispatchPower(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS DispatchPnp(PDEVICE_OBJECT fdo, PIRP Irp);
NTSTATUS OnReadWriteComplete(PDEVICE_OBJECT fdo, PIRP Irp, PURB ctx);

extern BOOLEAN win98;
extern UNICODE_STRING servkey;

#endif // DRIVER_H
```

National Semiconductor

May 1999

LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

General Description

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/˚K. With less than 1Ω dynamic impedance the device operates over a current range of 400 µA to 5 mA with virtually no change in performance. When calibrated at 25˚C the LM135 has typically less than 1˚C error over a 100˚C temperature range. Unlike other sensors the LM135 has a linear output.

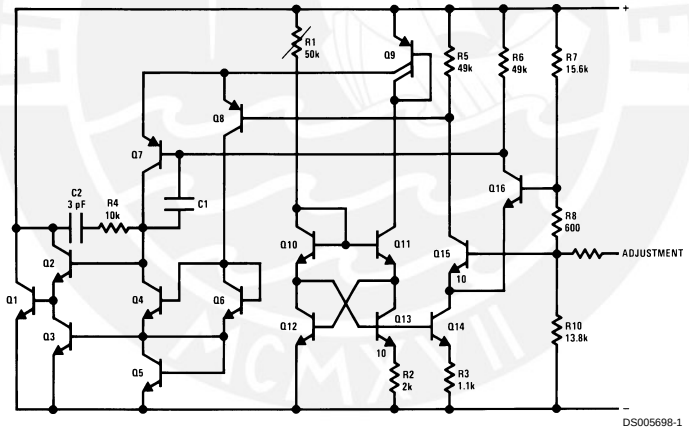
Applications for the LM135 include almost any type of temperature sensing over a −55˚C to +150˚C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

The LM135 operates over a −55˚C to +150˚C temperature range while the LM235 operates over a −40˚C to +125˚C temperature range. The LM335 operates from −40˚C to +100˚C. The LM135/LM235/LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

Features

- Directly calibrated in ˚Kelvin
- 1˚C initial accuracy available
- Operates from 400 µA to 5 mA
- Less than 1Ω dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200˚C overrange
- Low cost

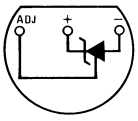
Schematic Diagram



LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

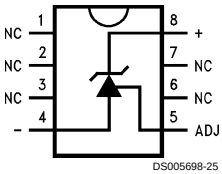
Connection Diagrams

TO-92
Plastic Package



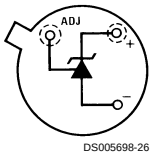
Bottom View
Order Number LM335Z
or LM335AZ
See NS Package
Number Z03A

SO-8
Surface Mount Package



Order Number LM335M
See NS Package
Number M08A

TO-46
Metal Can Package*



*Case is connected to negative pin

Bottom View
Order Number LM135H,
LM135H-MIL, LM235H,
LM335H, LM135AH,
LM235AH or LM335AH
See NS Package
Number H03H

Absolute Maximum Ratings (Note 4)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Reverse Current	15 mA
Forward Current	10 mA
Storage Temperature	
TO-46 Package	−60˚C to +180˚C
TO-92 Package	−60˚C to +150˚C
SO-8 Package	−65˚C to +150˚C

Specified Operating Temp. Range

	Continuous	Intermittent (Note 2)
LM135, LM135A	−55˚C to +150˚C	150˚C to 200˚C
LM235, LM235A	−40˚C to +125˚C	125˚C to 150˚C
LM335, LM335A	−40˚C to +100˚C	100˚C to 125˚C

Lead Temp. (Soldering, 10 seconds)	
TO-92 Package:	260˚C
TO-46 Package:	300˚C
SO-8 Package:	300˚C
Vapor Phase (60 seconds):	215˚C
Infrared (15 seconds):	220˚C

Temperature Accuracy (Note 1)

LM135/LM235, LM135A/LM235A

Parameter	Conditions	LM135A/LM235A			LM135/LM235			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25˚C, I_R = 1 mA	2.97	2.98	2.99	2.95	2.98	3.01	V
Uncalibrated Temperature Error	T_C = 25˚C, I_R = 1 mA		0.5	1		1	3	˚C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		1.3	2.7		2	5	˚C
Temperature Error with 25˚C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.3	1		0.5	1.5	˚C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		˚C
Non-Linearity	I_R = 1 mA		0.3	0.5		0.3	1	˚C

Temperature Accuracy (Note 1)

LM335, LM335A

Parameter	Conditions	LM335A			LM335			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25˚C, I_R = 1 mA	2.95	2.98	3.01	2.92	2.98	3.04	V
Uncalibrated Temperature Error	T_C = 25˚C, I_R = 1 mA		1	3		2	6	˚C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		2	5		4	9	˚C
Temperature Error with 25˚C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.5	1		1	2	˚C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		˚C
Non-Linearity	I_R = 1 mA		0.3	1.5		0.3	1.5	˚C

Electrical Characteristics (Note 1)

Parameter	Conditions	LM135/LM235 LM135A/LM235A			LM335 LM335A			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage Change with Current	400 µA≤I_R≤5 mA At Constant Temperature		2.5	10		3	14	mV
Dynamic Impedance	I_R=1 mA		0.5			0.6		Ω
Output Voltage Temperature Coefficient			+10			+10		mV/˚C
Time Constant	Still Air		80			80		sec
	100 ft/Min Air		10			10		sec
	Stirred Oil		1			1		sec
Time Stability	T_C=125˚C		0.2			0.2		˚C/khr

Electrical Characteristics (Note 1) (Continued)

Note 1: Accuracy measurements are made in a well-stirred oil bath. For other conditions, self heating must be considered.

Note 2: Continuous operation at these temperatures for 10,000 hours for H package and 5,000 hours for Z package may decrease life expectancy of the device.

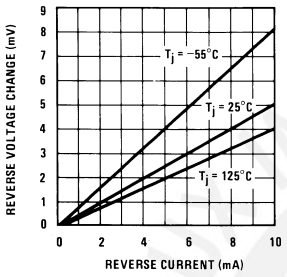
Note 3:

Thermal Resistance	TO-92	TO-46	SO-8
θ_{JA} (junction to ambient)	202˚C/W	400˚C/W	165˚C/W
θ_{JC} (junction to case)	170˚C/W	N/A	N/A

Note 4: Refer to RETS135H for military specifications.

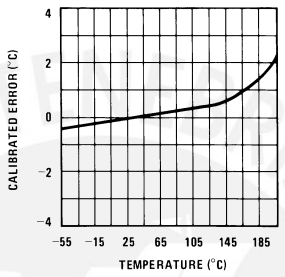
Typical Performance Characteristics

Reverse Voltage Change



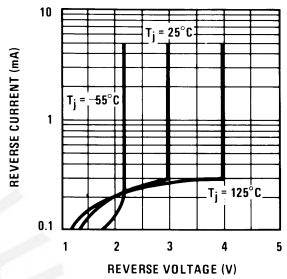
DS005698-27

Calibrated Error



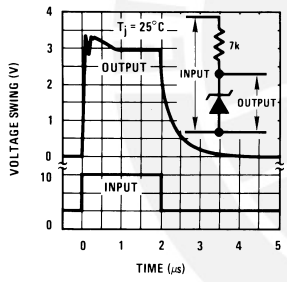
DS005698-28

Reverse Characteristics



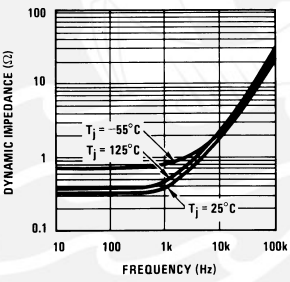
DS005698-29

Response Time



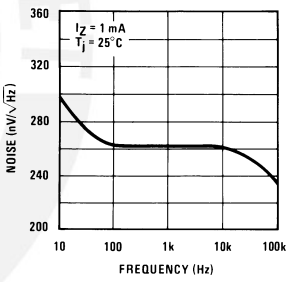
DS005698-30

Dynamic Impedance



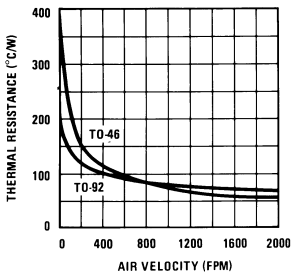
DS005698-31

Noise Voltage



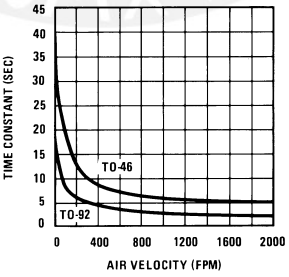
DS005698-32

Thermal Resistance Junction to Air



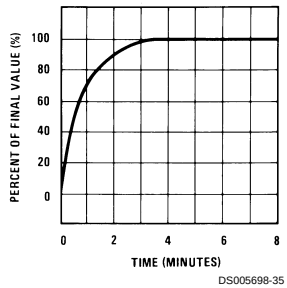
DS005698-33

Thermal Time Constant



DS005698-34

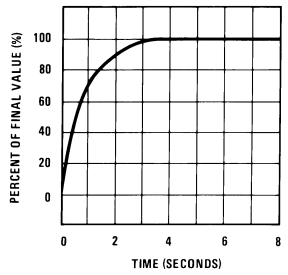
Thermal Response in Still Air



DS005698-35

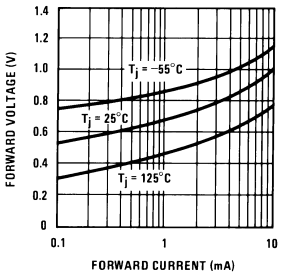
Typical Performance Characteristics (Continued)

Thermal Response in Stirred Oil Bath



DS005698-36

Forward Characteristics



DS005698-37

Application Hints

CALIBRATING THE LM135

Included on the LM135 chip is an easy method of calibrating the device for higher accuracies. A pot connected across the LM135 with the arm tied to the adjustment terminal allows a 1-point calibration of the sensor that corrects for inaccuracy over the full temperature range.

This single point calibration works because the output of the LM135 is proportional to absolute temperature with the extrapolated output of sensor going to 0V output at 0°K (−273.15°C). Errors in output voltage versus temperature are only slope (or scale factor) errors so a slope calibration at one temperature corrects at all temperatures.

The output of the device (calibrated or uncalibrated) can be expressed as:

$$V_{OUT_T} = V_{OUT_{T_o}} \times \frac{T}{T_o}$$

where T is the unknown temperature and T_o is a reference temperature, both expressed in degrees Kelvin. By calibrating the output to read correctly at one temperature the output at all temperatures is correct. Nominally the output is calibrated at 10 mV/°K.

To insure good sensing accuracy several precautions must be taken. Like any temperature sensing device, self heating can reduce accuracy. The LM135 should be operated at the lowest current suitable for the application. Sufficient current, of course, must be available to drive both the sensor and the calibration pot at the maximum operating temperature as well as any external loads.

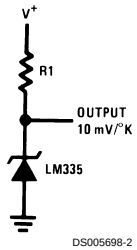
If the sensor is used in an ambient where the thermal resistance is constant, self heating errors can be calibrated out. This is possible if the device is run with a temperature stable current. Heating will then be proportional to zener voltage and therefore temperature. This makes the self heating error proportional to absolute temperature the same as scale factor errors.

WATERPROOFING SENSORS

Meltable inner core heat shrinkable tubing such as manufactured by Raychem can be used to make low-cost waterproof sensors. The LM335 is inserted into the tubing about ½" from the end and the tubing heated above the melting point of the core. The unfilled ½" end melts and provides a seal over the device.

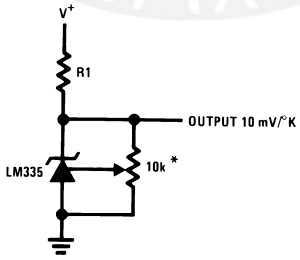
Typical Applications

Basic Temperature Sensor



DS005698-2

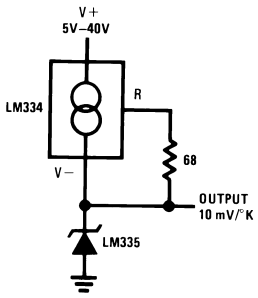
Calibrated Sensor



DS005698-9

*Calibrate for 2.982V at 25°C

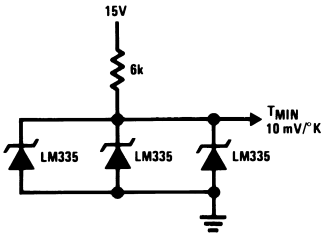
Wide Operating Supply



DS005698-10

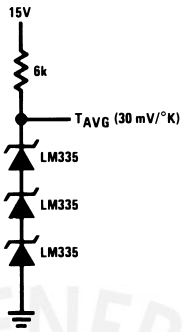
Typical Applications (Continued)

Minimum Temperature Sensing



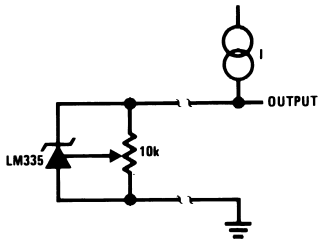
DS005698-4

Average Temperature Sensing



DS005698-18

Remote Temperature Sensing



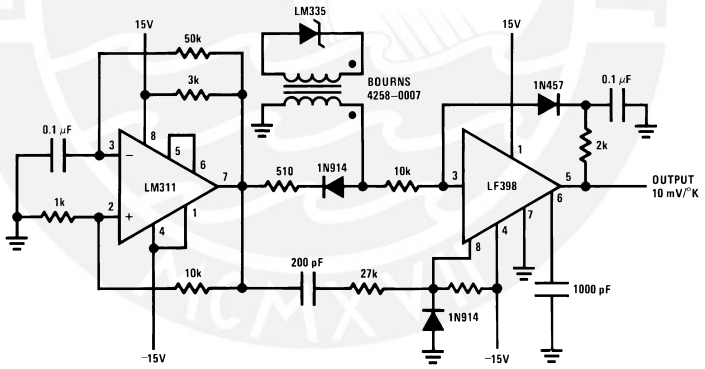
DS005698-19

Wire length for 1˚C error due to wire drop

	I_R = 1 mA	I_R = 0.5 mA*
AWG	FEET	FEET
14	4000	8000
16	2500	5000
18	1600	3200
20	1000	2000
22	625	1250
24	400	800

*For I_R = 0.5 mA, the trim pot must be deleted.

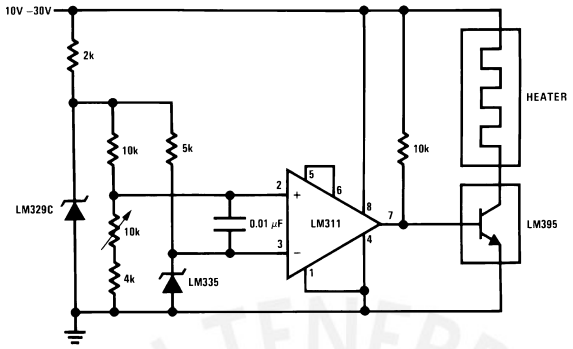
Isolated Temperature Sensor



DS005698-20

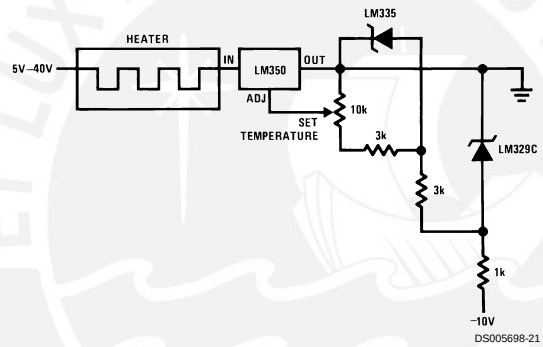
Typical Applications (Continued)

Simple Temperature Controller



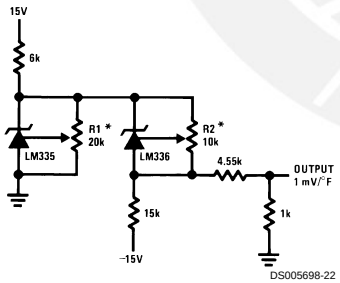
DS005698-5

Simple Temperature Control



DS005698-21

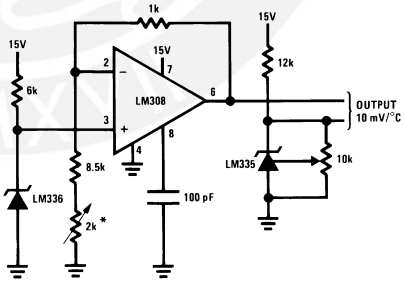
Ground Referred Fahrenheit Thermometer



DS005698-22

*Adjust R2 for 2.554V across LM336.
Adjust R1 for correct output.

Centigrade Thermometer

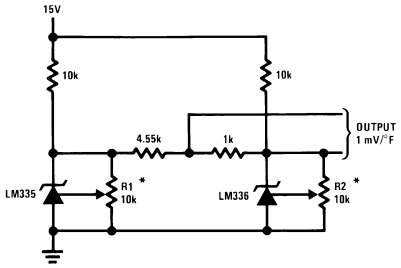


DS005698-23

*Adjust for 2.7315V at output of LM308

Typical Applications (Continued)

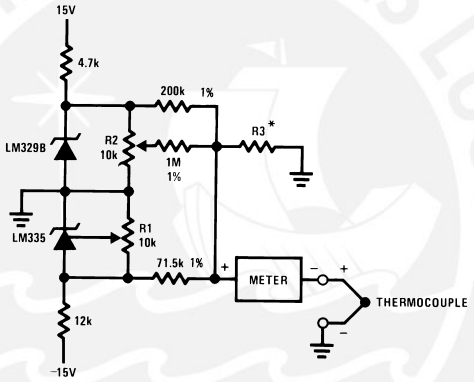
Fahrenheit Thermometer



DS005698-24

*To calibrate adjust R2 for 2.554V across LM336.
Adjust R1 for correct output.

THERMOCOUPLE COLD JUNCTION COMPENSATION
Compensation for Grounded Thermocouple



DS005698-6

*Select R3 for proper thermocouple type

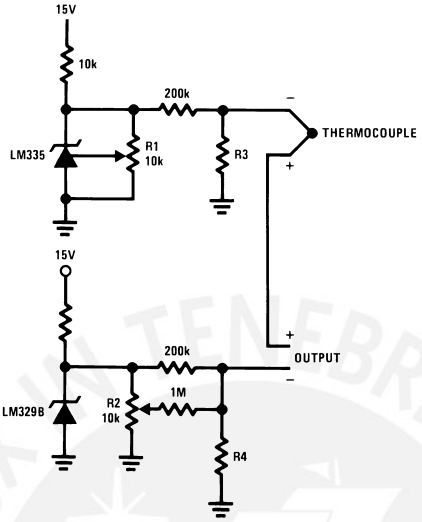
THERMO-COUPLE	R3 (±1%)	SEEBECK COEFFICIENT
J	377Ω	52.3 µV/°C
T	308Ω	42.8 µV/°C
K	293Ω	40.8 µV/°C
S	45.8Ω	6.4 µV/°C

Adjustments: Compensates for both sensor and resistor tolerances
1. Short LM329B
2. Adjust R1 for Seebeck Coefficient times ambient temperature (in degrees K) across R3.
3. Short LM335 and adjust R2 for voltage across R3 corresponding to thermocouple type

J	14.32 mV	K	11.17 mV
T	11.79 mV	S	1.768 mV

Typical Applications (Continued)

Single Power Supply Cold Junction Compensation



DS005698-11

*Select R3 and R4 for thermocouple type

THERMO-COUPLE	R3	R4	SEEBECK COEFFICIENT
J	1.05K	385Ω	52.3 µV/˚C
T	856Ω	315Ω	42.8 µV/˚C
K	816Ω	300Ω	40.8 µV/˚C
S	128Ω	46.3Ω	6.4 µV/˚C

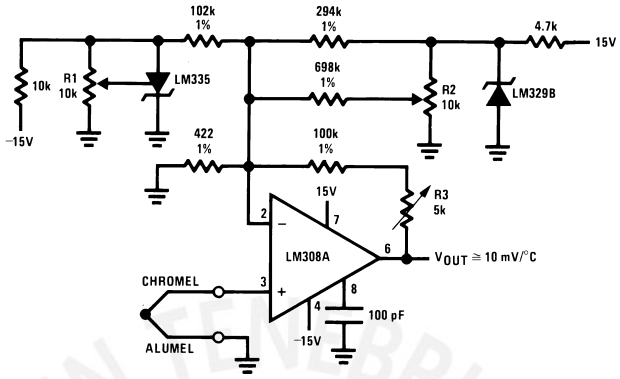
Adjustments:

1. Adjust R1 for the voltage across R3 equal to the Seebeck Coefficient times ambient temperature in degrees Kelvin.
2. Adjust R2 for voltage across R4 corresponding to thermocouple

J	14.32 mV
T	11.79 mV
K	11.17 mV
S	1.768 mV

Typical Applications (Continued)

Centigrade Calibrated Thermocouple Thermometer



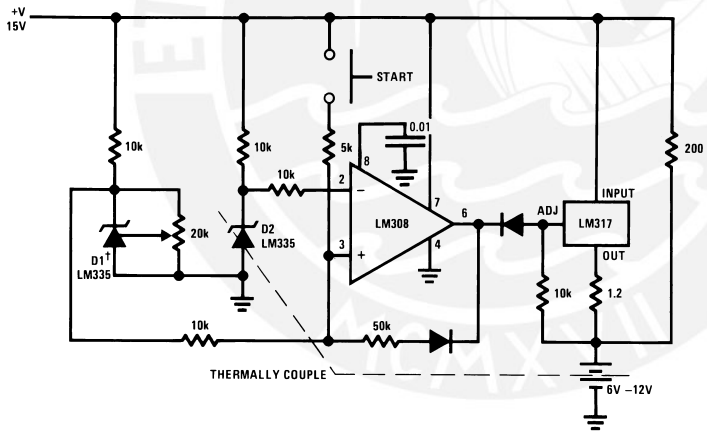
DS005698-12

Terminate thermocouple reference junction in close proximity to LM335.

Adjustments:

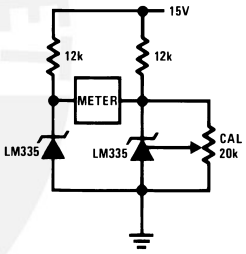
1. Apply signal in place of thermocouple and adjust R3 for a gain of 245.7.
2. Short non-inverting input of LM308A and output of LM329B to ground.
3. Adjust R1 so that V_{OUT} = 2.982V @ 25°C.
4. Remove short across LM329B and adjust R2 so that V_{OUT} = 246 mV @ 25°C.
5. Remove short across thermocouple.

Fast Charger for Nickel-Cadmium Batteries



DS005698-13

Differential Temperature Sensor

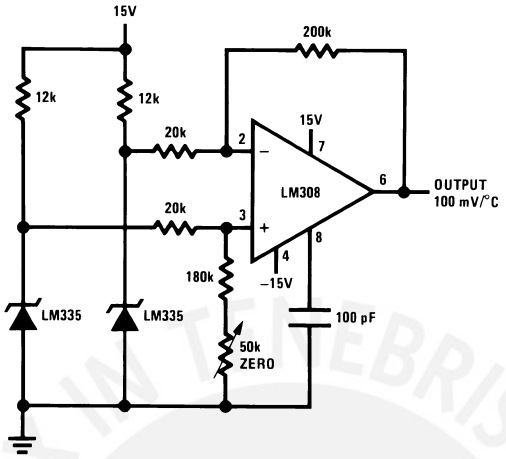


DS005698-7

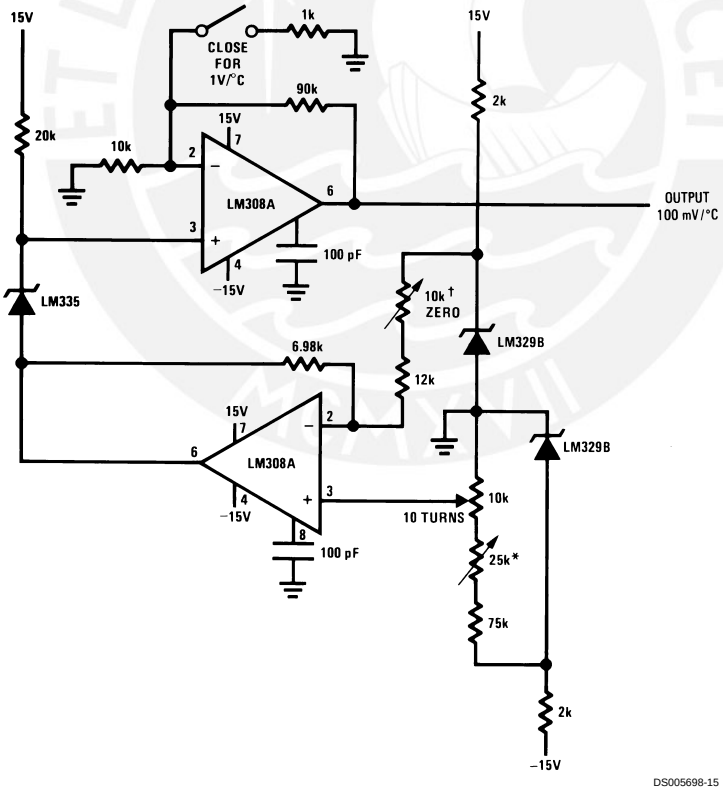
†Adjust D1 to 50 mV greater V_Z than D2.
Charge terminates on 5°C temperature rise. Couple D2 to battery.

Typical Applications (Continued)

Differential Temperature Sensor



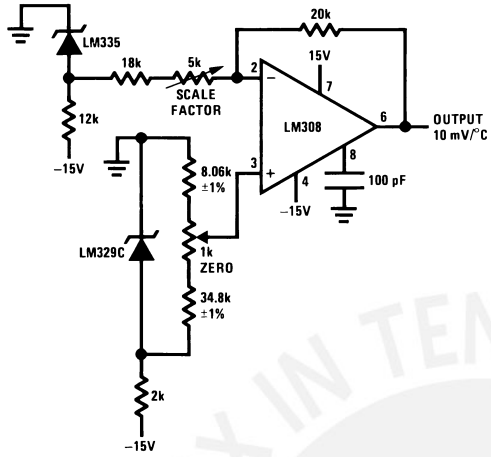
Variable Offset Thermometer‡



†Adjust for zero with sensor at 0°C and 10T pot set at 0°C
*Adjust for zero output with 10T pot set at 100°C and sensor at 100°C
‡Output reads difference between temperature and dial setting of 10T pot

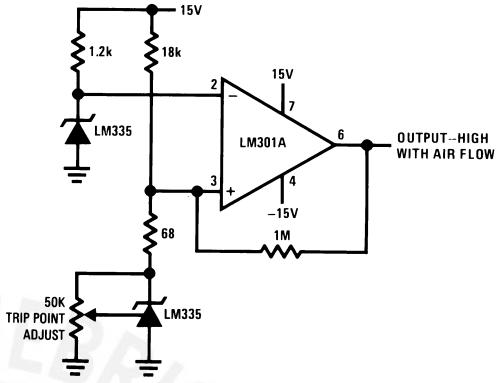
Typical Applications (Continued)

Ground Referred Centigrade Thermometer



DS005698-16

Air Flow Detector*



DS005698-17

*Self heating is used to detect air flow

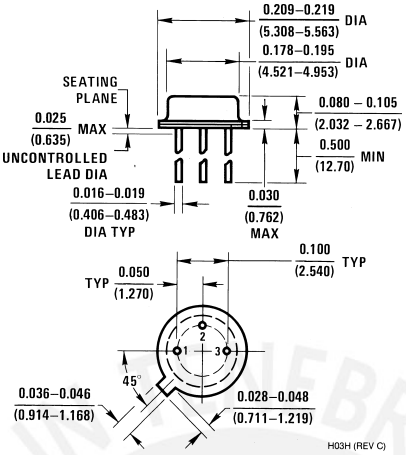
Definition of Terms

Operating Output Voltage: The voltage appearing across the positive and negative terminals of the device at specified conditions of operating temperature and current.

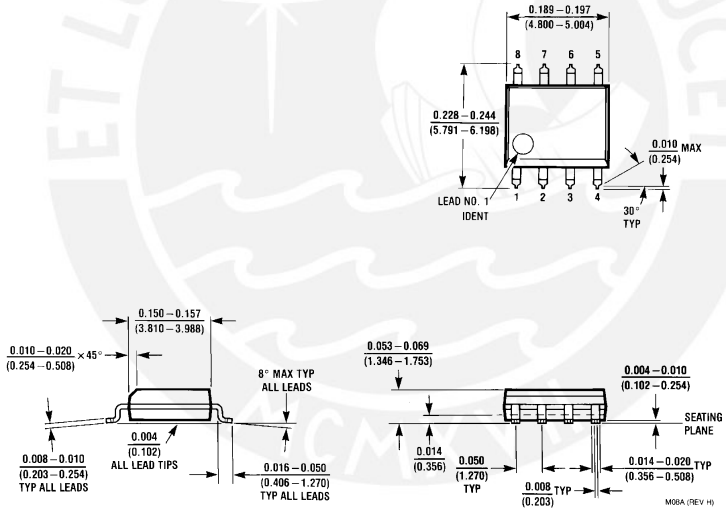
Uncalibrated Temperature Error: The error between the operating output voltage at 10 mV/°K and case temperature at specified conditions of current and case temperature.

Calibrated Temperature Error: The error between operating output voltage and case temperature at 10 mV/°K over a temperature range at a specified operating current with the 25°C error adjusted to zero.

Physical Dimensions inches (millimeters) unless otherwise noted

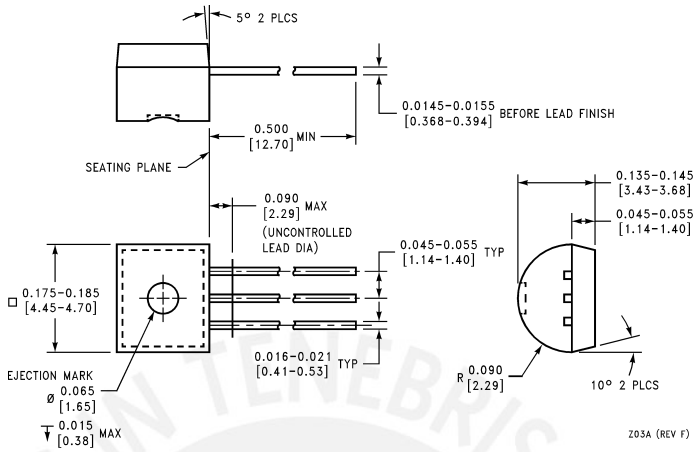


Metal Can Package (H)
Order Number LM135H, LM235H, LM335H, LM135AH, LM235AH or LM335AH
NS Package Number H03H



8-Lead Molded Small Outline Package (M)
Order Number LM335M
NS Package Number M08A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



Plastic Package
Order Number LM335Z or LM335AZ
NS Package Z03A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 1 80-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 1 80-530 85 85
English Tel: +49 (0) 1 80-532 78 32
Français Tel: +49 (0) 1 80-532 93 58
Italiano Tel: +49 (0) 1 80-534 16 80

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

User's Manual

EPIA-M
Mini-ITX Mainboard

P/N: 99-51-012561-14
Version 1.40
December 11, 2003

Copyright

Trademarks

Data protection

All data should be backed-up prior to the installation of any drive unit or storage peripheral. VIA will not be responsible for any loss of data resulting from the use, disuse or misuse of this or any other VIA product.

No Warranty

VIA has made every effort to ensure the accuracy of the content of this manual. However, it is possible that it may contain technical inaccuracies or typographical or other errors. Our products are under continual improvement and we reserve the right to make changes without notice. VIA will assume no liability for any inaccuracy found in this publication, nor for damages, direct, indirect, incidental, consequential or otherwise, that may result from such an inaccuracy, including without limitation loss of data or profits.

VIA provides this manual “as is”, and does not issue a warranty of any kind, express or implied, including without limitation implied warranties of merchantability or fitness for a particular purpose.

The information provided in this manual is subject to change without notice. VIA reserves the right to alter product designs, layouts or drivers without notification.

FCC-B Radio Frequency Interference Statement

This equipment has been tested and found to comply with the limits for a class B digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Notice 1

The changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Notice 2

Shielded interface cables and A.C. power cord, if any, must be used in order to comply with the emission limits.

Tested To Comply
With FCC Standards
FOR HOME OR OFFICE USE

Safety Instructions

1. Always read the safety instructions carefully.
2. Keep this User's Manual for future reference.
3. Keep this equipment away from humidity.
4. Lay this equipment on a reliable flat surface before setting it up.
5. The openings on the enclosure are for air convection hence protects the equipment from overheating. DO NOT COVER THE OPENINGS.
6. Make sure the voltage of the power source and adjust properly 110/220V before connecting the equipment to the power inlet.
7. Place the power cord in such a way that people cannot step on it. Do not place anything over the power cord.
8. Always unplug the power cord before inserting any add-on card or module.
9. All cautions and warnings on the equipment should be noted.
10. Never pour any liquid into the opening. Liquid can cause damage or electrical shock.
11. If any of the following situations arises, get the equipment checked by a service personnel:
    - The power cord or plug is damaged
    - Liquid has penetrated into the equipment
    - The equipment has been exposed to moisture
    - The equipment has not work well or you can not get it work according to User's Manual.
    - The equipment has dropped and damaged
    - If the equipment has obvious sign of breakage
12. DO NOT LEAVE THIS EQUIPMENT IN AN ENVIRONMENT UNCONDITIONED, STORAGE TEMPERATURE ABOVE 60°C (140°F), IT MAY DAMAGE THE EQUIPMENT.

CAUTION: Explosion or serious damage may occur if the battery is incorrectly replaced. Replace only with the same or equivalent type recommended by the manufacturer.

Box Contents

- 1 x VIA Mainboard
- 1 x User's Manual
- 1 x Floppy Ribbon Cable
- 1 x ATA-33/66/100 IDE Ribbon Cable
- 1 x Combo Module (2 port USB 2.0 and 2 port IEEE1394)
- 1 x Driver Utilities CD

Table of Contents

Chapter 1

Specifications

The ultra-compact and highly intergrated VIA EPIA-M Mini-ITX Mainboard is the smallest form factor mainboard specification available today, developed by VIA Technologies, Inc. as part of the company's open industry-wide total connectivity initiative. The mainboard enables the creation of an exciting new generation of small, ergonomic, innovative and affordable embedded systems. Through high level of integration, mini-ITX only occupy 66% of the size of FlexATX mainboard form factor. The mainboard comes with an embedded VIA Processor, boasting ultra low power consumption and cool, quiet operation.

This chapter includes the following sections:

Mainboard Specifications

CPU

- VIA C3 / EDEN EBGA Processor (on board)
- Enhanced Ball Grid Array Package (EBGA)
- Internal L1 128KB and L2 64KB cache memory

Chipset

- VIA CLE266 North Bridge
- VT8235 South Bridge

Graphics

- Integrated UniChrome graphics with MPEG-2 accelerator

Audio

- VT1616 six channel AC'97 Codec
- 3 Audio jacks: Line-in, Line-out and Mic-in; switched to 6-channel output during 6-channel operations with Smart 5.1 (See Appendix A)

Main Memory

- 1 DDR266 DIMM socket
- Up to 1GB memory size

PCI Bus & IDE

- 1 PCI slot
- 2 X UltraDMA 66 / 100 / 133 Connector

LAN

- VIA VT6103 10 / 100 Base-T Ethernet PHY

USB

- USB v2.0 / v1.1

Firewire

- IEEE 1394; VIA VT6307S 2-port Firewire

TV-Out (optional)

- VIA VT1622 TV-Out Controller
- Supports 640 x 480, 800 x 600, and 1024 x 768 NTSC/PAL TV

Onboard I/O Connectors

- Two 1394 connectors for two 1394 ports
- Front-panel audio connectors (Mic and Line Out)
- CD Audio-in connector
- 1 FIR connector; 1 PS2 connector
- Wake-on-LAN
- CPU / System Fan / FAN3
- 1 I^2C connector
- 1 Connector for LVDS module (Optional)
- Serial port connector for second COM port

Back Panel I/O Ports

- 1 PS2 mouse port; 1 PS2 keyboard port
- 1 Parallel port; 1 RJ-45 LAN port; 1 Serial port
- 2 USB 2.0 ports; 1 VGA port
- 1 RCA port (SPDIF or TV out); 1 S-Video port
- 3 Audio jacks: line-out, line-in and mic-in; can be switched to 6 channel output with Smart 5.1 (See Appendix A)

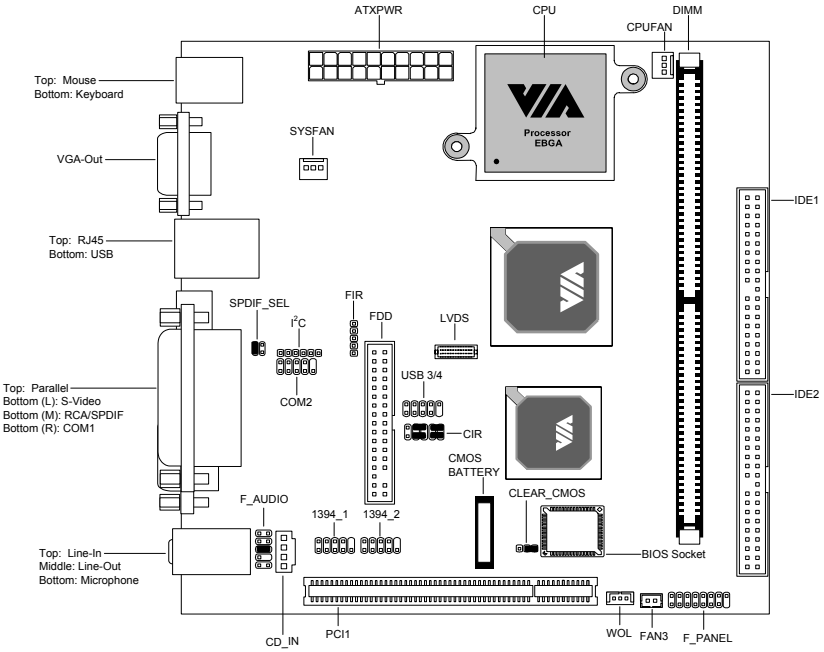
BIOS

- AwardBIOS with 2 / 4Mbit flash memory

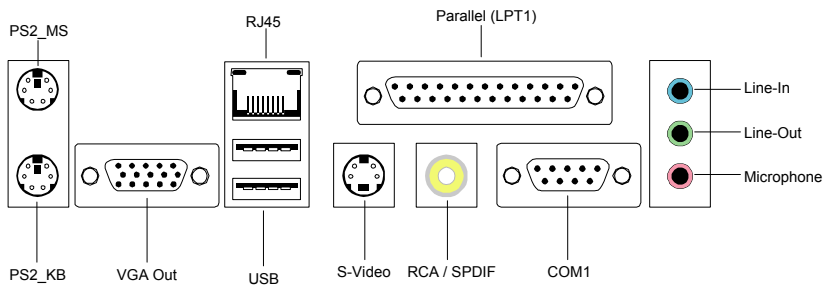
Form Factor

- 17 cm X 17 cm Mini-ITX (4 layers)

Mainboard Layout



Back Panel



Back Panel Ports

Port	Description	Page
Audio Jacks	Line-Out, Line-In, Microphone	2-10
COM 1	Serial port	2-10
LPT1	Parallel port	2-9
PS2-MS	PS2 mouse port	2-7
PS2-KB	PS2 keyboard port	2-7
RCA_JACK	RCA Video or SPDIF jack	2-8
RJ45	10/100 NIC port	2-8
S-VIDEO	S-Video Port	2-8
USB 1-2	Universal Serial Bus ports 1 - 2	2-8
VGA Out	VGA out port	2-8

Slots

Slot	Description	Page
DIMM	Memory module slot	2-4
PCI	Expansion card slot	2-20

Onboard Connectors and Jumpers

Connecter/Jumper	Description	Page
1394_1	Connector for first 1394 port	2-15
1394_2	Connector for second 1394 port	2-15
ATXPWR	ATX power cable connector	2-6
CD_IN	Onboard CD audio cable connector	2-16
CIR	Consumer IR connector	2-13
CLEAR_CMOS	Jumper to reset CMOS settings to default	2-19
COM2	Second serial port connector	2-15
F_AUDIO	Connectors for optional front audio panel	2-17
F_PANEL	Case connectors	2-12
Fans	CPU, System, Fan3	2-2
FDD	Floppy disk drive connector	2-16
FIR	Fast Infrared Radiation connector	2-13
I^2C	I^2C connector	2-17
IDE 1-2	IDE hard disk drive connectors	2-11
LVDS	LVDS connector	2-18
SPDIF_SEL	Sony Philips Digital Interface jumper	2-19
USB 3/4	Universal Serial Bus connectors 3 - 4	2-14
WOL	Wake On LAN connector	2-14

Chapter

2

Installation

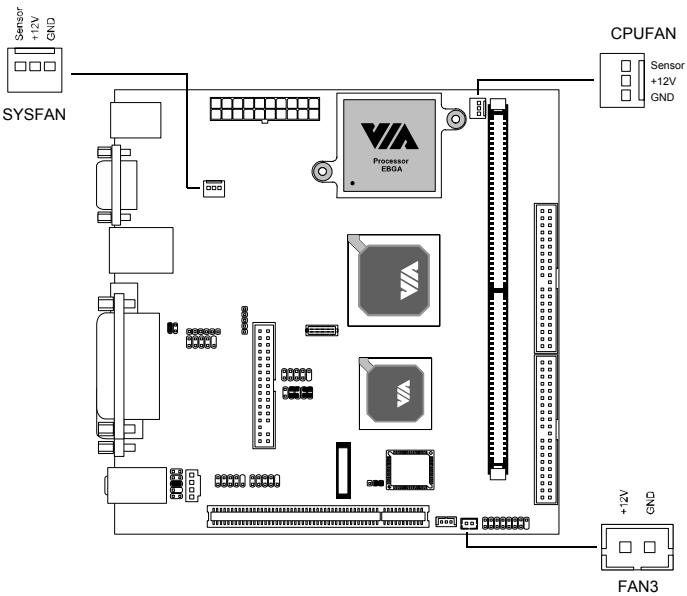
This chapter provides you with information about hardware setup procedures. While installing the mainboard, carefully hold the components and closely follow the installation procedures. Some components may be damaged if they are installed incorrectly.

It is recommended to use a grounded wrist strap before handling computer components. Static electricity can damage some components.

This chapter includes the following sections:

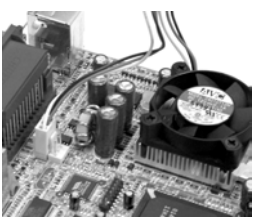
CPU

The VIA EPIA-M Mini-ITX Mainboard includes an embedded VIA Eden Processor or VIA C3™ E-Series Processor. The CPUFAN (CPU fan) and SYSFAN (system fan) run on +12V and maintain system cooling. When connecting the wire to the connectors, always be aware that the red wire is the Positive and should be connected to the +12V. The black wire is Ground and should be connected to GND. Both CPU and System fan connectors have sensors to detect fan speed, but the power fan does not have a sensor. FAN3 is an additional FAN connector.



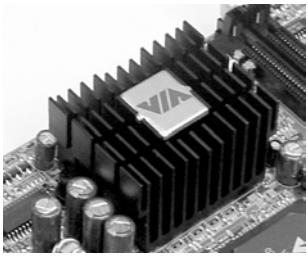
The VIA C3™ E-Series Processor

With low power consumption and advanced thermal dissipation properties, the embedded VIA C3™ E-Series requires only a small fan to guarantee performance and reliability. Ensure that the CPU Fan Connector is correctly installed as shown.



The VIA Eden Processor

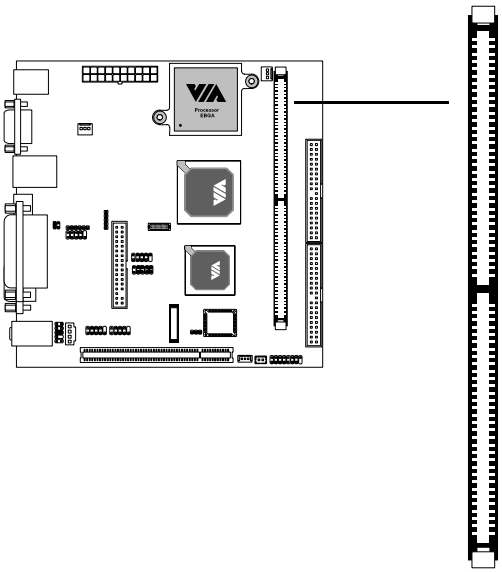
Providing ultra-low power consumption and advanced thermal dissipation properties, the VIA Eden Processor features a fanless design. The VIA Eden Processor requires only a heatsink as shown.



Warning: This motherboard is not designed to support overclocking. Any attempt to operate beyond product specifications is not recommended. We do not guarantee the damages or risks caused by operation beyond product specifications.

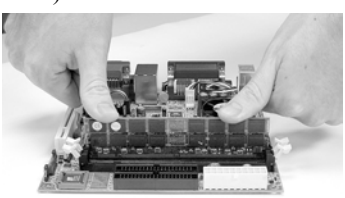
Memory Module Installation

The VIA EPIA-M Mini-ITX Mainboard provides one 184-pin DIMM slot for DDR266 SDRAM memory modules.



DDR SDRAM Module Installation Procedures

1. Push the white retaining latches at either end of the DIMM slot outwards.
2. Align the DDR SDRAM module with the corresponding notches on the DIMM slot. The modules will only fit if placed in the correct position.
3. With both hands, press the DDR SDRAM module down into the DIMM slot so that the white retaining latches rotate up and secure the module in place (see picture below).



Available DDR SDRAM Configurations

Refer to the table below for available DDR SDRAM configurations on the mainboard.

Slot	Memory Module	Total Memory
DIMM (Bank 0 & 1)	64MB, 128MB, 256MB, 512MB, 1GB	64 MB - 1 GB
Maximum System Memory Supported		64 MB - 1 GB

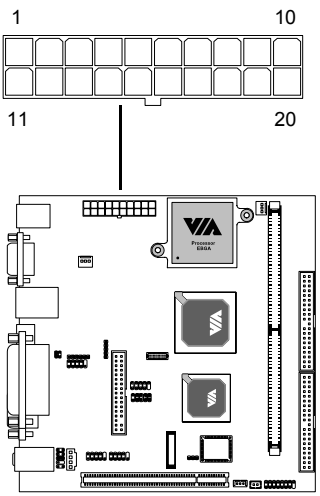
Connecting the Power Supply

The VIA EPIA-M Mini-ITX Mainboard requires an ATX power supply to be connected. Before inserting the power supply connector, always make sure that all components are installed correctly to ensure that no damage will be caused.

ATX 20-Pin Power Connector

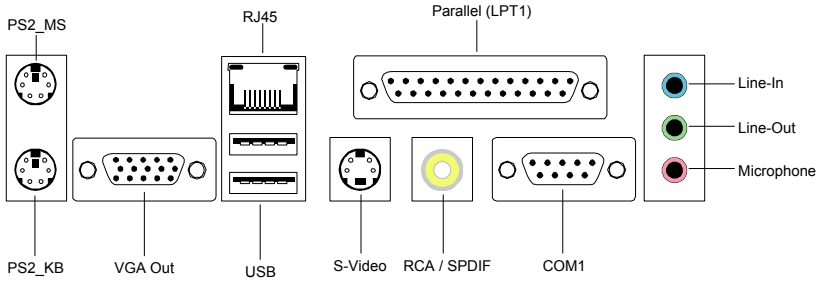
To connect the ATX power supply, make sure the plugs of the power supply are inserted in the proper orientation and the pins are correctly aligned. Then, push down the power supply plug firmly into the connector.

Pin	Signal
1	3.3V
2	3.3V
3	GND
4	5V
5	GND
6	5V
7	GND
8	PW_OK
9	5V_SB
10	12V
11	3.3V
12	-12V
13	GND
14	PS_ON
15	GND
16	GND
17	GND
18	NC
19	5V
20	5V



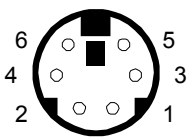
Back Panel Ports

The back panel has the following ports:



Mouse Port: PS2_MS

The mainboard provides a standard PS/2 mouse connector for attaching a PS/2 mouse. You can plug a PS/2 mouse directly into this connector. The connector location and pin assignments are as follows.

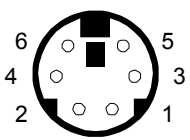


PS2 Mouse (6-pin female)

Pin	Signal	Description
1	Mouse DATA	Mouse data
2	NC	No connection
3	GND	Ground
4	VCC	+5V
5	Mouse Clock	Mouse clock
6	NC	No connection

Keyboard Port: PS2_KB

The mainboard provides a standard PS/2 keyboard connector for attaching a PS/2 keyboard. You can plug a PS/2 keyboard directly into this connector.

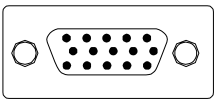


PS2 Keyboard (6-pin female)

Pin	Signal	Description
1	Keyboard DATA	Keyboard data
2	NC	No connection
3	GND	Ground
4	VCC	+5V
5	Keyboard Clock	Keyboard clock
6	NC	No connection

VGA Out

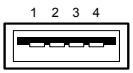
A DB-15 pin female connector that connects to a VGA monitor.



USB Ports

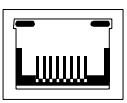
The mainboard provides 2 USB 2.0 ports. USB-compatible devices can be plugged directly into these ports.

Pin	Signal	Description
1	VCC	+ 5V
2	- DATA	Negative data channel
3	+ DATA	Positive data channel
4	GND	Ground



RJ45 10/100 NIC Port

The mainboard provides one standard RJ-45 port for connection to the Local Area Network (LAN). You can connect a network cable to the LAN port.

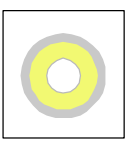


S-Video Port

This port allows S-Video output in NTSC and PAL modes.

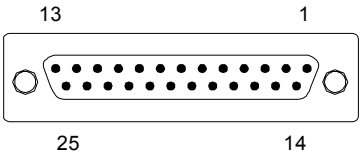
RCA Video or S/PDIF Port

This dual function port may be used either as a RCA Video port or as a S/PDIF port. See SPDIF_SEL in the Jumpers section for more details.



Parallel Port: LPT1

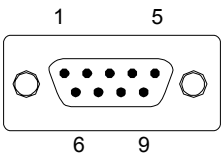
The mainboard provides a 25-pin female connector for LPT (parallel port). A parallel port is a standard printer port that supports Enhanced Parallel Port (EPP) and Extended Capabilities Parallel Port (ECP) modes.



Pin	Signal	Description
1	STROBE	Strobe
2	DATA0	Data 0
3	DATA1	Data 1
4	DATA2	Data 2
5	DATA3	Data 3
6	DATA4	Data 4
7	DATA5	Data 5
8	DATA6	Data 6
9	DATA7	Data 7
10	ACK#	Acknowledge
11	BUSY	Busy
12	PE	Paper End
13	SELECT	Select
14	AUTOFEED#	Automatic Feed
15	ERR#	Error
16	INIT#	Initialize Printer
17	SLIN#	Select In
18	GND	Ground
19	GND	Ground
20	GND	Ground
21	GND	Ground
22	GND	Ground
23	GND	Ground
24	GND	Ground
25	GND	Ground

Serial Ports: COM1

The mainboard offers two 9-pin male Serial Port connectors COM1. You can attach a serial mouse or other serial devices directly to these ports.



9-Pin Serial Port

Pin	Signal	Description
1	DCD	Data Carry Detect
2	SIN	Serial In or Receive Data
3	SOUT	Serial Out or Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicate

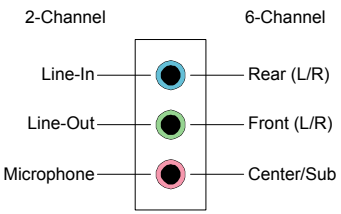
Audio Jacks: Line-In, Line-Out, Microphone

Jack	2-Channel	6-Channel
Line-In	Line in	Rear (Left / Right)
Line-Out	Line out	Front (Left / Right)
Mic	Microphone	Center / Subwoofer

The **Line-Out** jack is for connecting to external speakers or headphones.

The **Line-In** jack is for connecting to an external audio device such as a CD player, tape player, etc....

The **Mic** jack is for connecting to a microphone.

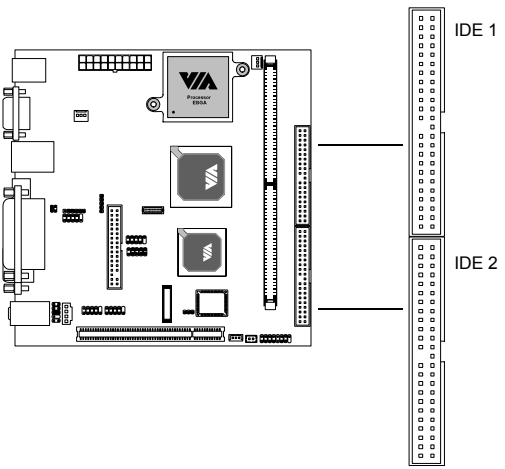


When 6-channel applications are used, all three jacks become output connectors with Smart 5.1 (See Appendix A) In order for the 6-channel audio to function, the operating system and multimedia application must be properly configured. Please note that Windows 98 only supports 4-channel audio.

Connectors

Hard Disk Connectors: IDE1 & IDE2

The mainboard has a 32-bit Enhanced PCI IDE and Ultra DMA 33/66/100/133 controller that provides PIO mode 0~4, Bus Master, and Ultra DMA 33/66/100/133 functions. You can connect up to four hard disk drive, CD-ROM, LS-120 and other devices. These connectors utilize the provided IDE hard disk cable.



IDE1 (Primary IDE Connector)

The first hard drive should always be connected to IDE1. IDE1 can connect a Master and a Slave drive. You must configure the second hard drive to Slave mode by setting the jumper accordingly.

IDE2 (Secondary IDE Connector)

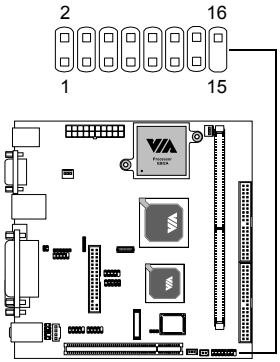
IDE2 can also connect a Master and a Slave drive.

If you install two hard disks on a single cable, you must set the jumper on the second hard disk drive to slave mode. Please refer to the hard disk documentation supplied by hard disk vendor for the jumper settings.

Case Connectors: F_PANEL

The F_PANEL connector block allows you to connect to the power switch, reset switch, power LED, HDD LED, SLED and the Speaker on the case.

Pin	Signal	Pin	Signal
1	PWR LED+	2	HDD LED+
3	PWR LED+	4	HDD LED-
5	PWR LED-	6	PW_BN+
7	SPEAKER+	8	PW_BN-
9	NC	10	RESET+
11	NC	12	RESET-
13	SPEAKER-	14	SLED+
15	NC	16	SLED-



Power Switch (PW_BN)
Connect to a 2-pin push button switch. Pressing this button will turn the system power on or off.

Reset Switch (RESET)
The Reset Switch is used to reboot the system rather than turning the power ON/OFF. Avoid rebooting while the HDD is working. You can connect the Reset Switch from the system case to this pin.

Power LED (PWR LED)
The LED is lit when the system is power on. If the system is in S1 (POS - Power On Suspend) or S3 (STR - Suspend To RAM) state, the LED will blink.

HDD LED
HDD LED shows the activity of a hard disk drive. Avoid turning the power off while HDD LED is lit. Connect the HDD LED from the system case to this pin.

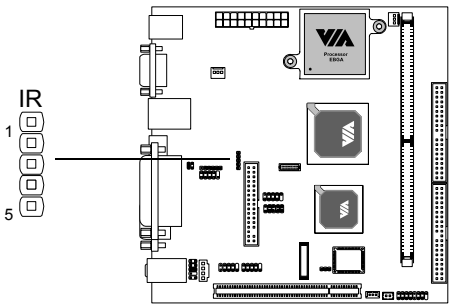
SLED
The SLED is lit when the system is in the S1 (POS - Power On Suspend) state.

Speaker
The speaker from the system case is connected to this pin

Fast IrDA Infrared Module Connector: IR

This connector allows you to connect an IrDA Infrared module. You must configure the setting through the BIOS setup to activate the IR function.

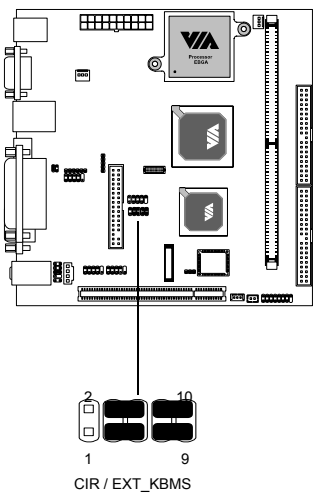
Pin	Signal
1	VCC
2	IRRX1
3	IRRX
4	GND
5	IRTX



Consumer Infrared Module, PS2 Header: CIR / EXT_KBMS

When the header is not in use, please short pin 3&5, pin 4&6, pin 7&9, and pin 8&10.

Pin	Signal	Pin	Signal
1	+5V	2	GND
3	KB_CLK	4	KB_DATA
5	EXT_KBCLK	6	EXT_KBDATA
7	MS_CLK	8	MS_DATA
9	EXT_MSCLK	10	EXT_MSDATA

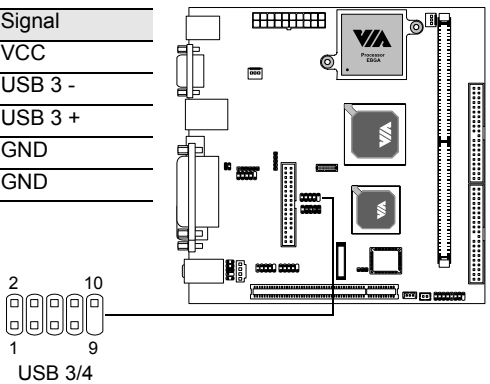


CIR / EXT_KBMS

USB pin-header: USB3/4

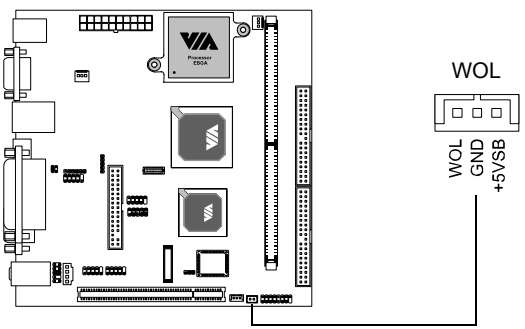
The mainboard provides 1 front USB pin-header connector, allowing up to 2 additional USB ports. Please plug the USB 2-port module onto this pin-header.

Pin	Signal	Pin	Signal
1	VCC	2	VCC
3	USB2 -	4	USB 3 -
5	USB2 +	6	USB 3 +
7	GND	8	GND
9	NC	10	GND



Wake-on LAN: WOL

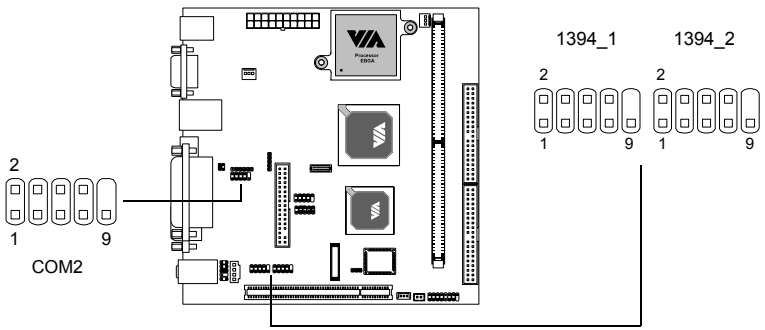
This connector allows you to connect a network card with the Wake-On LAN function. The connector will power up the system when a signal is received through the network card. Please note that the function of ACPI WOL may be disabled when users unplug the power cord or turn off the power button manually.



FireWire: IEEE1394

FireWire is a serial I/O interface that provides you fast data transfer rates. There are 2 FireWire ports available.

Pin	Signal	Pin	Signal
1	TPA0+	2	TPA0-
3	GND	4	GNF
5	TPB0+	6	TPB0-
7	1394_VDD	8	1394_VDD
9	GND	10	



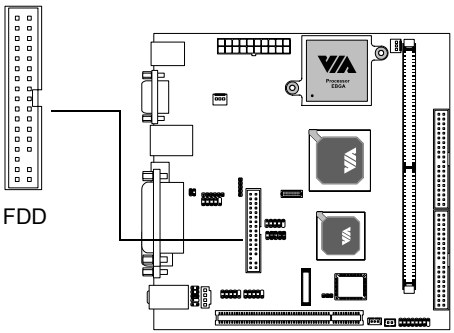
COM2: The Second Serial Port

COM2 is a pin header for second serial port.

Pin	Signal	Description
1	DCD	Data Carry Detect
2	SIN	Serial In or Receive Data
3	SOUT	Serial Out or Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicate

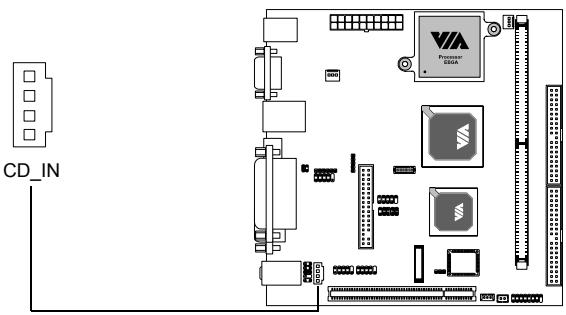
Floppy Disk Drive Connector: FDD

The floppy disk drive connector supports 360K, 720K, 1.2M. 1.44M, and 2.88M floppy disk types.



CD Audio Connector: CD_IN

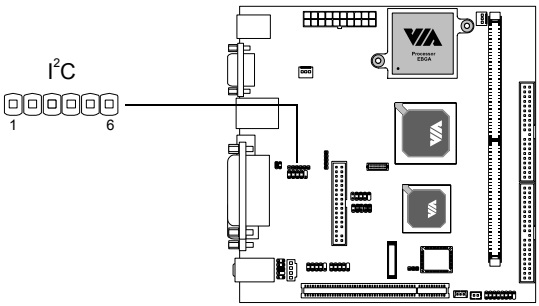
This connector is for the CD-ROM audio connector.



I²C Connector: I²C

This is for connecting a I²C device.

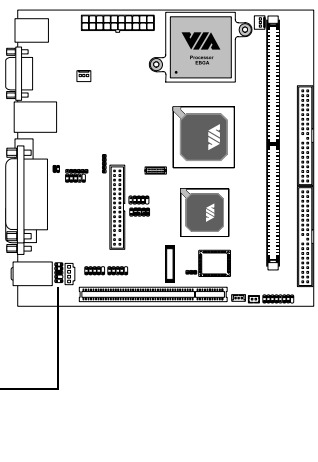
Pin	Signal
1	+3.3V
2	+3.3V
3	EL-ON
4	SMBCK
5	SMBDT
6	GND



Front Audio Panel: F_AUDIO

This connector allows you to connect a front audio panel to the mainboard. Only the line-out and microphone functions are available for use on the front panel. To connect the front audio cable, first remove the two red plastic jumpers.

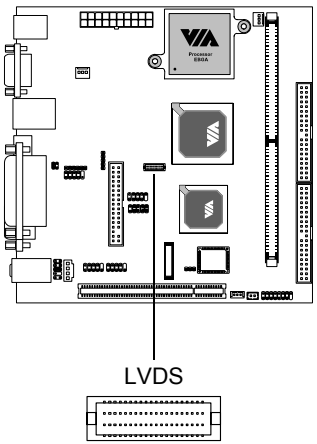
Pin	Signal	Pin	Signal
1	FRN_MIC	2	AGND
3	AUD_MIC_BIAS	4	+5V
5	LINE_OUT_R	6	Next_R
7	NC	8	Keypin
9	LINE_OUT_L	10	Next_L



LVDS Module Connector: LVDS (Optional)

This connector allows you to connect to a LVDS module. The LVDS connector may not be available on your mainboard. This is an option that is added during the manufacturing process. If you would like a mainboard with the LVDS connector, please contact your vendor or sales contact for more information.

Pin	Signal	Pin	Signal
1	GFPDE	2	GFPD3
3	GFPD0	4	GFPD4
5	GFPD1	6	GFPD5
7	GFPD2	8	GFPCLK
9	GFPHS	10	GFPD6
11	GFPVS	12	GFPD7
13	GFPD11	14	GFPD8
15	GFPD12	16	GFPD9
17	ENPVDD	18	GFPD10
19	ENPVEE	20	GFPD13
21	FPBKLP	22	GFPD14
23	PWRGD_SB	24	GFPD15
25	SPCLK2	26	GFPD16
27	SPD2	28	GFPD17
29	GND	30	GFPD18
31	GND	32	GFPD19
33	3.3V	34	GFPD20
35	GND	36	GFPD21
37	5V	38	GFPD22
39	5V	40	GFPD23

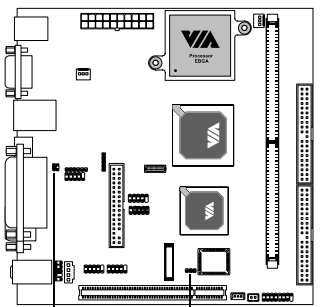


Jumpers

The mainboard provides jumpers for setting the mainboard's functions. This section will explain how to change settings for your mainboard's functions through the use of the jumpers.

Clear CMOS: CLEAR_CMOS

The onboard CMOS RAM stores system configuration data and has an onboard battery power supply. The long-life battery has a lifetime of at least 5 years. If you want to clear the system configuration data from the CMOS RAM, use the CLEAR_CMOS (Clear CMOS jumper). You can clear the CMOS by shorting 1-2 pin while the system is off. Then return it to the 2-3 pin position. Shorting the jumper while the system is on will damage the mainboard.



Setting	1	2	3
Clear	ON	ON	OFF
Keep	OFF	ON	ON

CLEAR_CMOS

SPDIF_SEL

RCA Video or S/PDIF Select: SPDIF_SEL

Users can select either RCA Video or S/PDIF as the enabled function on the dual-purpose port. For SPDIF, please short pins 1 and 2 (default). For RCA, short pins 3 and 4

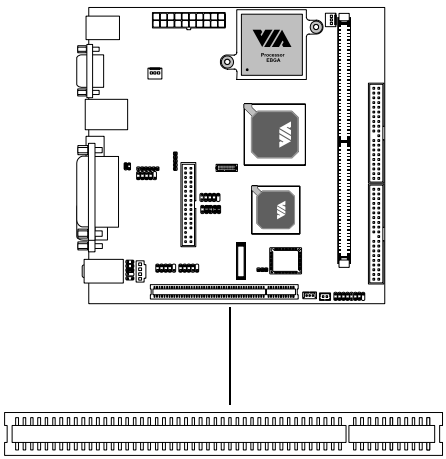
.

Setting	1	2	3	4
SPDIF	ON	ON	OFF	OFF
RCA	OFF	OFF	ON	ON

Slots

Peripheral Component Interconnect: PCI

The PCI slot allows you to insert PCI expansion card. When adding or removing expansion cards, make sure that you unplug the power supply first. Meanwhile, read the documentation for the expansion card to make any necessary hardware or software settings for the expansion card, such as jumpers, switches or BIOS configuration.



PCI Interrupt Request Routing

The IRQ, abbreviation of interrupt request line and pronounced I-R-Q, are hardware lines over which devices can send interrupt signals to the microprocessor. The "PCI & LAN" IRQ pins are typically connected to the PCI bus INT A# ~ INT D# pins as follows:

	Order 1	Order 2	Order 3	Order 4
PCI Slot	INT B#	INT C#	INT D#	INT A#
IEEE 1394	INT B#			

Chapter

3

BIOS Setup

This chapter gives you detailed explaination of each BIOS setup functions.

This chapter includes the following sections:

Entering Setup

Power on the computer and press **Delete** straight away to enter the BIOS setup menu. If you missed the BIOS setup entry point, you may restart the system and try again.

Control Keys

Keys	Description
Up Arrow	Move to the previous item
Down Arrow	Move to the next item
Left Arrow	Move to the item in the left side
Right Arrow	Move to the item in the right side
Enter	Select the item
Escape	Jumps to the Exit menu or returns to the main menu from a submenu
Page Up / +	Increase the numeric value or make changes
Page Down / -	Decrease the numeric value or make changes
F1	General help, only for Status Page Setup Menu and Option Page Setup Menu
F5	Restore the previous CMOS value from CMOS, only for Option Page Setup Menu
F6	Load the default CMOS value from Fail-Safe default table, only for Option Page Setup Menu
F7	Load Optimized defaults
F9	Jumps to the Main Menu
F10	Save all the CMOS changes and exit

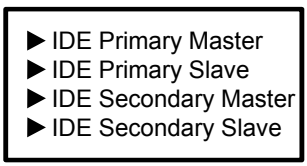
Getting Help

Main Menu

The main menu displays all BIOS setup categories. Use the control keys **Up/Down Arrow Keys** to select any item/sub-menu. Description of the selected/highlighted category is displayed at the bottom of the screen.

Sub-Menu

If you find a right pointer symbol (as shown in the right view) appears on the left of certain fields, this means a sub-menu is available. The sub-menu contains additional options. You can use control keys **Up/Down Arrow Keys** to highlight the field and press **Enter** to enter the sub-menu. To return from the sub-menu press **Esc**.

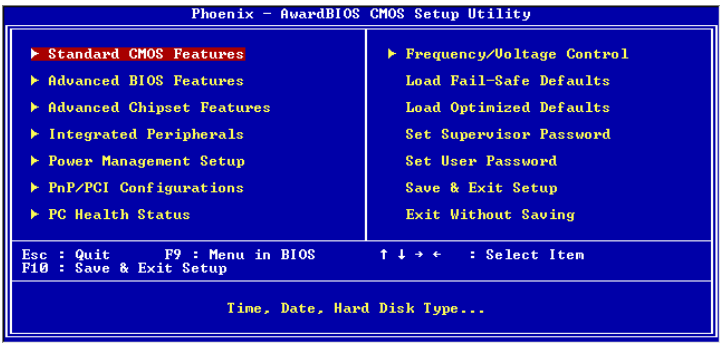


General Help: F1

The BIOS setup program provides a General Help screen. You can call up this screen from any menu/sub-menu by pressing **F1**. The help screen displays the keys for use and navigate the BIOS setup. Press **Esc** to exit the help screen.

The Main Menu

The Main Menu contains twelve setup functions and two exit choices. Use arrow keys to select the items and press **Enter** to accept or enter the sub-menu.



Standard CMOS Features

Use this menu to set basic system configurations.

Advanced BIOS Features

Use this menu to set the advanced features available on your system.

Advanced Chipset Features

Use this menu to set chipset specific features and optimize system performance.

Integrated Peripherals

Use this menu to set onboard peripherals features.

Power Management Setup

Use this menu to set onboard power management functions.

PnP/PCI Configurations

Use this menu to set the PnP and PCI configurations.

PC Health Status

This menu shows the PC health status.

Frequency/Voltage Control

Use this menu to set the system frequency and voltage control.

Load Fail-Safe Defaults

Use this menu option to load the BIOS default settings for minimal and stable system operations.

Load Optimized Defaults

Use this menu option to load BIOS default settings for optimal and high performance system operations.

Set Supervisor Password

Use this menu option to set the BIOS supervisor password.

Set User Password

Use this menu option to set the BIOS user password.

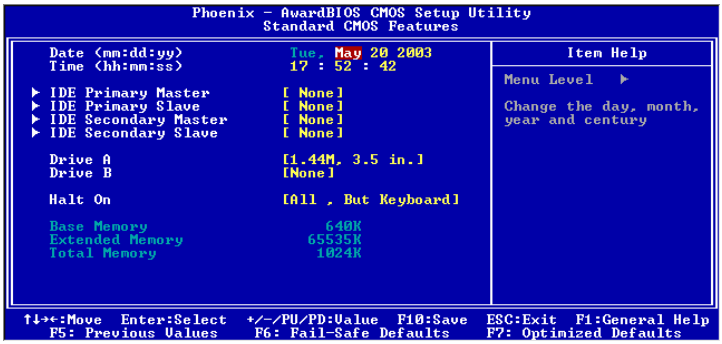
Save & Exit Setup

Save BIOS setting changes and exit setup.

Exit Without Saving

Abandon all BIOS setting changes and exit setup.

Standard CMOS Features



Date

The date format is <Day><Month><Date><Year>.

Day - day of the week, for example Friday. Read-only.

Month - the month from Jan to Dec.

Date - the date from 1 to 31.

Year - the year, range from 1999 to 2098.

Time

The time format is <Hour><Minute><Second>.

Drive A/B

Set the type of floppy drive installed. Settings: *None, 360K (5.25 in.), 1.2M (5.25 in.), 720K (3.5 in.), 1.44M (3.5 in.), 2.88M (3.5 in.)*

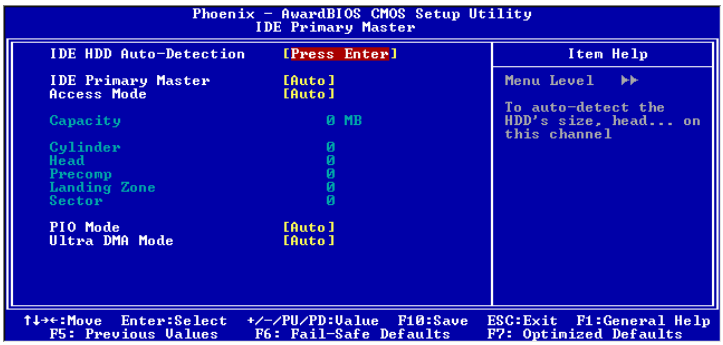
Halt On

Determine the system behaviour if an error is detected at boot. Settings are:

All Errors	System halts when any error is detected.
No Errors	System does not halt for any error.
All, But Keyboard	System halts for all non-key errors.
All, But Diskette	System halts for all non-disk errors.
All, But Disk/Key	System halts for all non-key and non-disk errors.

IDE Primary Master/Slave, Secondary Master/Slave

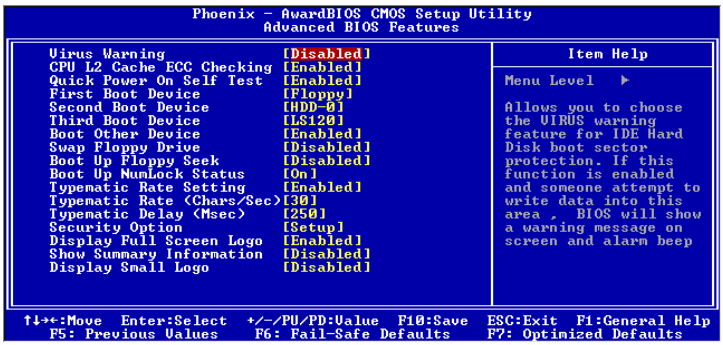
Press **Enter** to enter the sub-menu and the following screen appears:



The specifications of your drive must match with the drive table. The hard disk will not work properly if you enter improper information for this category. Select *Auto* whenever possible. If you select *Manual*, make sure the information provided is from your hard disk vendor or system manufacturer.

IDE <Primary Master>	The name of this menu item will match the name of the menu. The settings are *None*, *Auto*, *Manual*.
Access Mode	The settings are *CHS*, *LBA*, *Large*, *Auto*.
Capacity	The formatted size of the storage device.
Cylinder	Number of cylinders.
Head	Number of heads.
Precomp	Write precompensation.
Landing Zone	Cylinder location of the landing zone.
Sector	Number of sectors.
PIO Mode	The settings are *Mode 0/1/2/3/4*, *Auto*.
Ultra DMA Mode	The settings are *Disabled* and *Auto*.

Advanced BIOS Features



Virus Warning

Set the Virus Warning feature for IDE Hard Disk boot sector protection. If the function is enabled, any attempt to write data into this area will cause a beep and warning message display on screen. Settings: *Disabled* and *Enabled*

CPU L2 Cache ECC Checking

Set the ECC (Error-Correcting Code) feature for Level 2 cache. Facilitates error detection/correction when data passes through Level 2 cache. Settings: *Enabled* and *Disabled*

Quick Power On Self Test

Shorten Power On Self Test (POST) cycle and enable shorter bootup time. Allow BIOS to skip some check items during POST. Settings: *Enabled* and *Disabled*

First/Second/Third Boot Device

Set the boot device sequence as BIOS attempts to load the disk operating system. The settings are:

Floppy	The system will boot from floppy drive.
LS120	The system will boot from LS-120 drive.
HDD-0	The system will boot from first HDD.
SCSI	The system will boot from SCSI.
CD-ROM	The system will boot from CD-ROM.
HDD-1	The system will boot from second HDD.
HDD-2	The system will boot from third HDD.
HDD-3	The system will boot from fourth HDD.
ZIP100	The system will boot from ATAPI ZIP drive.
USB-FDD	The system will boot from USB floppy drive.
USB-ZIP	The system will boot from USB ZIP drive.
USB-CDROM	The system will boot from USB CDROM.
USB-HDD	The system will boot from USB HDD.
LAN	The system will boot from network drive.
Disabled	Disable this sequence.

Boot Other Device

Enable the system to boot from other devices if the system fails to boot from the First/Second/Third boot device. Settings: *Enabled* and *Disabled*

Swap Floppy Drive

If the system has two floppy drives, choose *Enabled* to assign physical drive B to logical drive A and vice versa. Settings: *Enabled* and *Disabled*

Boot Up Floppy Seek

Set floppy seek during POST, BIOS will determine whether the floppy is 40 or 80 tracks. Settings: *Enabled* and *Disabled*

Boot Up NumLock Status

Set the NumLock status when the system is powered on. *On* will turn key pad into number keys, and *Off* will turn key pad into arrow keys. Settings: *On* and *Off*

Typematic Rate Setting

When *Enabled*, you can set the Typematic Rate and Typematic Delay. Settings: *Enabled* and *Disabled*

Typematic Rate (Chars/Sec)

When Typematic Rate Setting is enabled, this item allows you to set the rate (characters/second) at which the keys are accelerated. Settings: *6, 8, 10, 12, 15, 20, 24* and *30*

Typematic Delay (Msec)

When Typematic Rate Setting is enabled, this item allows you to select the delay between when the key was first pressed and when the acceleration begins. Settings: *250, 500, 750* and *1000*

Security Option

If you have set a password, select whether the password is required every time the System boots, or only when you enter Setup. Settings are described below:

Setup	The password prompt appears only when end users try to run Setup.
System	A password prompt appears every time when the computer is powered on or when end users try to run Setup.

Display Full Screen Logo

Show full screen logo during BIOS bootup process. Settings: *Enabled* and *Disabled*

Show Summary Information

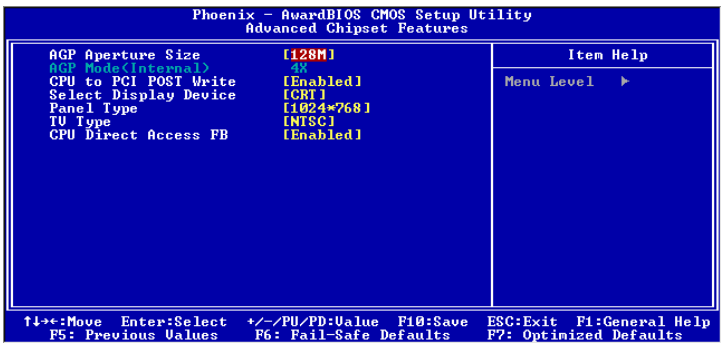
Show the summary information during the BIOS boot process. Settings: *Enabled* and *Disabled*

Display Small Logo

Show small energy star logo during BIOS bootup process. Settings: *Enabled* and *Disabled*

Advanced Chipset Features

The Advanced Chipset Features menu is used for optimizing the chipset functions.



WARNING: Do not change these settings unless you are familiar with the chipset.

AGP Aperture Size

This setting controls how much memory space can be allocated to AGP for video purposes. The aperture is a portion of the PCI memory address range dedicated to graphics memory address space. Host cycles that hit the aperture range are forwarded to the AGP without any translation. Settings: *4MB, 8MB, 16MB, 32MB, 64MB, 128MB,* and *256MB*

AGP Mode (Internal)

This mainboard supports the AGP 4x interface. AGP 4x can transfer video data at 1066MB/s and is backward-compatible with AGP2x and AGP1x.

CPU to PCI POST Write

When *Enabled*, CPU can write up to four words of data to the PCI write buffer before CPU must wait for PCI bus cycle to finish. If *Disabled*, CPU must wait after each write cycle until PCI bus signals that it is ready to receive more data. Settings: *Enabled* and *Disabled*

Select Display Device

This setting refers to the type of display being used with the system. Settings: *CRT, TV, CRT + TV, LCD* and *CRT + LCD*

Panel Type

This setting refers to the native resolution of the display being used with the system. Settings: *1600x1200, 1400.1050, 1280,1024, 1280x768,1024x768, 800x600* and *640x480*

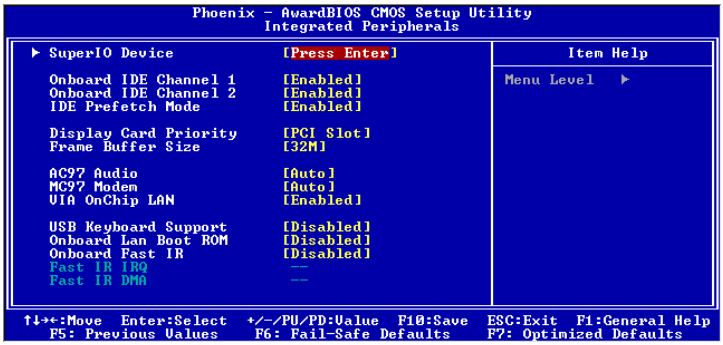
TV Type

This setting refers to the native resolution of the display being used with the system. Settings: *NTSC* and *PAL*

CPU Direct Access FB

Enable the CPU to directly access the frame buffer. Settings: *Enabled* and *Disabled*

Integrated Peripherals



Onboard IDE Channel 1/2

The integrated peripheral controller contains an IDE interface with support for two IDE channels. Choose *Enabled* to activate each channel separately. Settings: *Enabled* and *Disabled*

IDE Prefetch Mode

This allows your hard disk controller to use the fast block mode to transfer data to and from the hard disk drive. Block mode is also called block transfer, multiple commands or multiple sector read/write. *Enabled* enables IDE controller to use block mode; *Disabled* allows the controller to use standard mode. Settings: *Enabled* and *Disabled*

Display Card Priority

This setting specifies which VGA card is your primary graphics adapter. Settings: *PCI Slot* and *AGP*

Frame Buffer Size

This setting instructs the BIOS to reserved the specified amount of memory for the internal video controller. Settings: *16M, 32M, 64M*

AC97 Audio

Auto allows the mainboard to detect whether an audio device is used. If the device is detected, the onboard VIA AC'97 (Audio Codec'97) controller will be enabled; if not, it is disabled. Disable the controller if you want to use other controller cards to connect to an audio device. Settings: *Auto* and *Disabled*

MC97 Modem

Auto allows the mainboard to detect whether a modem is used. If the device is detected, the onboard VIA MC'97 (Modem Codec'97) controller will be enabled; if not, it is disabled. Disable the controller if you want to use other controller cards to connect to a modem. Settings: *Auto* and *Disabled*

VIA OnChip LAN

This setting allows you to make VIA OnChip LAN enabled or disabled. Settings: *Enabled* and *Disabled*

USB Keyboard Support

Enable USB Keyboard Support for DOS and Windows. Settings: *Enabled* and *Disabled*

Onboard Lan Boot ROM

Enable Onboard Lan Boot ROM for DOS and Windows. Settings: *Enabled* and *Disabled*

Onboard Fast IR

Enable Onboard Fast IR functions. Settings: *Enabled* and *Disabled*

Fast IR IRQ

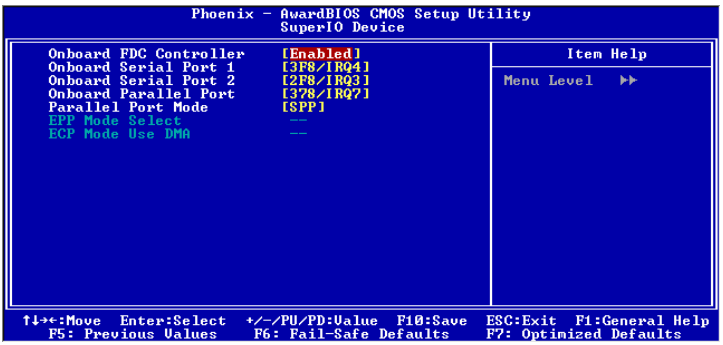
Set this field to reserve an IRQ for the Fast IR port. This field is only available if Onboard Fast IR is enabled. Settings: *3, 4*

Fast IR DMA

Set this field to choose the DMA channel. This field is only available if Onboard Fast IR is enabled. Settings: *6, 5*

SuperIO Device

Press **Enter** to enter the sub-menu and the following screen appears:



Onboard FDC Controller

Enable the onboard floppy controller. Select *Enabled* when you have installed a floppy disk drive. Settings: *Enabled, Disabled*

Onboard Serial Port 1/2

Set the base I/O port address and IRQ for the onboard serial port A/serial port B. Selecting *Auto* allows BIOS to automatically determine the correct base I/O port address. Settings:

Port	Settings					
1	Disabled	3F8/IRQ4	2F8/IRQ3	3E8/IRQ4	2E8/IRQ3	Auto
2	Disabled	3F8/IRQ4	2F8/IRQ3	3E8/IRQ4	2E8/IRQ3	Auto

Onboard Parallel Port

This specifies the I/O port address and IRQ of the onboard parallel port. Settings: *Disabled, 378/IRQ7, 278/IRQ5, 3BC/IRQ7*

Parallel Port Mode

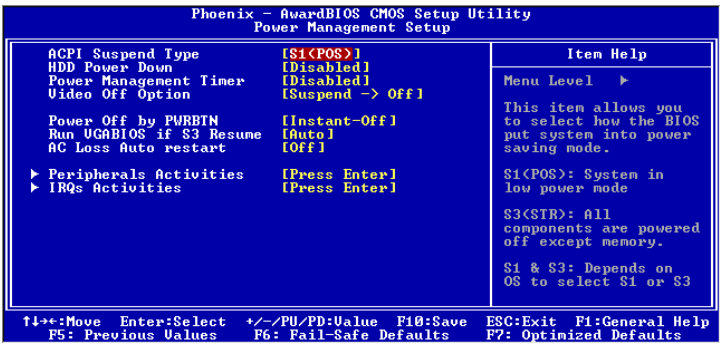
Set the parallel port mode. To operate the onboard parallel port as Standard Parallel Port, choose *SPP*. To operate the onboard parallel port in the EPP mode, choose *EPP*. By choosing *ECP*, the onboard parallel port will operate in ECP mode. Choosing *ECP + EPP* will allow the onboard parallel port to support both the ECP and EPP modes simultaneously. Settings: *SPP, EPP, ECP, ECP + EPP*

EPP Mode Select
EPP (Enhanced Parallel Port) comes in two modes: 1.9 and 1.7. EPP 1.9 is the newer version of the protocol and is backwards compatible with most EPP devices. If your EPP device does not work with the EPP 1.9 setting, try changing the setting to EPP 1.7.
Settings: *EPP 1.9, EPP 1.7*

ECP Mode Use DMA
ECP (Extended Capabilities Port) has two DMA channels that it can use. The default channel is 3. However, some expansion cards may use channel 3 as well. To solve this conflict, change the ECP channel to 1.
Select a DMA channel for the port. Settings: *1, 3*

Power Management Setup

The Power Management Setup menu configures the system to most effectively save energy while operating in a manner consistent with your own style of computer use.



ACPI Function

Activate the ACPI (Advanced Configuration and Power Management) Function. If your operating system is ACPI-aware (i.e. Windows 98/98SE/ ME/2000/XP) select Enabled. Settings: *Enabled* and *Disabled*

ACPI Suspend Type

Set the power saving mode for ACPI function. Settings are:

S1(POS)	S1/Power On Suspend (POS) is a low power state. In this state, no system context (CPU or chipset) is lost and hardware maintains all system context.
S3(STR)	S3/Suspend To RAM (STR) is a power-down state. In this state, power is supplied only to essential components such as main memory and wakeup-capable devices. The system context is saved to main memory, and context is restored from the memory when a “wakeup” event occurs.
S1 & S3	Depends on OS to select S1 or S3.

HDD Power Down

Set the time to power down HDD after hard disk inactivity. Settings: *Disabled* and *1~15* (minutes)

Power Management Timer

Set the idle time before system enters power saving mode. ACPI OS such as Windows XP will override this option. Settings: *Disabled* and *1/2/4/6/8/10/20/30/40* (minutes) and *1* (hour)

Video Off Option

Select whether or not to turn off the screen when system enters power saving mode, ACPI OS such as Windows XP will override this option. Settings are:

Always On	The screen is always on even when system enters power saving mode.
Suspend -> Off	The screen is turned off when system enters power saving mode.

Power Off by PWRBTN

This field configures the power button function. Settings are:

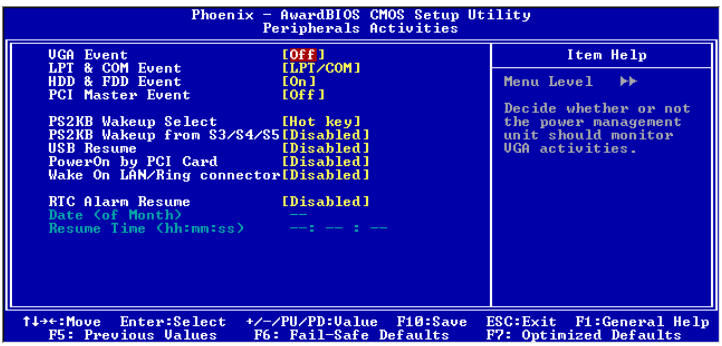
Delay 4 Sec	The system is turned off if power button is pressed for more than four seconds.
Instant-Off	The power button functions as a normal power-on/-off buttton.

Run VGABIOS if S3 Resume

Select whether to run VGA BIOS if resumed from S3 state. This is only necessary for older VGA drivers, select *Auto* if in doubt. Settings: *Auto, Yes* and *No*

Peripheral Activities

Press **Enter** to enter the sub-menu and the following screen appears:



VGA Event

Decide whether or not the power management unit should monitor VGA activities. Settings: *Off* and *On*

LPT & COM Event

Decide whether or not the power management unit should monitor parallel port (LPT) and serial port (COM) activities. Settings: *None, LPT, COM* and *LPT/COM*

HDD & FDD Event

Decide whether or not the power management unit should monitor hard disks and floppy drives activities. Settings: *Off* and *On*

PCI Master Event

Decide whether or not the power management unit should monitor PCI master activities. Settings: *Off* and *On*

PS2KB Wakeup Select

When select *Password*, please press **Page Up** or **Page Down** key to change *Password*, 8 characters maximum. Please note that PS2MS Wakeup from suspend and PS2KB Wakeup from suspend will be disabled while changing the passward. Settings: *Hot key* and *Password*

PS2KB Wakeup from suspend
Select which Hot-Key to wake-up the system from power saving mode. Settings: *Disabled, Ctrl+F1, Ctrl+F2, Ctrl+F3, Ctrl+F4, Ctrl+F5, Ctrl+F6, Ctrl+F7, Ctrl+F8, Ctrl+F9, Ctrl+F10, Ctrl+F11, Ctrl+F12, Power, Wake* and *Any Key*

USB Resume
Decide whether or not USB devices can wake the system from suspend state. Settings: *Disabled* and *Enabled*

PowerOn by PCI Card
Decide whether or not any PCI card can power up the system or resume from suspend state. Such PCI cards include LAN, onboard USB ports, etc. Settings: *Disabled* and *Enabled*

Wake On LAN/Ring Connector
Decide whether or not any Ring-In signals from the modem can wake up the system or resume from suspend state. Settings: *Disabled* and *Enabled*

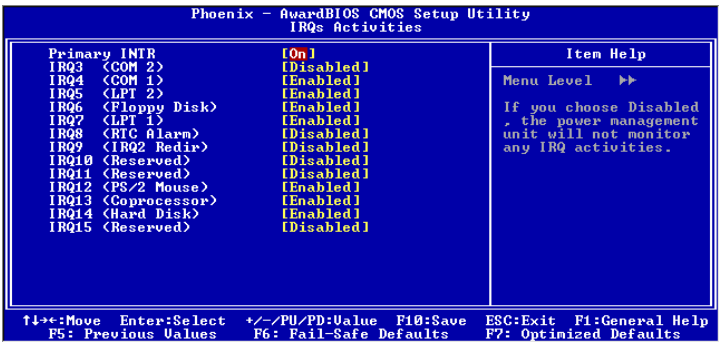
RTC Alarm Resume
The field is used to enable or disable the feature of booting up the system on a scheduled time/date. Settings: *Disabled* and *Enabled*

Date (of Month)
This field can only be set if RTC Alarm Resume is enabled. The field specifies the date for *RTC Alarm Resume*.

Resume Time (hh:mm:ss)
This field can only be set if RTC Alarm Resume is enabled. The field specifies the time for *RTC Alarm Resume*.

IRQs Activities

Press **Enter** to enter the sub-menu and the following screen appears:



Primary INTR

Selecting *On* will cause the system to wake up from power saving modes if activity is detected from any enabled IRQ channels. Settings: *Off, On*

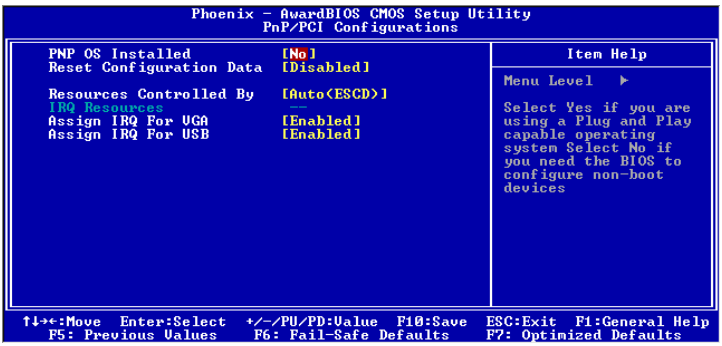
IRQ3~IRQ15

Enables or disables the monitoring of the specified IRQ line. If set to *Enabled*, the activity of the specified IRQ line will prevent the system from entering power saving modes or awaken it from power saving modes. These fields are only available if Primary INTR is on. Settings: *Enabled* and *Disabled*

Note: IRQ (Interrupt Request) lines are system resources allocated to I/O devices. When an I/O device needs to gain attention of the operating system, it signals this by causing an IRQ to occur. After receiving the signal, when the operating system is ready, the system will interrupt itself and perform the service required by the IO device.

PNP/PCI Configurations

This section describes the BIOS configuration of the PCI bus system. This section covers some very technical items and it is strongly recommended that only experienced users should make any changes to the default settings.



PNP OS Installed

When set to *Yes*, BIOS will only initialize the PnP cards used for booting (VGA, IDE, SCSI). The rest of the cards will be initialized by the PnP operating system like Windows® 95 or 98/98SE. When set to *No*, BIOS will initialize all the PnP cards. Set to *Yes* the operating system is Plug & Play capable. Settings: *No* and *Yes*

Reset Configuration Data

Normally, you leave this field *Disabled*. Select *Enabled* to reset Extended System Configuration Data (ESCD) when you exit Setup if you have installed a new add-on and the system reconfiguration has caused such a serious conflict that the operating system can not boot. Settings: *Enabled* and *Disabled*

Resource Controlled By

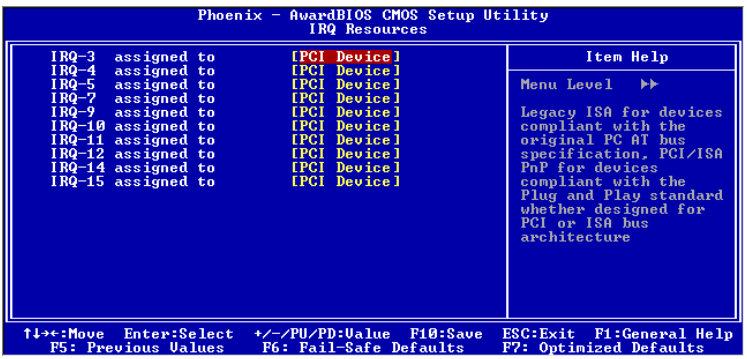
The BIOS can automatically configure all the boot and Plug and Play compatible devices. Choose *Auto(ESCD)* if unsure, the BIOS will automatically assign IRQ, DMA and memory base address fields. Settings: *Auto (ESCD)* and *Manual*

Assign IRQ For VGA/USB

Assign IRQ for VGA and USB devices. Settings: *Disabled* and *Enabled*

IRQ Resources

The items are adjustable only when *Resources Controlled By* is set to *Manual*. Press **Enter** and you will enter the sub-menu of the items.



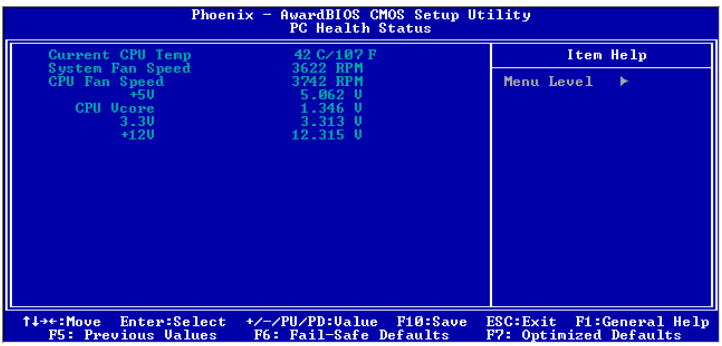
IRQ Resources list IRQ 3/4/5/7/9/10/11/12/14/15 for users to set each IRQ a type depending on the type of device using the IRQ. Settings:

PCI Device For Plug & Play compatible devices designed for PCI bus architecture.

Reserved The IRQ will be reserved for further request.

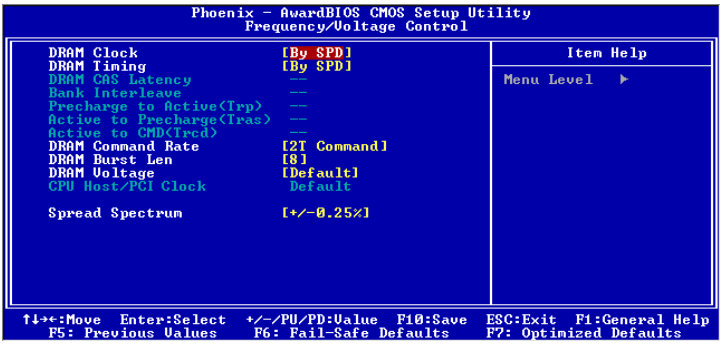
PC Health Status

This section shows the status of your CPU, fan, warning for overall system status.



The PC Health Status displays the current status of all of the monitored hardware devices/components such as CPU voltages, temperatures and fan speeds.

Frequency / Voltage Control



DRAM Clock

The chipset supports synchronous and asynchronous mode between host clock and DRAM clock frequency. Settings: *66 MHz, 100 MHz, 133 MHz,* and *By SPD*

DRAM Timing

The value in this field depends on the memory modules installed in your system. Changing the value from the factory setting is not recommended unless you install new memory that has a different performance rating than the original modules. Settings: *Manual* and *By SPD*

DRAM CAS Latency

This item adjusts the speed it takes for the memory module to complete a command. Generally, a lower setting will improve the performance of your system. However, if your system becomes less stable, you should change it to a higher setting. This field is only available when DRAM Timing is set to Manual. Settings: *2, 2.5*

Bank Interleave

Set the interleave mode of the SDRAM interface. Interleaving allows banks of SDRAM to alternate their refresh and access cycles. One bank will undergo its refresh cycle while another is being accessed. This improves performance of the SDRAM by masking the refresh time of each bank. This field is only available when DRAM Timing is set to Manual. Settings: *Disabled, 2 Bank, 4 Bank*

Precharge to Active (Trp)

This field controls the length of time it takes to precharge a row in the memory module before the row becomes active. Longer values are safer but may not offer the best performance. This field is only available when DRAM Timing is set to Manual. Settings: *2T, 3T*

Active to Precharge (Tras)

This field controls the length of time it a row stays active before precharging. Longer values are safer buy may not offer the best performance. This field is only available when DRAM Timing is set to Manual. Settings: *5T, 6T*

Active to CMD (Trcd)

This field is only available when DRAM Timing is set to Manual. Settings: *2T, 3T*

DRAM Command Rate

This field controls how fast the memory controller sends out commands. Lower setting equals faster command rate. Please note that some memory modules may not be able to handle lower settings. Settings: *2T Command, 1T Command*

DRAM Burst Len

This field sets the length of time for one burst of data during a read/write transaction. Longer settings equals better memory performance. Settings: *4, 8*

DRAM Voltage

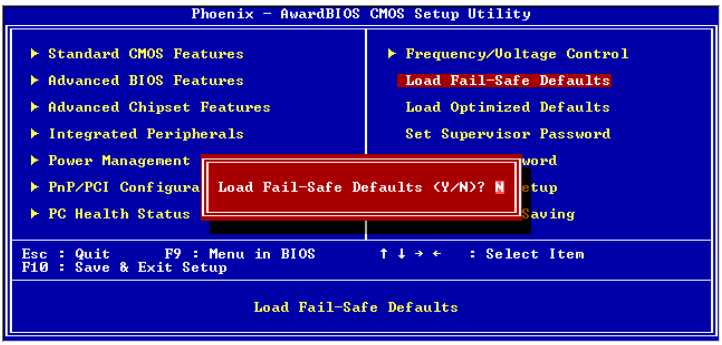
This field sets the voltage for the memory module. Settings: *2.9V, 2.8V, 2.6V, 2.5V*

Spread Spectrum

When the mainboard's clock generator pulses, the extreme values (spikes) of the pulses creates EMI (Electromagnetic Interference). The Spread Spectrum function reduces the EMI generated by modulating the pulses so that the spikes of the pulses are reduced to flatter curves.

Load Fail-Safe Defaults

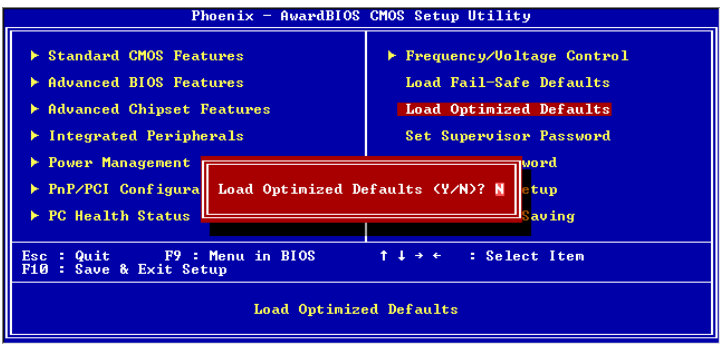
This option on the main menu allows users to restore all the BIOS settings to the default Fail Safe values. These values are set by the mainboard manufacturer to provide a minimal and stable system.
When you select Load-Fail Safe Defaults, a message as below appears:



Entering *Y* loads the default BIOS values that provide a minimal and stable system configuration.

Load Optimized Defaults

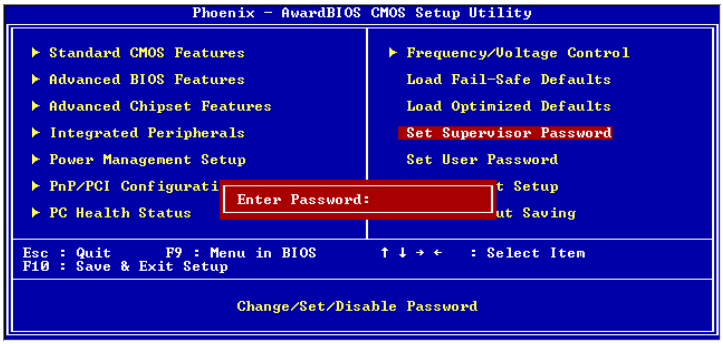
This option on the main menu allows users to restore all the BIOS settings to the default Optimized values. The Optimized Defaults are the default values also set by the mainboard manufacturer for both optimized and stable performance of the mainboard.
When you select Load Optimized Defaults, a message as below appears:



Entering *Y* loads the default values that are factory settings for optimal and stable system performance.

Set Supervisor / User Password

When you select this function, a message as below will appear on the screen:



Type the password, up to eight characters in length, and press **Enter**. The password typed now will clear any previously set password from CMOS memory. You will be prompted to confirm the password. Re-type the password and press **Enter**. You may also press **Esc** to abort the selection and not enter a password.

To clear a set password, just press **Enter** when you are prompted to enter the password. A message will show up confirming the password will be disabled. Once the password is disabled, the system will boot and you can enter Setup without entering any password.

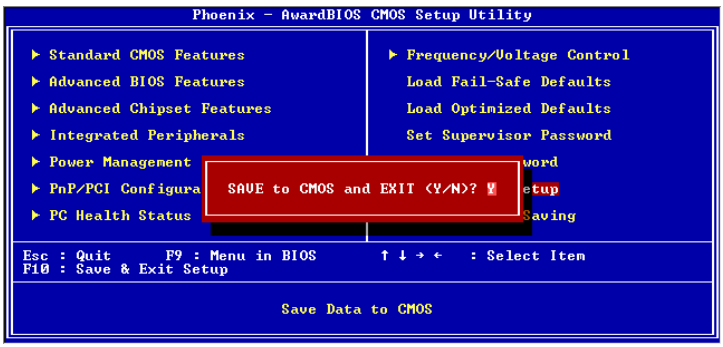
When a password has been set, you will be prompted to enter it every time you try to enter Setup. This prevents an unauthorized person from changing any part of your system configuration.

There are two types of passwords you can set. A Supervisor password and a User password. When a Supervisor password is used, the user can start BIOS Setup program and change the settings of the setup menus. When a User password is used, the user can start the BIOS Setup program but does not have the right to change the settings of the setup menus.

Additionally, when a password is enabled, you can also have BIOS to request a password each time the system is booted. This would prevent unauthorized use of your computer. The setting to determine when the password prompt is required is the Security Option of the Advanced BIOS Features menu. If the Security Option is set to *System*, the password is required both at boot and at entry to Setup. If set to *Setup*, password prompt only occurs when trying to enter Setup.

Save & Exit Setup

When you want to quit the Setup menu, you can select this option to save the changes and quit. A message as below will appear on the screen:

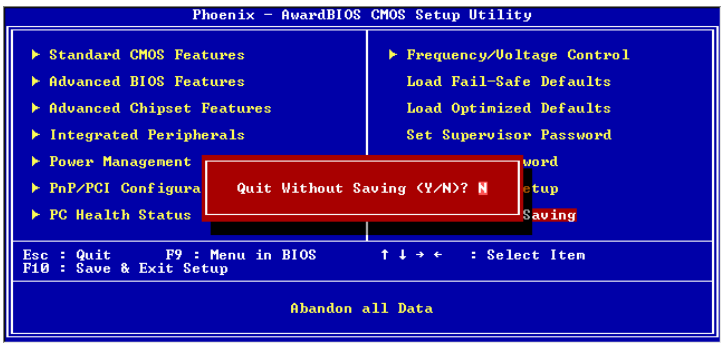


Entering *Y* will allow you to quit the Setup Utility and save the user setup changes to RTC CMOS.

Entering *N* will return to the Setup Utility.

Exit Without Saving

When you want to quit the Setup menu, you can select this option to abandon the changes. A message as below will appear on the screen:



Entering *Y* will allow you to quit the Setup Utility without saving any changes to RTC CMOS.

Entering *N* will return to the Setup Utility.

Chapter

4

Driver Installation

This chapter gives you brief descriptions of each mainboard drivers and applications. You must install VIA chipset drivers first before installing other drivers such as audio or VGA drivers. The applications will only function correctly if the necessary drivers are already installed.

This chapter includes the following sections:

Driver Utilities

Getting Started

The mainboard includes a Driver Utilities CD which contains driver utilities and software to enhance the performance of the mainboard. Please check that you have this CD in your retail box. If the CD is missing in your retail box, please contact your local dealer for the CD.

Note: The driver utilities and software are updated from time to time. Please visit our website (http://www.viaembedded.com/) for the latest updated mainboard driver and utilities.

Running the Driver Utilities CD

To start using the CD, just simply insert the CD into your local CD-ROM or DVD-ROM drive. The CD should run automatically when you close your CD-ROM or DVD-ROM drive. The driver utilities and software menu screen should then appear on your desktop. If the CD does not run automatically, you can run the CD manually by typing “D:\Setup.exe” at Start\Run.

(Please note that D: might not be your CD-ROM/DVD-ROM drive letter. Make sure you type the correct letter of CD-ROM/DVD-ROM drive on your system).

CD Content

The driver utilities and software in this CD are:

- **VIA 4in1 Drivers**: Contains VIA ATAPI Vendor Support Driver (enables the performance enhancing bus mastering functions on ATA-capable Hard Disk Drives and ensures IDE device compatibility), AGP VxD Driver (provides service routines to your VGA driver and interface directly to hardware, providing fast graphical access), IRQ Routing Miniport Driver (sets the system's PCI IRQ routing sequence) and VIA INF Driver (enables the VIA Power Management function).
- **VIA Graphics Driver:** Enhance the onboard VIA graphic chip.
- **VIA Audio Driver**: Enhance the onboard VIA audio chip.
- **VIA USB 2.0 Driver**: Enhance VIA USB 2.0 ports.
- **VIA LAN Driver**: Enhance the onboard VIA LAN chip.
- **VIA FIR Driver:** Support for FIR.

Appendix A

Smart 5.1

This chapter gives you brief description of how Smart 5.1 is enabled.

This chapter includes the following sections:

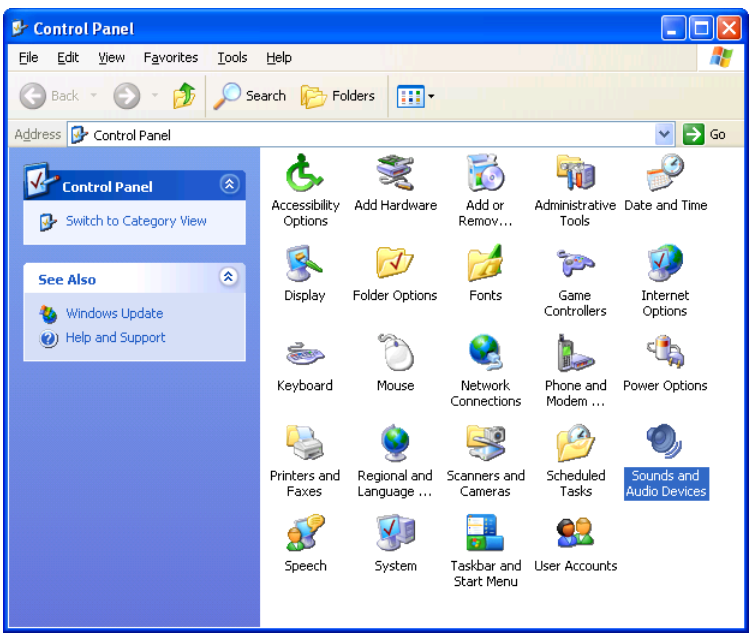
Intelligent 6-Channel Audio

Enabling Smart 5.1 Intelligent 6 Channel Audio

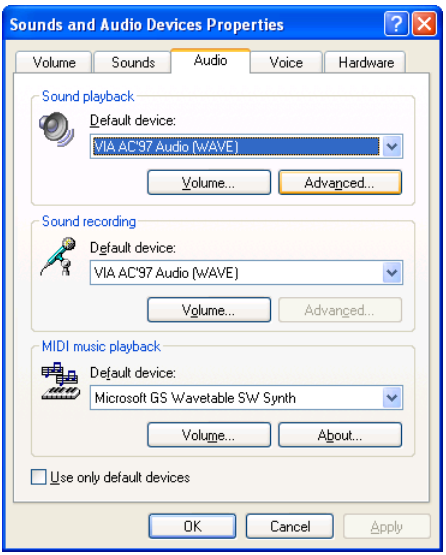
Smart5.1 allows the user to output 6 channel audio directly from the audio jacks on the mainboard, using the traditional line-in and microphone jacks as output jacks. For it to work properly, both the OS and the software application used need to support 6 channel audio. Win98 supports 4 channel only.Please follow the example A and B to enable the Smart 5.1 function, and the examples are based on Windows XP. Start the settings in Control Panel of your computer.

Example A

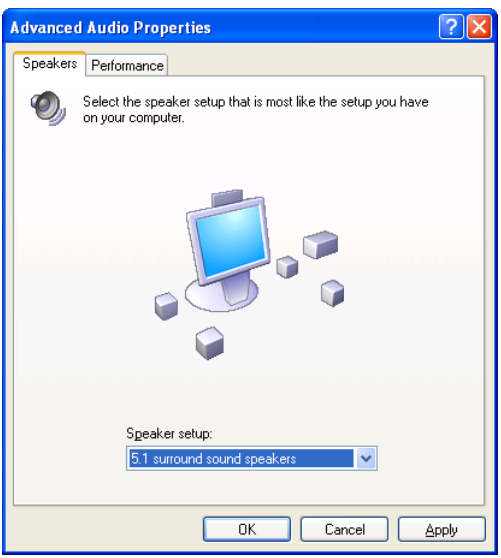
1. Double click [Sounds and Audio Devices] icon in Control Panel.



2. The panel of [Sounds and Audio Devices Properties] appears and select [Audio] tab. Then press [Advanced] as shown in the picture.

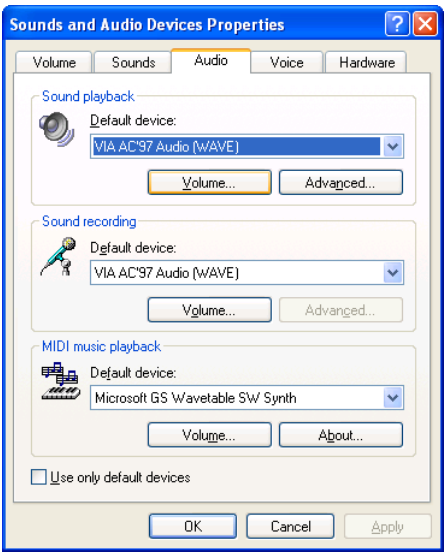


3. Choose [5.1 surround sound speakers] to support the 6 channel function.

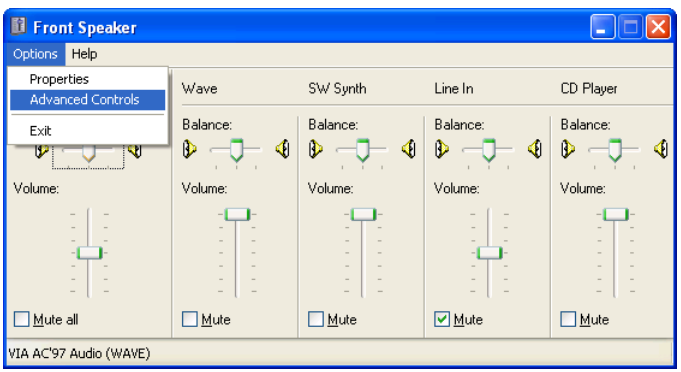


Example B

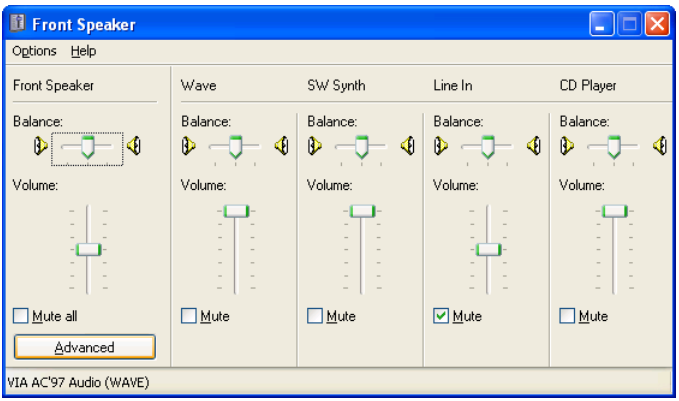
1. Double click [Sounds and Audio Devices] icon in Control Panel and then select [Audio] tab on the panel as shown below. Press [Volume] button in the [Sound playback] column.



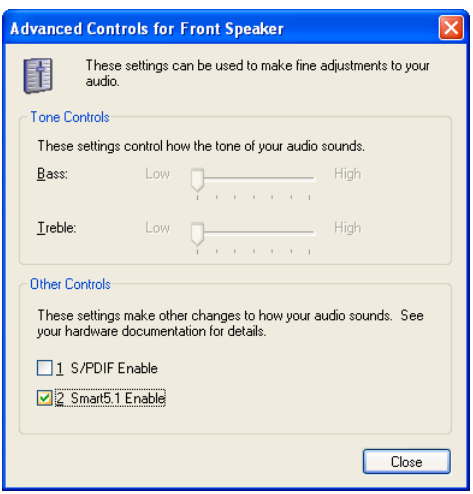
2. [Front Speaker] panel appears and then select [Options] menu to check the item [Advanced Controls].



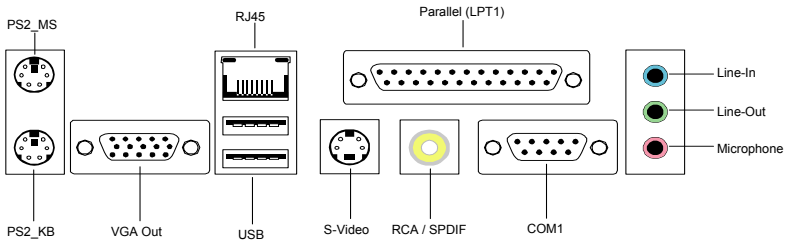
3. Then [Front Speaker] panel displays [Advanced] button and press it.

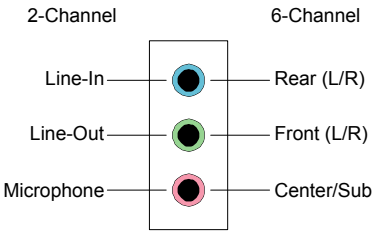


4. Check the item [Smart5.1 Enable] in the panel below.



After completing the previous settings, you need to connect your speakers to the audio jacks as shown below.





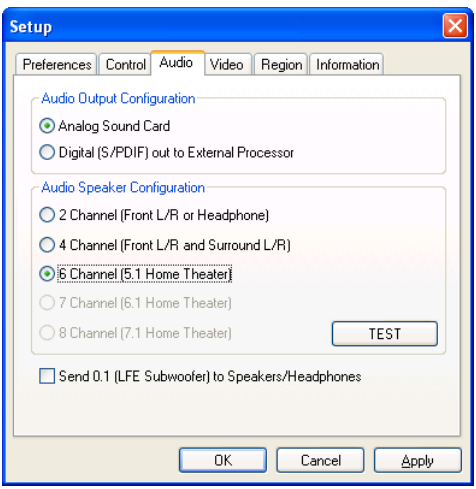
Following the system setup, users need to install software for playing DVD. Currently the two main DVD-playing applications are WIN-DVD v4.0 and Power DVD XP v4.0. Both of them are able to support 5.1 channel. Please follow the instructions below to do the proper settings for Smart 5.1.

WIN-DVD v4.0

1. Open the application and click on the right arrow icon shown as the picture below. Then select [Audio Effect].

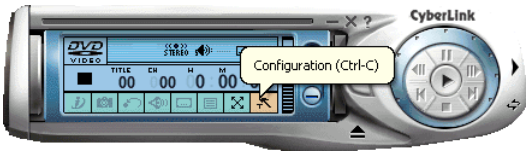
2. The panel of Audio Effect appears and click on the lower right corner button as shown in the picture below.

3. The [Setup] panel appears and select [Audio] tab. Then choose the item [6 Channel (5.1 Home Theater)] in the column of [Audio Speaker Configuration]. Finally users can click [Test] button to verify the channel output. You will hear sound of flowing water from different speakers if each setup has been completed successfully.

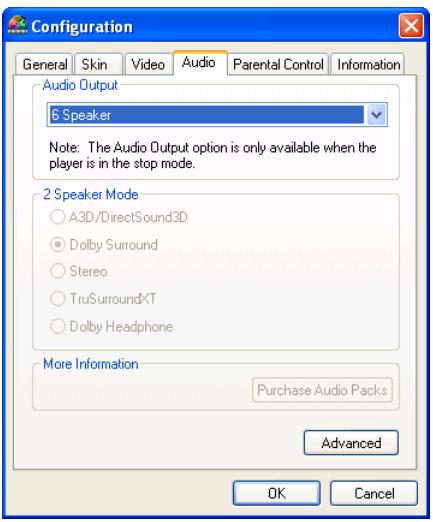


Power-DVD XP v4.0

1. Open the application and click on the [Configuration] icon shown as the picture below.



2. The panel of Configuration appears and select [Audio] tab. Then choose [6 Speaker] in the column of [Audio Output] and click [Ok].



Through the system operation and software settings, users can take advantage of Smart 5.1 6-channel output with ease!

National Semiconductor

June 1999

LM2576/LM2576HV Series
SIMPLE SWITCHER® 3A Step-Down Voltage Regulator

General Description

The LM2576 series of regulators are monolithic integrated circuits that provide all the active functions for a step-down (buck) switching regulator, capable of driving 3A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3V, 5V, 12V, 15V, and an adjustable output version.

Requiring a minimum number of external components, these regulators are simple to use and include internal frequency compensation and a fixed-frequency oscillator.

The LM2576 series offers a high-efficiency replacement for popular three-terminal linear regulators. It substantially reduces the size of the heat sink, and in some cases no heat sink is required.

A standard series of inductors optimized for use with the LM2576 are available from several different manufacturers. This feature greatly simplifies the design of switch-mode power supplies.

Other features include a guaranteed ±4% tolerance on output voltage within specified input voltages and output load conditions, and ±10% on the oscillator frequency. External shutdown is included, featuring 50 µA (typical) standby current. The output switch includes cycle-by-cycle current limiting, as well as thermal shutdown for full protection under fault conditions.

Features

- 3.3V, 5V, 12V, 15V, and adjustable output versions
- Adjustable version output voltage range, 1.23V to 37V (57V for HV version) ±4% max over line and load conditions
- Guaranteed 3A output current
- Wide input voltage range, 40V up to 60V for HV version
- Requires only 4 external components
- 52 kHz fixed frequency internal oscillator
- TTL shutdown capability, low power standby mode
- High efficiency
- Uses readily available standard inductors
- Thermal shutdown and current limit protection
- P+ Product Enhancement tested

Applications

- Simple high-efficiency step-down (buck) regulator
- Efficient pre-regulator for linear regulators
- On-card switching regulators
- Positive to negative converter (Buck-Boost)

Typical Application (Fixed Output Voltage Versions)

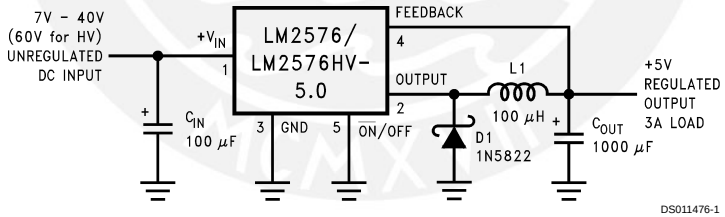
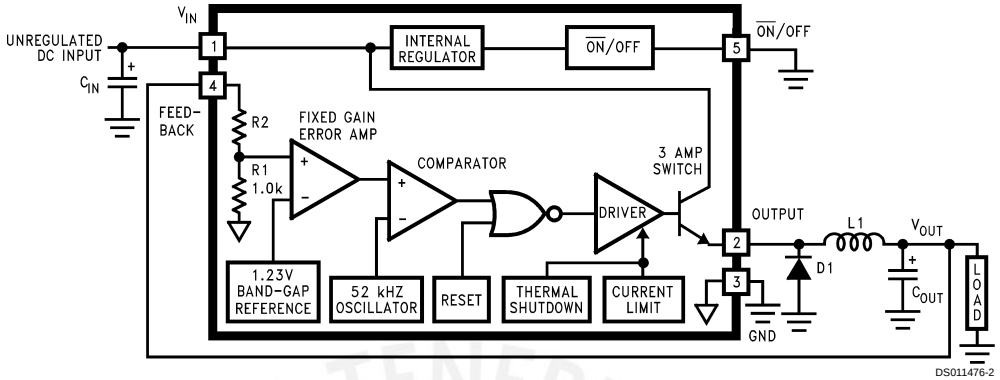


FIGURE 1.

SIMPLE SWITCHER® is a registered trademark of National Semiconductor Corporation.

Block Diagram



3.3V R2 = 1.7k
5V, R2 = 3.1k
12V, R2 = 8.84k
15V, R2 = 11.3k
For ADJ. Version
R1 = Open, R2 = 0Ω
Patent Pending

Ordering Information

Temperature Range	Output Voltage					NS Package Number	Package Type
	3.3	5.0	12	15	ADJ		
−40°C ≤ T_A ≤ 125°C	LM2576HVS-3.3	LM2576HVS-5.0	LM2576HVS-12	LM2576HVS-15	LM2576HVS-ADJ	TS5B	TO-263
	LM2576S-3.3	LM2576S-5.0	LM2576S-12	LM2576S-15	LM2576S-ADJ		
	LM2576HVSX-3.3	LM2576HVSX-5.0	LM2576HVSX-12	LM2576HVSX-15	LM2576HVSX-ADJ	TS5B Tape & Reel	
	LM2576SX-3.3	LM2576SX-5.0	LM2576SX-12	LM2576SX-15	LM2576SX-ADJ		
	LM2576HVT-3.3	LM2576HVT-5.0	LM2576HVT-12	LM2576HVT-15	LM2576HVT-ADJ	T05A	TO-220
	LM2576T-3.3	LM2576T-5.0	LM2576T-12	LM2576T-15	LM2576T-ADJ		
	LM2576HVT-3.3 Flow LB03	LM2576HVT-5.0 Flow LB03	LM2576HVT-12 Flow LB03	LM2576HVT-15 Flow LB03	LM2576HVT-ADJ Flow LB03	T05D	
	LM2576T-3.3 Flow LB03	LM2576T-5.0 Flow LB03	LM2576T-12 Flow LB03	LM2576T-15 Flow LB03	LM2576T-ADJ Flow LB03		

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Maximum Supply Voltage	
LM2576	45V
LM2576HV	63V
$\overline{ON}$ /OFF Pin Input Voltage	$-0.3V \le V \le +V_{IN}$
Output Voltage to Ground	
(Steady State)	−1V
Power Dissipation	Internally Limited
Storage Temperature Range	−65˚C to +150˚C
Maximum Junction Temperature	150˚C
Minimum ESD Rating	
(C = 100 pF, R = 1.5 kΩ)	2 kV
Lead Temperature	
(Soldering, 10 Seconds)	260˚C

Operating Ratings

Temperature Range	
LM2576/LM2576HV	$-40˚C \le T_J \le +125˚C$
Supply Voltage	
LM2576	40V
LM2576HV	60V

LM2576-3.3, LM2576HV-3.3 Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-3.3 LM2576HV-3.3		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Output Voltage	V_{IN} = 12V, I_{LOAD} = 0.5A Circuit of *Figure 2*	3.3	 3.234 3.366	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576	6V ≤ V_{IN} ≤ 40V, 0.5A ≤ I_{LOAD} ≤ 3A Circuit of *Figure 2*	3.3	 3.168/**3.135** 3.432/**3.465**	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576HV	6V ≤ V_{IN} ≤ 60V, 0.5A ≤ I_{LOAD} ≤ 3A Circuit of *Figure 2*	3.3	 3.168/**3.135** 3.450/**3.482**	V V(Min) V(Max)
η	Efficiency	V_{IN} = 12V, I_{LOAD} = 3A	75		%

LM2576-5.0, LM2576HV-5.0 Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with *Figure 2* **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-5.0 LM2576HV-5.0		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Output Voltage	V_{IN} = 12V, I_{LOAD} = 0.5A Circuit of *Figure 2*	5.0	 4.900 5.100	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576	0.5A ≤ I_{LOAD} ≤ 3A, 8V ≤ V_{IN} ≤ 40V Circuit of *Figure 2*	5.0	 4.800/**4.750** 5.200/**5.250**	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576HV	0.5A ≤ I_{LOAD} ≤ 3A, 8V ≤ V_{IN} ≤ 60V Circuit of *Figure 2*	5.0	 4.800/**4.750** 5.225/**5.275**	V V(Min) V(Max)
η	Efficiency	V_{IN} = 12V, I_{LOAD} = 3A	77		%

LM2576-12, LM2576HV-12 Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-12 LM2576HV-12		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Output Voltage	V_{IN} = 25V, I_{LOAD} = 0.5A Circuit of *Figure 2*	12	 11.76 12.24	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576	0.5A ≤ I_{LOAD} ≤ 3A, 15V ≤ V_{IN} ≤ 40V Circuit of *Figure 2*	12	 11.52/**11.40** 12.48/**12.60**	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576HV	0.5A ≤ I_{LOAD} ≤ 3A, 15V ≤ V_{IN} ≤ 60V Circuit of *Figure 2*	12	 11.52/**11.40** 12.54/**12.66**	V V(Min) V(Max)
η	Efficiency	V_{IN} = 15V, I_{LOAD} = 3A	88		%

LM2576-15, LM2576HV-15 Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-15 LM2576HV-15		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Output Voltage	V_{IN} = 25V, I_{LOAD} = 0.5A Circuit of *Figure 2*	15	 14.70 15.30	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576	0.5A ≤ I_{LOAD} ≤ 3A, 18V ≤ V_{IN} ≤ 40V Circuit of *Figure 2*	15	 14.40/**14.25** 15.60/**15.75**	V V(Min) V(Max)
V_{OUT}	Output Voltage LM2576HV	0.5A ≤ I_{LOAD} ≤ 3A, 18V ≤ V_{IN} ≤ 60V Circuit of *Figure 2*	15	 14.40/**14.25** 15.68/**15.83**	V V(Min) V(Max)
η	Efficiency	V_{IN} = 18V, I_{LOAD} = 3A	88		%

LM2576-ADJ, LM2576HV-ADJ Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-ADJ LM2576HV-ADJ		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Feedback Voltage	V_{IN} = 12V, I_{LOAD} = 0.5A V_{OUT} = 5V, Circuit of *Figure 2*	1.230	 1.217 1.243	V V(Min) V(Max)

LM2576-ADJ, LM2576HV-ADJ
Electrical Characteristics (Continued)

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range.

Symbol	Parameter	Conditions	LM2576-ADJ LM2576HV-ADJ		Units (Limits)
			Typ	Limit (Note 2)	
SYSTEM PARAMETERS (Note 3) **Test Circuit** *Figure 2*					
V_{OUT}	Feedback Voltage LM2576	0.5A ≤ I_{LOAD} ≤ 3A, 8V ≤ V_{IN} ≤ 40V V_{OUT} = 5V, Circuit of *Figure 2*	1.230	 1.193/**1.180** 1.267/**1.280**	V V(Min) V(Max)
V_{OUT}	Feedback Voltage LM2576HV	0.5A ≤ I_{LOAD} ≤ 3A, 8V ≤ V_{IN} ≤ 60V V_{OUT} = 5V, Circuit of *Figure 2*	1.230	 1.193/**1.180** 1.273/**1.286**	V V(Min) V(Max)
η	Efficiency	V_{IN} = 12V, I_{LOAD} = 3A, V_{OUT} = 5V	77		%

All Output Voltage Versions
Electrical Characteristics

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range. Unless otherwise specified, V_{IN} = 12V for the 3.3V, 5V, and Adjustable version, V_{IN} = 25V for the 12V version, and V_{IN} = 30V for the 15V version. I_{LOAD} = 500 mA.

Symbol	Parameter	Conditions	LM2576-XX LM2576HV-XX		Units (Limits)
			Typ	Limit (Note 2)	
DEVICE PARAMETERS					
I_b	Feedback Bias Current	V_{OUT} = 5V (Adjustable Version Only)	50	100/**500**	nA
f_O	Oscillator Frequency	(Note 11)	52	 47/**42** 58/**63**	kHz kHz (Min) kHz (Max)
V_{SAT}	Saturation Voltage	I_{OUT} = 3A (Note 4)	1.4	 1.8/**2.0**	V V(Max)
DC	Max Duty Cycle (ON)	(Note 5)	98	 93	% %(Min)
I_{CL}	Current Limit	(Notes 4, 11)	5.8	 4.2/**3.5** 6.9/**7.5**	A A(Min) A(Max)
I_L	Output Leakage Current	(Notes 6, 7): Output = 0V Output = −1V Output = −1V	 7.5 	2 30	mA(Max) mA mA(Max)
I_Q	Quiescent Current	(Note 6)	5	 10	mA mA(Max)
I_{STBY}	Standby Quiescent Current	$\overline{ON}$ /OFF Pin = 5V (OFF)	50	 200	µA µA(Max)
θ_{JA} θ_{JA} θ_{JC} θ_{JA}	Thermal Resistance	T Package, Junction to Ambient (Note 8) T Package, Junction to Ambient (Note 9) T Package, Junction to Case S Package, Junction to Ambient (Note 10)	65 45 2 50		˚C/W

All Output Voltage Versions
Electrical Characteristics (Continued)

Specifications with standard type face are for T_J = 25˚C, and those with **boldface type** apply over full Operating Temperature Range. Unless otherwise specified, V_{IN} = 12V for the 3.3V, 5V, and Adjustable version, V_{IN} = 25V for the 12V version, and V_{IN} = 30V for the 15V version. I_{LOAD} = 500 mA.

Symbol	Parameter	Conditions	LM2576-XX LM2576HV-XX		Units (Limits)
			Typ	Limit (Note 2)	
$\overline{ON}$ /OFF CONTROL Test Circuit *Figure 2*					
V_{IH}	$\overline{ON}$ /OFF Pin	V_{OUT} = 0V	1.4	2.2/**2.4**	V(Min)
V_{IL}	Logic Input Level	V_{OUT} = Nominal Output Voltage	1.2	1.0/**0.8**	V(Max)
I_{IH}	$\overline{ON}$ /OFF Pin Input Current	$\overline{ON}$ /OFF Pin = 5V (OFF)	12	30	µA µA(Max)
I_{IL}		$\overline{ON}$ /OFF Pin = 0V (ON)	0	10	µA µA(Max)

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Operating Ratings indicate conditions for which the device is intended to be functional, but do not guarantee specific performance limits. For guaranteed specifications and test conditions, see the Electrical Characteristics.

Note 2: All limits guaranteed at room temperature (standard type face) and at temperature extremes (bold type face). All room temperature limits are 100% production tested. All limits at temperature extremes are guaranteed via correlation using standard Statistical Quality Control (SQC) methods.

Note 3: External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in the *Figure 2* test circuit, system performance will be as shown in system parameters section of Electrical Characteristics.

Note 4: Output pin sourcing current. No diode, inductor or capacitor connected to output.

Note 5: Feedback pin removed from output and connected to 0V.

Note 6: Feedback pin removed from output and connected to +12V for the Adjustable, 3.3V, and 5V versions, and +25V for the 12V and 15V versions, to force the output transistor OFF.

Note 7: V_{IN} = 40V (60V for high voltage version).

Note 8: Junction to ambient thermal resistance (no external heat sink) for the 5 lead TO-220 package mounted vertically, with ½ inch leads in a socket, or on a PC board with minimum copper area.

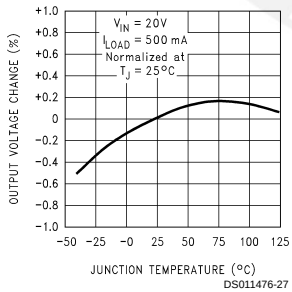
Note 9: Junction to ambient thermal resistance (no external heat sink) for the 5 lead TO-220 package mounted vertically, with ¼ inch leads soldered to a PC board containing approximately 4 square inches of copper area surrounding the leads.

Note 10: If the TO-263 package is used, the thermal resistance can be reduced by increasing the PC board copper area thermally connected to the package. Using 0.5 square inches of copper area, θ_{JA} is 50˚C/W, with 1 square inch of copper area, θ_{JA} is 37˚C/W, and with 1.6 or more square inches of copper area, θ_{JA} is 32˚C/W.

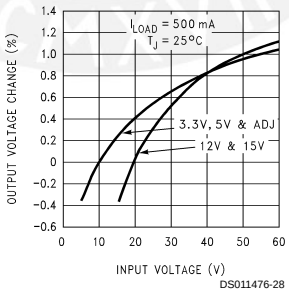
Note 11: The oscillator frequency reduces to approximately 11 kHz in the event of an output short or an overload which causes the regulated output voltage to drop approximately 40% from the nominal output voltage. This self protection feature lowers the average power dissipation of the IC by lowering the minimum duty cycle from 5% down to approximately 2%.

Typical Performance Characteristics (Circuit of *Figure 2*)

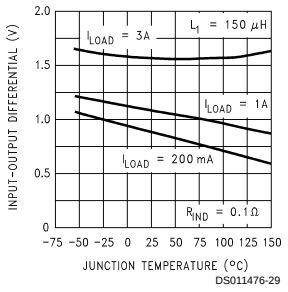
Normalized Output Voltage



Line Regulation

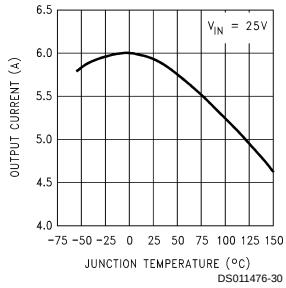


Dropout Voltage



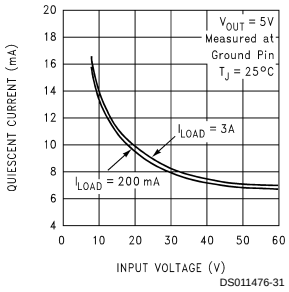
Typical Performance Characteristics (Circuit of *Figure 2*) (Continued)

Current Limit



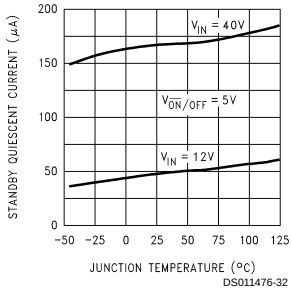
DS011476-30

Quiescent Current



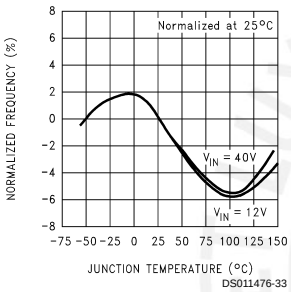
DS011476-31

Standby Quiescent Current



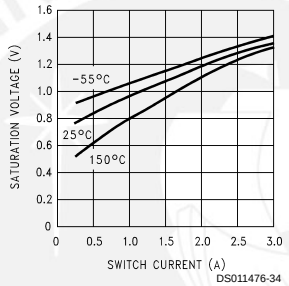
DS011476-32

Oscillator Frequency



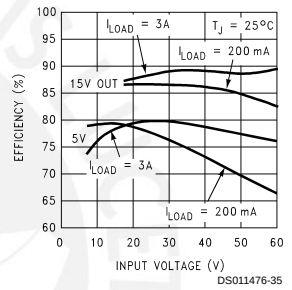
DS011476-33

Switch Saturation Voltage



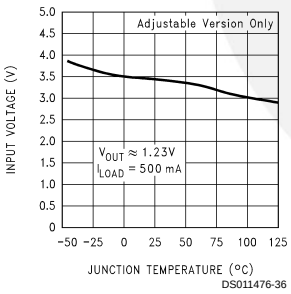
DS011476-34

Efficiency



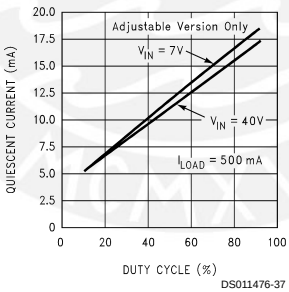
DS011476-35

Minimum Operating Voltage



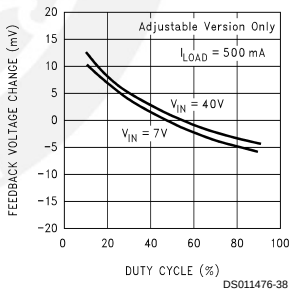
DS011476-36

Quiescent Current vs Duty Cycle



DS011476-37

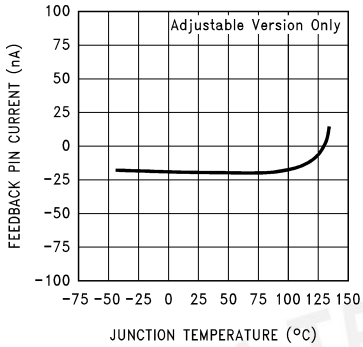
Feedback Voltage vs Duty Cycle



DS011476-38

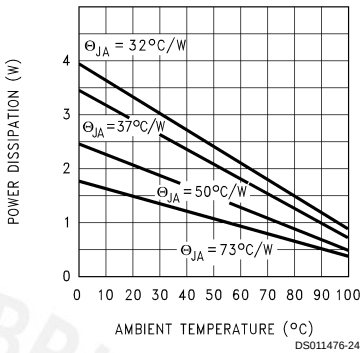
Typical Performance Characteristics (Circuit of *Figure 2*) (Continued)

Feedback Pin Current



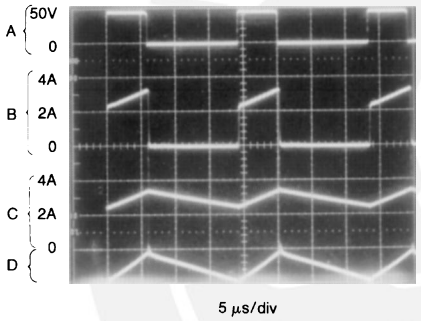
DS011476-4

Maximum Power Dissipation (TO-263) (See Note 10)



DS011476-24

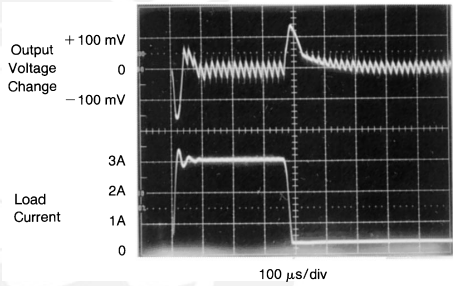
Switching Waveforms



DS011476-6

V_{OUT} = 15V
A: Output Pin Voltage, 50V/div
B: Output Pin Current, 2A/div
C: Inductor Current, 2A/div
D: Output Ripple Voltage, 50 mV/div, AC-Coupled
Horizontal Time Base: 5 µs/div

Load Transient Response



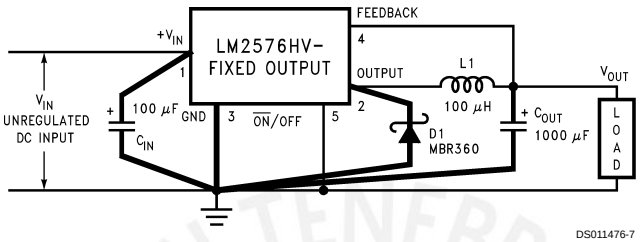
DS011476-5

Test Circuit and Layout Guidelines

As in any switching regulator, layout is very important. Rapidly switching currents associated with wiring inductance generate voltage transients which can cause problems. For minimal inductance and ground loops, the length of the leads indicated by heavy lines should be kept as short as possible.

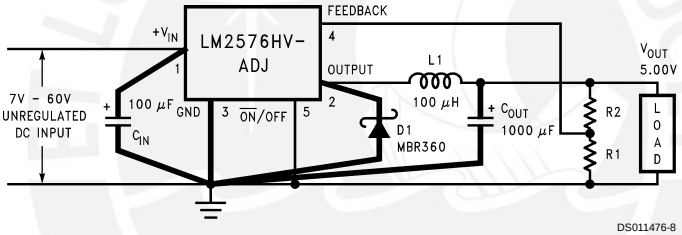
Single-point grounding (as indicated) or ground plane construction should be used for best results. When using the Adjustable version, physically locate the programming resistors near the regulator, to keep the sensitive feedback wiring short.

Fixed Output Voltage Versions



C_{IN} — 100 µF, 75V, Aluminum Electrolytic
C_{OUT} — 1000 µF, 25V, Aluminum Electrolytic
D_1 — Schottky, MBR360
L_1 — 100 µH, Pulse Eng. PE-92108
R_1 — 2k, 0.1%
R_2 — 6.12k, 0.1%

Adjustable Output Voltage Version



$$V_{OUT} = V_{REF}\left(1 + \frac{R_2}{R_1}\right)$$

$$R_2 = R_1\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

where V_{REF} = 1.23V, R1 between 1k and 5k.

FIGURE 2.

LM2576 Series Buck Regulator Design Procedure

PROCEDURE (Fixed Output Voltage Versions)	EXAMPLE (Fixed Output Voltage Versions)
Given: V_{OUT} = Regulated Output Voltage (3.3V, 5V, 12V, or 15V) V_{IN}(Max) = Maximum Input Voltage I_{LOAD}(Max) = Maximum Load Current	**Given:** V_{OUT} = 5V V_{IN}(Max) = 15V I_{LOAD}(Max) = 3A
1. Inductor Selection (L1) **A.** Select the correct Inductor value selection guide from *Figures 3, 4, 5* or *Figure 6*. (Output voltages of 3.3V, 5V, 12V or 15V respectively). For other output voltages, see the design procedure for the adjustable version. **B.** From the inductor value selection guide, identify the inductance region intersected by V_{IN}(Max) and I_{LOAD}(Max), and note the inductor code for that region. **C.** Identify the inductor value from the inductor code, and select an appropriate inductor from the table shown in *Figure 3*. Part numbers are listed for three inductor manufacturers. The inductor chosen must be rated for operation at the LM2576 switching frequency (52 kHz) and for a current rating of 1.15 x I_{LOAD}. For additional inductor information, see the inductor section in the Application Hints section of this data sheet.	**1. Inductor Selection (L1)** **A.** Use the selection guide shown in *Figure 4*. **B.** From the selection guide, the inductance area intersected by the 15V line and 3A line is L100. **C.** Inductor value required is 100 µH. From the table in *Figure 3*. Choose AIE 415-0930, Pulse Engineering PE92108, or Renco RL2444.
2. Output Capacitor Selection (C_{OUT}) **A.** The value of the output capacitor together with the inductor defines the dominate pole-pair of the switching regulator loop. For stable operation and an acceptable output ripple voltage, (approximately 1% of the output voltage) a value between 100 µF and 470 µF is recommended. **B.** The capacitor's voltage rating should be at least 1.5 times greater than the output voltage. For a 5V regulator, a rating of at least 8V is appropriate, and a 10V or 15V rating is recommended. Higher voltage electrolytic capacitors generally have lower ESR numbers, and for this reason it may be necessary to select a capacitor rated for a higher voltage than would normally be needed.	**2. Output Capacitor Selection (C_{OUT})** **A.** C_{OUT} = 680 µF to 2000 µF standard aluminum electrolytic. **B.**Capacitor voltage rating = 20V.
3. Catch Diode Selection (D1) **A.**The catch-diode current rating must be at least 1.2 times greater than the maximum load current. Also, if the power supply design must withstand a continuous output short, the diode should have a current rating equal to the maximum current limit of the LM2576. The most stressful condition for this diode is an overload or shorted output condition. **B.** The reverse voltage rating of the diode should be at least 1.25 times the maximum input voltage.	**3. Catch Diode Selection (D1)** **A.**For this example, a 3A current rating is adequate. **B.** Use a 20V 1N5823 or SR302 Schottky diode, or any of the suggested fast-recovery diodes shown in *Figure 8*.
4. Input Capacitor (C_{IN}) An aluminum or tantalum electrolytic bypass capacitor located close to the regulator is needed for stable operation.	**4. Input Capacitor (C_{IN})** A 100 µF, 25V aluminum electrolytic capacitor located near the input and ground pins provides sufficient bypassing.

LM2576 Series Buck Regulator Design Procedure (Continued)

INDUCTOR VALUE SELECTION GUIDES (For Continuous Mode Operation)

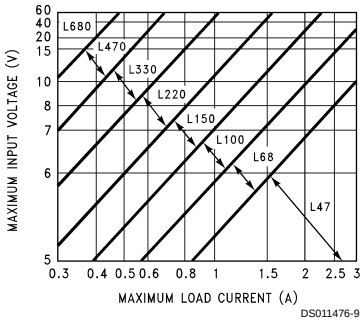


FIGURE 3. LM2576(HV)-3.3

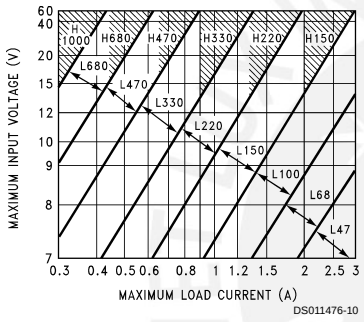


FIGURE 4. LM2576(HV)-5.0

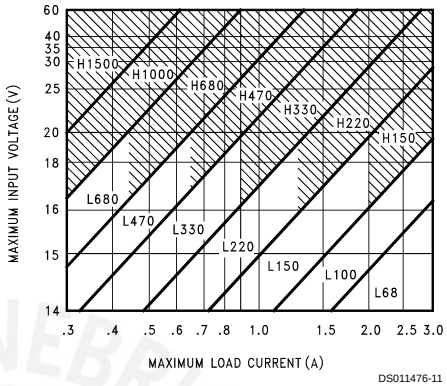


FIGURE 5. LM2576(HV)-12

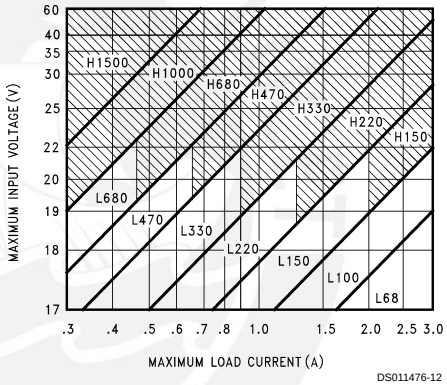


FIGURE 6. LM2576(HV)-15

LM2576 Series Buck Regulator Design Procedure (Continued)

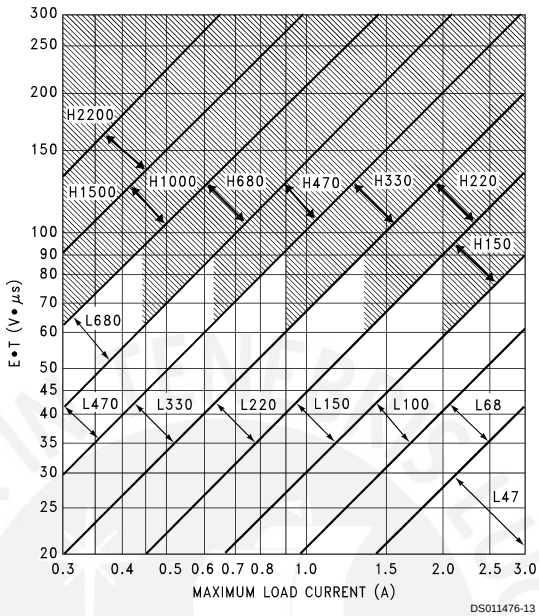


FIGURE 7. LM2576(HV)-ADJ

PROCEDURE (Adjustable Output Voltage Versions)	EXAMPLE (Adjustable Output Voltage Versions)
Given: V_{OUT} = Regulated Output Voltage V_{IN}(Max) = Maximum Input Voltage I_{LOAD}(Max) = Maximum Load Current F = Switching Frequency *(Fixed at 52 kHz)*	**Given:** V_{OUT} = 10V V_{IN}(Max) = 25V I_{LOAD}(Max) = 3A F = 52 kHz
1. Programming Output Voltage *(Selecting R1 and R2, as shown in Figure 2)* Use the following formula to select the appropriate resistor values. $V_{OUT} = V_{REF}\left(1 + \frac{R_2}{R_1}\right)$ where $V_{REF} = 1.23V$ R_1 can be between 1k and 5k. *(For best temperature coefficient and stability with time, use 1% metal film resistors)* $R_2 = R_1\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$	**1. Programming Output Voltage** *(Selecting R1 and R2)* $V_{OUT} = 1.23\left(1 + \frac{R_2}{R_1}\right)$ Select R1 = 1k $R_2 = R_1\left(\frac{V_{OUT}}{V_{REF}} - 1\right) = 1k\left(\frac{10V}{1.23V} - 1\right)$ R_2 = 1k (8.13 − 1) = 7.13k, closest 1% value is 7.15k

LM2576 Series Buck Regulator Design Procedure (Continued)

PROCEDURE (Adjustable Output Voltage Versions)	EXAMPLE (Adjustable Output Voltage Versions)
2. Inductor Selection (L1) **A.** Calculate the inductor Volt • microsecond constant, E • T (V • µs), from the following formula: $E \bullet T = (V_{IN} - V_{OUT}) \frac{V_{OUT}}{V_{IN}} \bullet \frac{1000}{F\ (in\ kHz)} (V \bullet \mu s)$ **B.** Use the E • T value from the previous formula and match it with the E • T number on the vertical axis of the **Inductor Value Selection Guide** shown in *Figure 7*. **C.** On the horizontal axis, select the maximum load current. **D.** Identify the inductance region intersected by the E • T value and the maximum load current value, and note the inductor code for that region. **E.** Identify the inductor value from the inductor code, and select an appropriate inductor from the table shown in *Figure 9*. Part numbers are listed for three inductor manufacturers. The inductor chosen must be rated for operation at the LM2576 switching frequency (52 kHz) and for a current rating of 1.15 x I_{LOAD}. For additional inductor information, see the inductor section in the application hints section of this data sheet.	**2. Inductor Selection (L1)** **A.** Calculate E • T (V • µs) $E \bullet T = (25 - 10) \bullet \frac{10}{25} \bullet \frac{1000}{52} = 115\ V \bullet \mu s$ **B.** E • T = 115 V • µs **C.** I_{LOAD}(Max) = 3A **D.** Inductance Region = H150 **E.** Inductor Value = 150 µH *Choose from* ***AIE*** *part #415-0936* ***Pulse Engineering*** *part* #PE-531115, or ***Renco part #RL2445.***
3. Output Capacitor Selection (C_{OUT}) **A.** The value of the output capacitor together with the inductor defines the dominate pole-pair of the switching regulator loop. For stable operation, the capacitor must satisfy the following requirement: $C_{OUT} \geq 13{,}300 \frac{V_{IN}(Max)}{V_{OUT} \bullet L(\mu H)} (\mu F)$ The above formula yields capacitor values between 10 µF and 2200 µF that will satisfy the loop requirements for stable operation. But to achieve an acceptable output ripple voltage, (approximately 1% of the output voltage) and transient response, the output capacitor may need to be several times larger than the above formula yields. **B.** The capacitor's voltage rating should be at last 1.5 times greater than the output voltage. For a 10V regulator, a rating of at least 15V or more is recommended. Higher voltage electrolytic capacitors generally have lower ESR numbers, and for this reason it may be necessary to select a capacitor rate for a higher voltage than would normally be needed.	**3. Output Capacitor Selection (C_{OUT})** $C_{OUT} > 13{,}300 \frac{25}{10 \bullet 150} = 22.2\ \mu F$ However, for acceptable output ripple voltage select $C_{OUT} \geq 680\ \mu F$ C_{OUT} = 680 µF electrolytic capacitor
4. Catch Diode Selection (D1) **A.** The catch-diode current rating must be at least 1.2 times greater than the maximum load current. Also, if the power supply design must withstand a continuous output short, the diode should have a current rating equal to the maximum current limit of the LM2576. The most stressful condition for this diode is an overload or shorted output. See diode selection guide in *Figure 8*. **B.** The reverse voltage rating of the diode should be at least 1.25 times the maximum input voltage.	**4. Catch Diode Selection (D1)** **A.** For this example, a 3.3A current rating is adequate. **B.** Use a 30V 31DQ03 Schottky diode, or any of the suggested fast-recovery diodes in *Figure 8*.
5. Input Capacitor (C_{IN}) An aluminum or tantalum electrolytic bypass capacitor located close to the regulator is needed for stable operation.	**5. Input Capacitor (C_{IN})** A 100 µF aluminum electrolytic capacitor located near the input and ground pins provides sufficient bypassing.

LM2576 Series Buck Regulator Design Procedure (Continued)

To further simplify the buck regulator design procedure, National Semiconductor is making available computer design software to be used with the SIMPLE SWITCHER line of switching regulators. ***Switchers Made Simple*** *(Version 3.3) is available on a (3½") diskette for IBM compatible computers from a National Semiconductor sales office in your area.*

V_R	Schottky		Fast Recovery	
	3A	4A–6A	3A	4A–6A
20V	1N5820 MBR320P SR302	1N5823	The following diodes are all rated to 100V 31DF1 HER302	The following diodes are all rated to 100V 50WF10 MUR410 HER602
30V	1N5821 MBR330 31DQ03 SR303	50WQ03 1N5824		
40V	1N5822 MBR340 31DQ04 SR304	MBR340 50WQ04 1N5825		
50V	MBR350 31DQ05 SR305	50WQ05		
60V	MBR360 DQ06 SR306	50WR06 50SQ060		

FIGURE 8. Diode Selection Guide

Inductor Code	Inductor Value	Schott (Note 12)	Pulse Eng. (Note 13)	Renco (Note 14)
L47	47 µH	671 26980	PE-53112	RL2442
L68	68 µH	671 26990	PE-92114	RL2443
L100	100 µH	671 27000	PE-92108	RL2444
L150	150 µH	671 27010	PE-53113	RL1954
L220	220 µH	671 27020	PE-52626	RL1953
L330	330 µH	671 27030	PE-52627	RL1952
L470	470 µH	671 27040	PE-53114	RL1951
L680	680 µH	671 27050	PE-52629	RL1950
H150	150 µH	671 27060	PE-53115	RL2445
H220	220 µH	671 27070	PE-53116	RL2446
H330	330 µH	671 27080	PE-53117	RL2447
H470	470 µH	671 27090	PE-53118	RL1961
H680	680 µH	671 27100	PE-53119	RL1960
H1000	1000 µH	671 27110	PE-53120	RL1959
H1500	1500 µH	671 27120	PE-53121	RL1958
H2200	2200 µH	671 27130	PE-53122	RL2448

Note 12: Schott Corporation, (612) 475-1173, 1000 Parkers Lake Road, Wayzata, MN 55391.

Note 13: Pulse Engineering, (619) 674-8100, P.O. Box 12235, San Diego, CA 92112.

Note 14: Renco Electronics Incorporated, (516) 586-5566, 60 Jeffryn Blvd. East, Deer Park, NY 11729.

FIGURE 9. Inductor Selection by Manufacturer's Part Number

Application Hints

INPUT CAPACITOR (C_{IN})

To maintain stability, the regulator input pin must be bypassed with at least a 100 µF electrolytic capacitor. The capacitor's leads must be kept short, and located near the regulator.

If the operating temperature range includes temperatures below −25˚C, the input capacitor value may need to be larger. With most electrolytic capacitors, the capacitance value decreases and the ESR increases with lower temperatures and age. Paralleling a ceramic or solid tantalum capacitor will increase the regulator stability at cold temperatures. For maximum capacitor operating lifetime, the capacitor's RMS ripple current rating should be greater than

$$1.2 \times \left(\frac{t_{ON}}{T}\right) \times I_{LOAD}$$

where $\frac{t_{ON}}{T} = \frac{V_{OUT}}{V_{IN}}$ for a buck regulator

and $\frac{t_{ON}}{T} = \frac{|V_{OUT}|}{|V_{OUT}| + V_{IN}}$ for a buck-boost regulator.

INDUCTOR SELECTION

All switching regulators have two basic modes of operation: continuous and discontinuous. The difference between the two types relates to the inductor current, whether it is flowing continuously, or if it drops to zero for a period of time in the normal switching cycle. Each mode has distinctively different operating characteristics, which can affect the regulator performance and requirements.

The LM2576 (or any of the SIMPLE SWITCHER family) can be used for both continuous and discontinuous modes of operation.

The inductor value selection guides in *Figure 3* through *Figure 7* were designed for buck regulator designs of the continuous inductor current type. When using inductor values shown in the inductor selection guide, the peak-to-peak inductor ripple current will be approximately 20% to 30% of the maximum DC current. With relatively heavy load currents, the circuit operates in the continuous mode (inductor current always flowing), but under light load conditions, the circuit will be forced to the discontinuous mode (inductor current falls to zero for a period of time). This discontinuous mode of operation is perfectly acceptable. For light loads (less than approximately 300 mA) it may be desirable to operate the regulator in the discontinuous mode, primarily because of the lower inductor values required for the discontinuous mode.

The selection guide chooses inductor values suitable for continuous mode operation, but if the inductor value chosen is prohibitively high, the designer should investigate the possibility of discontinuous operation. The computer design software ***Switchers Made Simple*** will provide all component values for discontinuous (as well as continuous) mode of operation.

Inductors are available in different styles such as pot core, toriod, E-frame, bobbin core, etc., as well as different core materials, such as ferrites and powdered iron. The least expensive, the bobbin core type, consists of wire wrapped on a ferrite rod core. This type of construction makes for an inexpensive inductor, but since the magnetic flux is not completely contained within the core, it generates more electromagnetic interference (EMI). This EMI can cause problems in sensitive circuits, or can give incorrect scope readings because of induced voltages in the scope probe.

The inductors listed in the selection chart include ferrite pot core construction for AIE, powdered iron toroid for Pulse Engineering, and ferrite bobbin core for Renco.

An inductor should not be operated beyond its maximum rated current because it may saturate. When an inductor begins to saturate, the inductance decreases rapidly and the inductor begins to look mainly resistive (the DC resistance of the winding). This will cause the switch current to rise very rapidly. Different inductor types have different saturation characteristics, and this should be kept in mind when selecting an inductor.

The inductor manufacturer's data sheets include current and energy limits to avoid inductor saturation.

INDUCTOR RIPPLE CURRENT

When the switcher is operating in the continuous mode, the inductor current waveform ranges from a triangular to a sawtooth type of waveform (depending on the input voltage). For a given input voltage and output voltage, the peak-to-peak amplitude of this inductor current waveform remains constant. As the load current rises or falls, the entire sawtooth current waveform also rises or falls. The average DC value of this waveform is equal to the DC load current (in the buck regulator configuration).

If the load current drops to a low enough level, the bottom of the sawtooth current waveform will reach zero, and the switcher will change to a discontinuous mode of operation. This is a perfectly acceptable mode of operation. Any buck switching regulator (no matter how large the inductor value is) will be forced to run discontinuous if the load current is light enough.

OUTPUT CAPACITOR

An output capacitor is required to filter the output voltage and is needed for loop stability. The capacitor should be located near the LM2576 using short pc board traces. Standard aluminum electrolytics are usually adequate, but low ESR types are recommended for low output ripple voltage and good stability. The ESR of a capacitor depends on many factors, some which are: the value, the voltage rating, physical size and the type of construction. In general, low value or low voltage (less than 12V) electrolytic capacitors usually have higher ESR numbers.

The amount of output ripple voltage is primarily a function of the ESR (Equivalent Series Resistance) of the output capacitor and the amplitude of the inductor ripple current (ΔI_{IND}). See the section on inductor ripple current in Application Hints.

The lower capacitor values (220 µF–1000 µF) will allow typically 50 mV to 150 mV of output ripple voltage, while larger-value capacitors will reduce the ripple to approximately 20 mV to 50 mV.

Output Ripple Voltage = (ΔI_{IND}) (ESR of C_{OUT})

To further reduce the output ripple voltage, several standard electrolytic capacitors may be paralleled, or a higher-grade capacitor may be used. Such capacitors are often called "high-frequency," "low-inductance," or "low-ESR." These will reduce the output ripple to 10 mV or 20 mV. However, when operating in the continuous mode, reducing the ESR below 0.03Ω can cause instability in the regulator.

Application Hints (Continued)

Tantalum capacitors can have a very low ESR, and should be carefully evaluated if it is the only output capacitor. Because of their good low temperature characteristics, a tantalum can be used in parallel with aluminum electrolytics, with the tantalum making up 10% or 20% of the total capacitance.

The capacitor's ripple current rating at 52 kHz should be at least 50% higher than the peak-to-peak inductor ripple current.

CATCH DIODE

Buck regulators require a diode to provide a return path for the inductor current when the switch is off. This diode should be located close to the LM2576 using short leads and short printed circuit traces.

Because of their fast switching speed and low forward voltage drop, Schottky diodes provide the best efficiency, especially in low output voltage switching regulators (less than 5V). Fast-Recovery, High-Efficiency, or Ultra-Fast Recovery diodes are also suitable, but some types with an abrupt turn-off characteristic may cause instability and EMI problems. A fast-recovery diode with soft recovery characteristics is a better choice. Standard 60 Hz diodes (e.g., 1N4001 or 1N5400, etc.) are also **not suitable.** See *Figure 8* for Schottky and "soft" fast-recovery diode selection guide.

OUTPUT VOLTAGE RIPPLE AND TRANSIENTS

The output voltage of a switching power supply will contain a sawtooth ripple voltage at the switcher frequency, typically about 1% of the output voltage, and may also contain short voltage spikes at the peaks of the sawtooth waveform.

The output ripple voltage is due mainly to the inductor sawtooth ripple current multiplied by the ESR of the output capacitor. (See the inductor selection in the application hints.)

The voltage spikes are present because of the the fast switching action of the output switch, and the parasitic inductance of the output filter capacitor. To minimize these voltage spikes, special low inductance capacitors can be used, and their lead lengths must be kept short. Wiring inductance, stray capacitance, as well as the scope probe used to evaluate these transients, all contribute to the amplitude of these spikes.

An additional small LC filter (20 µH & 100 µF) can be added to the output (as shown in *Figure 15*) to further reduce the amount of output ripple and transients. A 10 x reduction in output ripple voltage and transients is possible with this filter.

FEEDBACK CONNECTION

The LM2576 (fixed voltage versions) feedback pin must be wired to the output voltage point of the switching power supply. When using the adjustable version, physically locate both output voltage programming resistors near the LM2576 to avoid picking up unwanted noise. Avoid using resistors greater than 100 kΩ because of the increased chance of noise pickup.

$\overline{ON}$ /OFF INPUT

For normal operation, the $\overline{ON}$ /OFF pin should be grounded or driven with a low-level TTL voltage (typically below 1.6V). To put the regulator into standby mode, drive this pin with a high-level TTL or CMOS signal. The $\overline{ON}$ /OFF pin can be safely pulled up to $+V_{IN}$ without a resistor in series with it. The $\overline{ON}$ /OFF pin should not be left open.

GROUNDING

To maintain output voltage stability, the power ground connections must be low-impedance (see *Figure 2*). For the 5-lead TO-220 and TO-263 style package, both the tab and pin 3 are ground and either connection may be used, as they are both part of the same copper lead frame.

HEAT SINK/THERMAL CONSIDERATIONS

In many cases, only a small heat sink is required to keep the LM2576 junction temperature within the allowed operating range. For each application, to determine whether or not a heat sink will be required, the following must be identified:

1. Maximum ambient temperature (in the application).
2. Maximum regulator power dissipation (in application).
3. Maximum allowed junction temperature (125˚C for the LM2576). For a safe, conservative design, a temperature approximately 15˚C cooler than the maximum temperatures should be selected.
4. LM2576 package thermal resistances θ_{JA} and θ_{JC}.

Total power dissipated by the LM2576 can be estimated as follows:

$$P_D = (V_{IN})(I_Q) + (V_O/V_{IN})(I_{LOAD})(V_{SAT})$$

where I_Q (quiescent current) and V_{SAT} can be found in the Characteristic Curves shown previously, V_{IN} is the applied minimum input voltage, V_O is the regulated output voltage, and I_{LOAD} is the load current. The dynamic losses during turn-on and turn-off are negligible if a Schottky type catch diode is used.

When no heat sink is used, the junction temperature rise can be determined by the following:

$$\Delta T_J = (P_D)\ (\theta_{JA})$$

To arrive at the actual operating junction temperature, add the junction temperature rise to the maximum ambient temperature.

$$T_J = \Delta T_J + T_A$$

If the actual operating junction temperature is greater than the selected safe operating junction temperature determined in step 3, then a heat sink is required.

When using a heat sink, the junction temperature rise can be determined by the following:

$$\Delta T_J = (P_D)\ (\theta_{JC} + \theta_{interface} + \theta_{Heat\ sink})$$

The operating junction temperature will be:

$$T_J = T_A + \Delta T_J$$

As above, if the actual operating junction temperature is greater than the selected safe operating junction temperature, then a larger heat sink is required (one that has a lower thermal resistance).

Included on the **Switcher Made Simple** design software is a more precise (non-linear) thermal model that can be used to determine junction temperature with different input-output parameters or different component values. It can also calculate the heat sink thermal resistance required to maintain the regulators junction temperature below the maximum operating temperature.

Additional Applications

INVERTING REGULATOR

Figure 10 shows a LM2576-12 in a buck-boost configuration to generate a negative 12V output from a positive input voltage. This circuit bootstraps the regulator's ground pin to the

Additional Applications (Continued)

negative output voltage, then by grounding the feedback pin, the regulator senses the inverted output voltage and regulates it to −12V.

For an input voltage of 12V or more, the maximum available output current in this configuration is approximately 700 mA. At lighter loads, the minimum input voltage required drops to approximately 4.7V.

The switch currents in this buck-boost configuration are higher than in the standard buck-mode design, thus lowering the available output current. Also, the start-up input current of the buck-boost converter is higher than the standard buck-mode regulator, and this may overload an input power source with a current limit less than 5A. Using a delayed turn-on or an undervoltage lockout circuit (described in the next section) would allow the input voltage to rise to a high enough level before the switcher would be allowed to turn on.

Because of the structural differences between the buck and the buck-boost regulator topologies, the buck regulator design procedure section can not be used to to select the inductor or the output capacitor. The recommended range of inductor values for the buck-boost design is between 68 µH and 220 µH, and the output capacitor values must be larger than what is normally required for buck designs. Low input voltages or high output currents require a large value output capacitor (in the thousands of micro Farads).

The peak inductor current, which is the same as the peak switch current, can be calculated from the following formula:

$$I_p \approx \frac{I_{LOAD}\,(V_{IN} + |V_O|)}{V_{IN}} + \frac{V_{IN}\,|V_O|}{V_{IN} + |V_O|} \times \frac{1}{2L_1\,f_{osc}}$$

Where f_{osc} = 52 kHz. Under normal continuous inductor current operating conditions, the minimum V_{IN} represents the worst case. Select an inductor that is rated for the peak current anticipated.

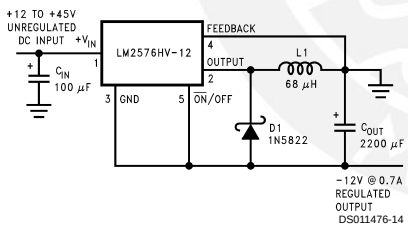


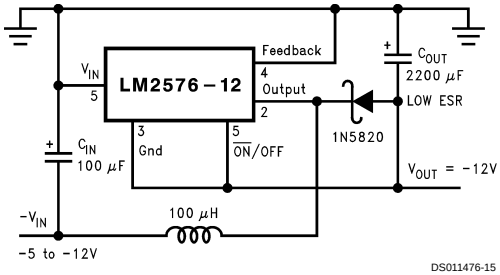
FIGURE 10. Inverting Buck-Boost Develops −12V

Also, the maximum voltage appearing across the regulator is the absolute sum of the input and output voltage. For a −12V output, the maximum input voltage for the LM2576 is +28V, or +48V for the LM2576HV.

The ***Switchers Made Simple*** (version 3.0) design software can be used to determine the feasibility of regulator designs using different topologies, different input-output parameters, different components, etc.

NEGATIVE BOOST REGULATOR

Another variation on the buck-boost topology is the negative boost configuration. The circuit in *Figure 11* accepts an input voltage ranging from −5V to −12V and provides a regulated −12V output. Input voltages greater than −12V will cause the output to rise above −12V, but will not damage the regulator.



Typical Load Current
400 mA for V_{IN} = −5.2V
750 mA for V_{IN} = −7V
Note: Heat sink may be required.

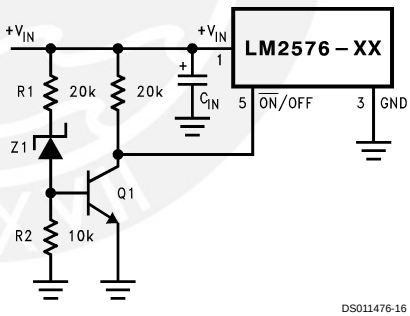
FIGURE 11. Negative Boost

Because of the boosting function of this type of regulator, the switch current is relatively high, especially at low input voltages. Output load current limitations are a result of the maximum current rating of the switch. Also, boost regulators can not provide current limiting load protection in the event of a shorted load, so some other means (such as a fuse) may be necessary.

UNDERVOLTAGE LOCKOUT

In some applications it is desirable to keep the regulator off until the input voltage reaches a certain threshold. An undervoltage lockout circuit which accomplishes this task is shown in *Figure 12*, while *Figure 13* shows the same circuit applied to a buck-boost configuration. These circuits keep the regulator off until the input voltage reaches a predetermined level.

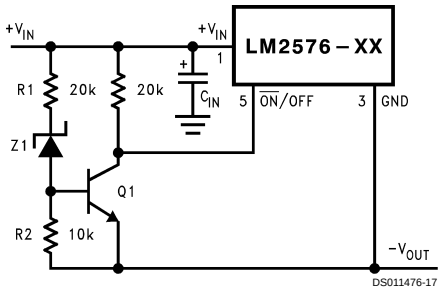
$V_{TH} \approx V_{Z1} + 2V_{BE}(Q1)$



Note: Complete circuit not shown.

FIGURE 12. Undervoltage Lockout for Buck Circuit

Additional Applications (Continued)



Note: Complete circuit not shown (see *Figure 10*).

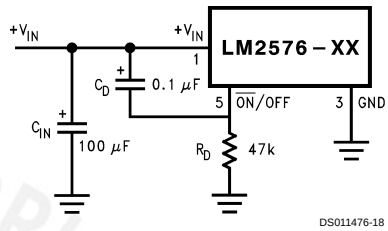
FIGURE 13. Undervoltage Lockout for Buck-Boost Circuit

DELAYED STARTUP

The $\overline{\text{ON}}$ /OFF pin can be used to provide a delayed startup feature as shown in *Figure 14*. With an input voltage of 20V and for the part values shown, the circuit provides approximately 10 ms of delay time before the circuit begins switching. Increasing the RC time constant can provide longer delay times. But excessively large RC time constants can cause problems with input voltages that are high in 60 Hz or 120 Hz ripple, by coupling the ripple into the $\overline{\text{ON}}$ /OFF pin.

ADJUSTABLE OUTPUT, LOW-RIPPLE POWER SUPPLY

A 3A power supply that features an adjustable output voltage is shown in *Figure 15*. An additional L-C filter that reduces the output ripple by a factor of 10 or more is included in this circuit.



Note: Complete circuit not shown.

FIGURE 14. Delayed Startup

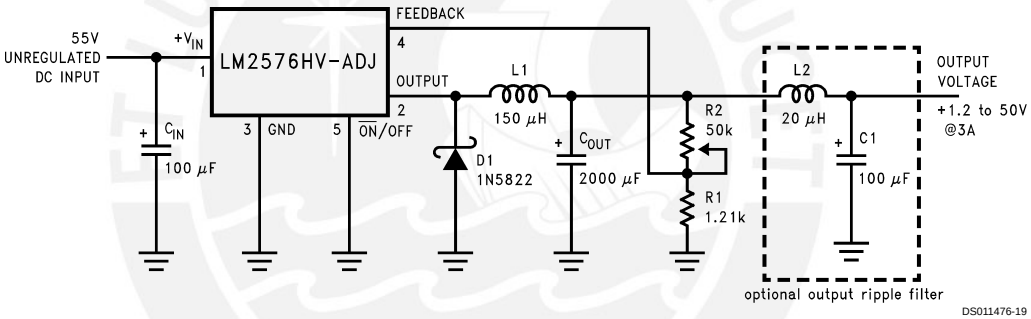


FIGURE 15. 1.2V to 55V Adjustable 3A Power Supply with Low Output Ripple

Definition of Terms

BUCK REGULATOR

A switching regulator topology in which a higher voltage is converted to a lower voltage. Also known as a step-down switching regulator.

BUCK-BOOST REGULATOR

A switching regulator topology in which a positive voltage is converted to a negative voltage without a transformer.

DUTY CYCLE (D)

Ratio of the output switch's on-time to the oscillator period.

for buck regulator

$$D = \frac{t_{ON}}{T} = \frac{V_{OUT}}{V_{IN}}$$

for buck-boost regulator

$$D = \frac{t_{ON}}{T} = \frac{|V_O|}{|V_O| + V_{IN}}$$

CATCH DIODE OR CURRENT STEERING DIODE

The diode which provides a return path for the load current when the LM2576 switch is OFF.

EFFICIENCY (η)

The proportion of input power actually delivered to the load.

$$\eta = \frac{P_{OUT}}{P_{IN}} = \frac{P_{OUT}}{P_{OUT} + P_{LOSS}}$$

CAPACITOR EQUIVALENT SERIES RESISTANCE (ESR)

The purely resistive component of a real capacitor's impedance (see *Figure 16*). It causes power loss resulting in capacitor heating, which directly affects the capacitor's operating lifetime. When used as a switching regulator output filter, higher ESR values result in higher output ripple voltages.

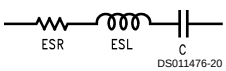


FIGURE 16. Simple Model of a Real Capacitor

Definition of Terms (Continued)

Most standard aluminum electrolytic capacitors in the 100 µF–1000 µF range have 0.5Ω to 0.1Ω ESR. Higher-grade capacitors ("low-ESR", "high-frequency", or "low-inductance") in the 100 µF–1000 µF range generally have ESR of less than 0.15Ω.

EQUIVALENT SERIES INDUCTANCE (ESL)

The pure inductance component of a capacitor (see *Figure 16*). The amount of inductance is determined to a large extent on the capacitor's construction. In a buck regulator, this unwanted inductance causes voltage spikes to appear on the output.

OUTPUT RIPPLE VOLTAGE

The AC component of the switching regulator's output voltage. It is usually dominated by the output capacitor's ESR multiplied by the inductor's ripple current (ΔI_{IND}). The peak-to-peak value of this sawtooth ripple current can be determined by reading the Inductor Ripple Current section of the Application hints.

CAPACITOR RIPPLE CURRENT

RMS value of the maximum allowable alternating current at which a capacitor can be operated continuously at a specified temperature.

STANDBY QUIESCENT CURRENT (I_{STBY})

Supply current required by the LM2576 when in the standby mode ($\overline{ON}$ /OFF pin is driven to TTL-high voltage, thus turning the output switch OFF).

INDUCTOR RIPPLE CURRENT (ΔI_{IND})

The peak-to-peak value of the inductor current waveform, typically a sawtooth waveform when the regulator is operating in the continuous mode (vs. discontinuous mode).

CONTINUOUS/DISCONTINUOUS MODE OPERATION

Relates to the inductor current. In the continuous mode, the inductor current is always flowing and never drops to zero, vs. the discontinuous mode, where the inductor current drops to zero for a period of time in the normal switching cycle.

INDUCTOR SATURATION

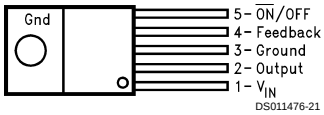
The condition which exists when an inductor cannot hold any more magnetic flux. When an inductor saturates, the inductor appears less inductive and the resistive component dominates. Inductor current is then limited only by the DC resistance of the wire and the available source current.

OPERATING VOLT MICROSECOND CONSTANT ($E \cdot T_{op}$)

The product (in Volt•µs) of the voltage applied to the inductor and the time the voltage is applied. This $E \cdot T_{op}$ constant is a measure of the energy handling capability of an inductor and is dependent upon the type of core, the core area, the number of turns, and the duty cycle.

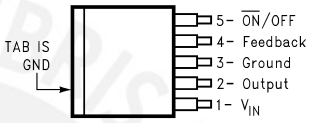
Connection Diagrams (Note 15)

Straight Leads
5-Lead TO-220 (T)
Top View

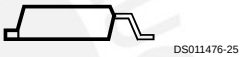


LM2576T-XX or LM2576HVT-XX
NS Package Number T05A

TO-263 (S)
5-Lead Surface-Mount Package
Top View

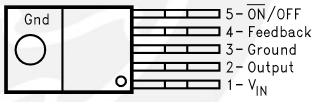


Side View

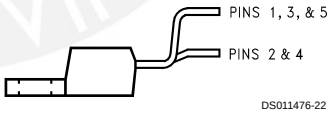


LM2576S-XX or LM2576HVS-XX
NS Package Number TS5B
LM2576SX-XX or LM2576HVSX-XX
NS Package Number TS5B, Tape and Reel

Bent, Staggered Leads
5-Lead TO-220 (T)
Top View



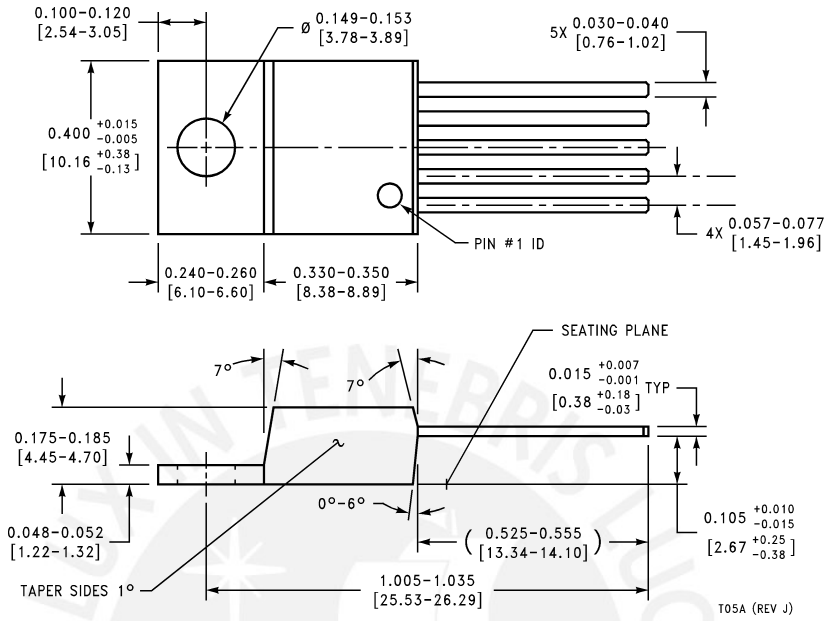
Side View



LM2576T-XX Flow LB03
or LM2576HVT-XX Flow LB03
NS Package Number T05D

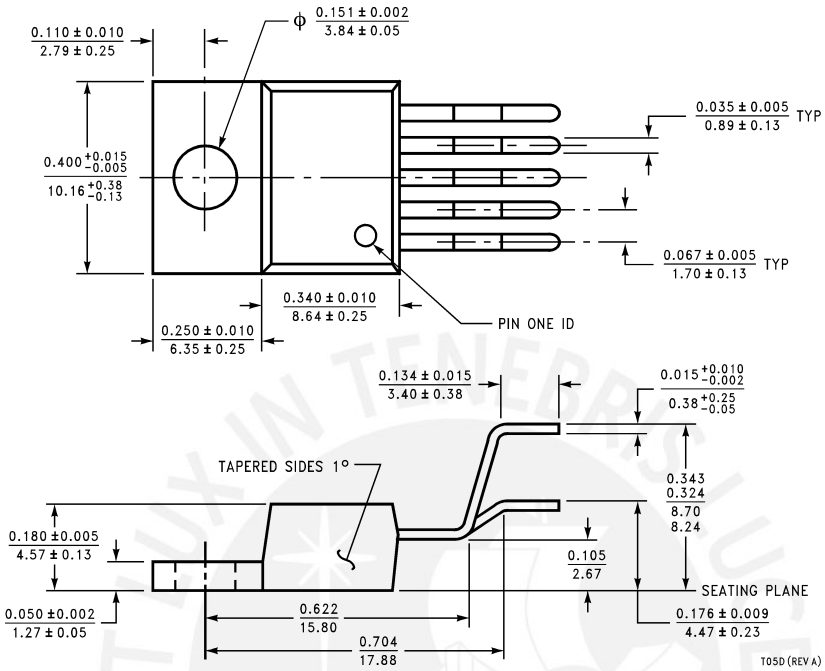
Note 15: (XX indicates output voltage option. See ordering information table for complete part number.)

Physical Dimensions inches (millimeters) unless otherwise noted



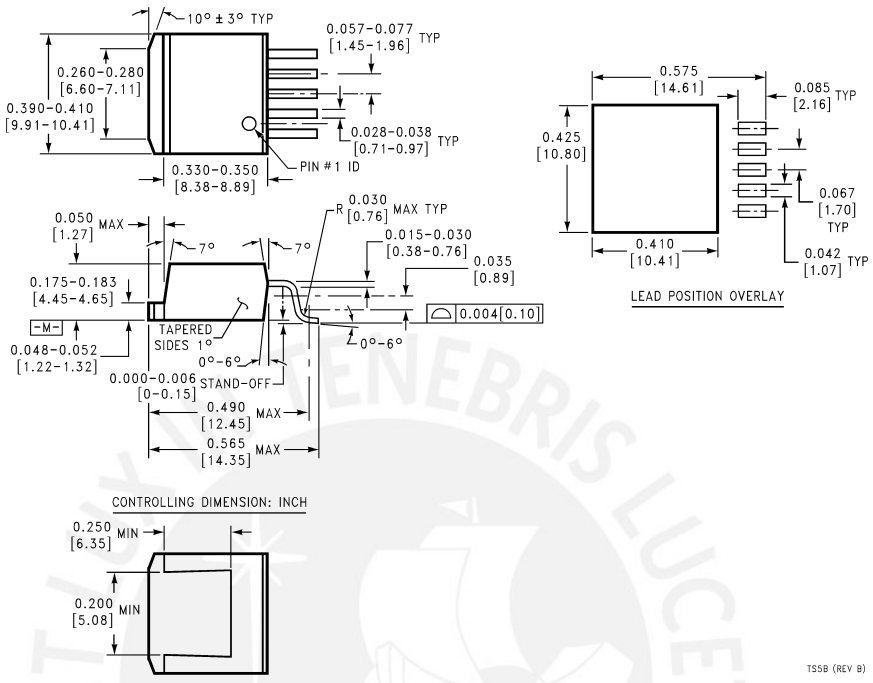
5-Lead TO-220 (T)
Order Number LM2576T-3.3, LM2576HVT-3.3, LM2576T-5.0, LM2576HVT-5.0, LM2576T-12, LM2576HVT-12, LM2576T-15, LM2576HVT-15, LM2576T-ADJ or LM2576HVT-ADJ
NS Package Number T05A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



Bent, Staggered 5-Lead TO-220 (T)
Order Number LM2576T-3.3 Flow LB03, LM2576T-XX Flow LB03, LM2576HVT-3.3 Flow LB03, LM2576T-5.0 Flow LB03, LM2576HVT-5.0 Flow LB03, LM2576T-12 Flow LB03, LM2576HVT-12 Flow LB03, LM2576T-15 Flow LB03, LM2576HVT-15 Flow LB03, LM2576T-ADJ Flow LB03 or LM2576HVT-ADJ Flow LB03
NS Package Number T05D

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



5-Lead TO-263 (S)
Order Number LM2576S-3.3, LM2576S-5.0, LM2576S-12,LM2576S-15, LM2576S-ADJ, LM2576HVS-3.3, LM2576HVS-5.0, LM2576HVS-12, LM2576HVS-15, or LM2576HVS-ADJ
NS Package Number TS5B
5-Lead TO-263 in Tape & Reel (SX)
Order Number LM2576SX-3.3, LM2576SX-5.0, LM2576SX-12, LM2576SX-15, LM2576SX-ADJ, LM2576HVSX-3.3, LM2576HVSX-5.0, LM2576HVSX-12, LM2576HVSX-15, or LM2576HVSX-ADJ
NS Package Number TS5B

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 1 80-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 1 80-530 85 85
English Tel: +49 (0) 1 80-532 78 32
Français Tel: +49 (0) 1 80-532 93 58
Italiano Tel: +49 (0) 1 80-534 16 80

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507