ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

TESIS DE GRADO

“Control Remoto vía Internet de un proceso industrial”

Previo la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA INDUSTRIAL Y AUTOMATIZACIÓN.

Presentada por:

Wendy Vanessa Abad Rodríguez

Christian Oliver Rodríguez Vera

GUAYAQUIL- ECUADOR

Año : 2006

DEDICATORIA

A nuestros padres

A nuestros hermanos y familiares

A nuestros profesores

A nuestros compañeros

AGRADECIMIENTO

Agradecemos a Dios

Agradecemos a nuestros profesores,

por sus enseñanzas diarias en las aulas.

Agradecemos a nuestros padres y familiares,

Por toda la comprensión y apoyo constante

TRIBUNAL DE GRADUACIÓN

Ing. Gustavo Bermùdez
DECANO DE LA
FIEC

Ing. Hugo Villavicencio
DIRECTOR DE TESIS

Ing. Holger Cevallos
VOCAL PRINCIPAL

Ing. Alberto Larco
VOCAL PRINCIPAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponden exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"

(Reglamento de Graduación de la ESPOL)

Wendy Abad Rodríguez

Christian Rodríguez Vera

RESUMEN

En Ecuador, la industria de las aves sobresale, gracias a la gran demanda que esta carne tiene en todos los estratos de la población, nuestro proyecto es orientado a la incubación de huevos de codornices, porque son más fáciles de manejar por su tamaño. Sin embargo el sistema simula las condiciones naturales para la incubación de cualquier ave: pollo, pato, gavilán, codornices, pavo, etc. La ventaja de nuestro proyecto es el monitoreo vía Internet, cuya característica se la puede adaptar para monitorear otros tipos de procesos con los cambios respectivos del hardware, en este caso nos centraremos en el proceso de incubación de las codornices. Es un proyecto que surge de la necesidad del agricultor junto con la facilidad de la tecnología. De acuerdo a un análisis en las aves codornices, necesitan de otra ave o mucho tiempo para que se acostumbren a incubar sus propios huevos.

Esta incubadora controlada automáticamente ahorra tiempo y aves, ya que por ave se incuba aproximadamente 10 huevos, mientras con una incubadora podemos incubar de 50 a 100 huevos. Esto hace que el agricultor pueda tener más aves para poder exportar la carne en mayor cantidad y ampliar su visión. Ecuador es un país que últimamente esta desarrollando sus producciones avícolas y el sector floricultor. Con la venida del mencionado TLC, tenemos que buscar soluciones rápidas en este caso tratar de explotar lo que tenemos y demostrar calidad ante los demás países, conjuntamente con los apoyos de Corporaciones y Tecnología.

Los cambios que tienen lugar en el huevo durante la incubación se presentan ordenados y regidos por leyes naturales. Estos cambios se producen, con normalidad, solamente bajo niveles determinados de temperatura, humedad, contenido químico del aire y posiciones del huevo.

Por otra parte, el mismo huevo incubado modifica el medio que lo rodea al emitir calor, gases y vapor de agua hacia el mismo. Podemos definir al régimen de incubación, por tanto, como el medio externo del desarrollo embrionario, condicionado por niveles establecidos de los factores de ese medio. El régimen de incubación es el conjunto de factores físicos presentes en el medio ambiente que rodea al huevo.

Los factores que lo integran son: temperatura, humedad, ventilación y volteo de los huevos. De todos ellos la temperatura oficia como el factor de mayor importancia, ya que, inclusive, las variaciones en sus valores pueden resultar letales para muchos embriones.

En el capitulo 1 se describen los objetivos del sistema y su justificación
En el capitulo 2 se explican los fundamentos teóricos en los que se basa el proyecto tales como el Hardware, Firmware del microcontrolador, Interfaz lógica de la transmisión USB y comunicación Internet
En el capitulo 3 se presenta el diseño del sistema de control y los diagramas de bloques
En el capitulo 4 se describen la fase de implementación, el plano de la construcción de la incubadora y el diagrama.
En el capitulo 5 se presentan las conclusiones y algunas recomendaciones que pueden ser de utilidad para implementar funcionalidades adicionales.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPITULO 1

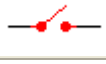
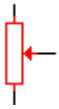
CAPITULO 2

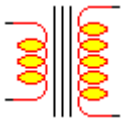
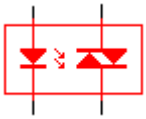
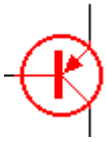
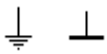
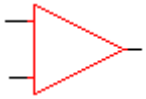
CAPITULO 3

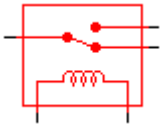
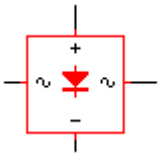
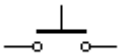
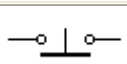
CAPITULO 4

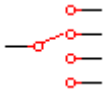
ÍNDICE DE TABLAS

SIMBOLOGÍA ELÉCTRICA

	Resistencia, tiene dos terminales sin polaridad.
	Capacitor Cerámico o No Polarizado.Tiene dos terminales y sin polaridad.
	Capacitor Electrolítico o de Tantalio. Tiene dos terminales y polaridad. El terminal que abarca es el negativo, mientras que el pequeño central es el positivo.
	Diodo LED. Tiene dos contactos normalmente. Tiene polaridad aunque como todo diodo se lo denomina ánodo y cátodo. El cátodo debe ir al positivo y el ánodo al negativo para que el LED se ilumine.
	Interruptor. Tiene solo dos terminales sin polaridad.
	Capacitor variable. Tiene dos terminales con un tornillo para ajustar su capacidad. No tiene polaridad.
	Resistencia Variable, potenciómetro o Trimpot. Tiene tres terminales, dos de los cuales son los extremos de la resistencia y el central es el cursor que se desplaza por la misma. En los potenciómetros suelen estar en ese orden, mientras que en los trimpot varia según su tipo.
	Triac. Tiene tres terminales. Dos son por donde la corriente pasa (AC). Estas no tienen polaridad. La restante es la de control. Su posición y encapsulado varía según el dispositivo.
	Tiristor. Suele denominarse diodo controlado. Sus terminales son ánodo, cátodo y compuerta. Sus cápsula y patillaje cambia según el componente.

	Diodo. Tiene dos terminales, con polaridad. Uno es el ánodo y suele estar representado en el encapsulado por un anillo. El otro es el cátodo.
	Transformador. La cantidad de terminales varía segun cuantos bobinados y tomas tenga. Como mínimo son tres para los autotransformadores y cuatro en adelante para los transformadores. No tienen polaridad aunque si orientación magnética de los bobinados.
	Opto-Triac. Tiene cuatro terminales útiles, aunque suele venir en encapsulados DIL de seis pines. Dos terminales son para el LED que actual como control. Estos terminales son ánodo y cátodo. Otros dos terminales son del Triac, que como todo dispositivo de ese tipo no tiene polaridad.
	Transistor Bipolar PNP. Tiene tres terminales. Uno es la base, que aparece a la izquierda, solo. Otro es el emisor, que aparece a la derecha, arriba, con una flecha hacia el centro. El último es el colector, que aparece a la derecha, abajo.
	Transistor Bipolar NPN. La base esta sola del lado izquierdo. El emisor esta del lado derecho hacia abajo con una flecha, pero en este caso hacia afuera. El colector esta en el lado derecho superior.
	Transistor IGBT PNP. El emisor es el de la flecha, el colector el otro del mismo lado que el emisor mientras que la base esta sola del lado izquierdo.
	Transistor IGBT NPN. Sigue los mismos lineamientos anteriores.
	Cristal de Cuarzo. Tiene dos terminales sin polaridad.
	Puesta a tierra y masa, respectivamente.
	Amplificador Operacional. Tiene básicamente tres terminales. Dos de entrada de las cuales una es inversora (señalada con un -) y otra es no inversora (señalada con un +). La tercera es salida. Adicionalmente tiene dos terminales de alimentación y puede tener otras conexiones

	para, por ejemplo, manejar ganancia.
	Bobina o inductor sobre aire. Tiene dos terminales que no tienen polaridad. Esta armada sobre el aire, sin nucleo. Puede tener devanados intermedios.
	Bobina o inductor sobre núcleo. Idem anterior solo que esta montada sobre una forma.
	Relé. Tiene como mínimo cuatro terminales. Dos de ellos son para controlar la bobina que mueve la llave. Los otros dos (o mas) son de la llave en si.
	Lámpara de Neón. Tiene dos terminales sin polaridad.
	Instrumento de medición. Tiene dos terminales. Si llegase a tener polaridad ésta es representada por signos + y -.
	Conector. Suele esquematizar al conector RCA o al BNC. El terminal central suele ser señal y el envolvente suele ser masa.
	Punto de conexión. Suele representar una toma de control, un pin determinado o una entrada. En su interior se rotula su función abreviada.
	Puente rectificador. Generalmente compuesto por cuatro diodos en serie. Tiene cuatro conexiones.
	Alternativa al puente rectificador. Idem Anterior.
	Pulsador Normal Abierto en estado de reposo. Tiene dos terminales sin polaridad.
	Pulsador Normal Cerrado en estado pulsado. Tiene dos terminales sin polaridad.
	Pulsador Normal Cerrado en estado de reposo. Tiene dos terminales sin polaridad.
	Punto de conexión. Suele representar una entrada o un punto de alimentación.
	Punto de empalme. Se emplea para unir un cable a otro.

	Fusible. Tiene dos terminales y no tiene polaridad.
	Selector. Viene de tres o mas contactos dependiendo de la cantidad de posiciones que tenga. No tiene polaridad aunque si orden de contactos. Cada selector tiene su propio esquema de conexionado.
	Carga. Suele representar una lámpara resistiva, aunque nada dice que sea solo eso.. Tiene dos contactos sin polaridad. De ser una carga polarizada se indica con + y -.
	Motor. Tiene dos contactos a menos que se indique lo contrario en el circuito. Cuando son de alterna no tienen polaridad. Cuando son de continua la polaridad se señala con un + y un -
	Interruptor con piloto de neón. Tiene tres conectores usualmente. Dos de ellos son de la llave y el tercero (que suele ser un delgado alambre) viene de la lámpara de neón para conectar al otro polo y así iluminarla.
	Opto Acoplador con transistor Darlington. Tiene generalmente cinco conexiones aunque la cápsula sea DIL de 6 pines. Dos son para el LED de control y tres para el transistor darlington.

INTRODUCCIÓN

En el campo industrial existen diferentes procesos, que consisten en el tratamiento completo de una serie de entradas de un dispositivo dando como resultado una acción.

En este proyecto, monitoreamos la temperatura, humedad y realizamos el respectivo control PWM para mantener las condiciones de temperatura adecuadas de una **Incubadora de Codornices**.

Como todo proceso tiene un controlador, en este caso hemos seleccionado el PIC 16F877A, que posee un modulo analógico y la facilidad de construir el control PWM con el software MICROCODE STUDIO.

Los Datos de Temperatura y humedad van a ser transferidos a través del cable USB (Universal Serial Bus) ya que debido a su velocidad de transmisión y otras ventajas que detallaremos mas adelante, hemos seleccionado esta interfase plug and play. Luego los datos son visualizados por medio del Software del sistema diseñado en Visual Basic 6.0, en el cual es posible setear los valores de temperatura y la orden de rotación de giro de los huevos.

Además a través de una librería grafica de Visual Basic Trend ActiveX Control, podremos visualizar la curva de temperatura y humedad.

Tenemos un segundo control, que es Vía Internet, donde utilizaremos el control WinSock para enviar los datos servidor- cliente y a través de Internet.

Tanto el servidor como el cliente podrá visualizar la temperatura, Humedad y poder enviar mensajes mutuamente.

CAPÍTULO 1

1. GENERALIDADES

Nuestro proyecto esta dedicado al área industrial y hemos visto conveniente dividirlo en tres partes importantes:

1. Proceso
   Nuestro proceso es la incubación de codornices, implementando la incubadora y el control respectivo.
2. Transmisión de datos entre Incubadora-computadora.
   En nuestro proceso utilizamos la transmisión USB.
3. Transmisión de datos vía Internet
   En nuestro proceso, utilizamos TCP/IP, con el Winsock ya que se pueden crear aplicaciones que coleccionen información y se envien Cliente-Servidor

1.1 OBJETIVO

- Controlar el proceso de incubación de Codornices monitoreado a través del Internet e interfase con Visual Basic con una comunicación USB (Universal Serial Bus).
- Prototipo de una Incubadora para Aves

1.2 SISTEMA DE TRANSMISIÓN DE DATOS

El desarrollo de la computación y su integración con las telecomunicaciones en la telemática han propiciado el surgimiento de nuevas formas de comunicación, que son aceptadas cada vez por más personas. La información a la que se accede a través de Internet combina el texto con la imagen y el sonido, es decir, se trata de una información multimedia, una forma de comunicación que esta conociendo un enorme desarrollo gracias a la generalización de computadores personales dotadas del hardware y software necesarios.

Hoy día resulta muy interesante observar como los avances tecnológicos nos sorprenden por la evolución tan rápida que presentan y cada vez son más fáciles de usar para cualquier persona volviéndose muy amigables y no necesitas ser un experto para poder comprender su funcionamiento, usarlos o instalarlos, este es el caso de Universal Serial Bus, mejor conocido como USB.

1.2.1 TRANSMISIÓN UNIVERSAL SERIAL BUS

Universal Serial Bus, mejor conocido como USB. La tecnología USB contribuye de forma notable al desarrollo de la telefonía mediante PC.

Tanto para las grandes como para las pequeñas empresas, la arquitectura de USB hace posible la fácil conexión a los PC de PBX y teléfonos digitales, sin requerir la instalación de tarjetas especiales de expansión. El ancho de banda de USB permite la conexión de interfaces de alta velocidad

1.2.1.1 FUNCIONAMIENTO Y ESTRUCTURA DEL USB

Trabaja como interfaz para transmisión de datos y distribución de energía, que ha sido introducida en el mercado de PC´s y periféricos para mejorar las lentas interfaces serie (RS-232) y paralelo. Esta interfaz de 4 hilos, 12 Mbps y "plug and play", distribuye 5V para alimentación, transmite datos y está siendo adoptada rápidamente por la industria informática.

Es un bus basado en el paso de un testigo, semejante a otros buses como los de las redes locales en anillo con paso de testigo y las redes FDDI. El controlador USB distribuye testigos por el bus . El dispositivo cuya dirección coincide con la que porta el testigo responde aceptando o enviando datos al controlador. Este también gestiona la distribución de energía a los periféricos que lo requieran.

Emplea una topología de estrellas apiladas que permite el funcionamiento simultáneo de 127 dispositivos a la vez . En la raíz o vértice de las capas, está el controlador anfitrión o host que controla todo el tráfico que circula por el bus . Esta topología permite a muchos dispositivos conectarse a un único bus lógico sin que los dispositivos que se encuentran más abajo en la pirámide sufran retardo . A diferencia de otras arquitecturas, USB no es un bus de almacenamiento y envío, de forma que no se produce retardo en el envío de un paquete de datos hacia capas inferiores.

El sistema de bus serie universal USB consta de tres componentes:

- Controlador
- Hubs o Concentradores
- Periféricos

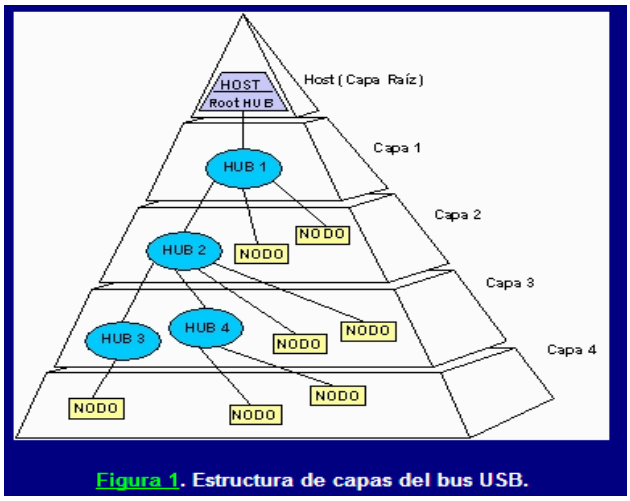


Figura 1- Estructura de capas del bus USB

Controlador

Reside dentro del PC y es responsable de las comunicaciones entre los periféricos USB y la CPU del PC . Es también responsable de la admisión de los periféricos dentro del bus, tanto si se detecta una conexión como una desconexión . Para cada periférico añadido, el controlador determina su tipo y le asigna una dirección lógica para utilizarla siempre en las comunicaciones con el mismo . Si se producen errores durante la conexión, el controlador lo comunica a la CPU, que, a su vez, lo transmite al usuario . Una vez se ha producido la conexión correctamente, el controlador asigna al periférico los recursos del sistema que éste precise para su funcionamiento. El controlador también es responsable del control de flujo de datos entre el periférico y la CPU.

Concentradores o hubs

Son distribuidores inteligentes de datos y alimentación, y hacen posible la conexión a un único puerto USB de 127 dispositivos . De una forma selectiva reparten datos y alimentación hacia sus puertas descendentes y permiten la comunicación hacia su puerta de retorno o ascendente . Un hub de 4 puertos, por ejemplo, acepta datos del PC para un periférico por su puerta de retorno o ascendente y los distribuye a las 4 puertas descendentes si fuera necesario .

Los concentradores también permiten las comunicaciones desde el periférico hacia el PC, aceptando datos en las 4 puertas descendentes y enviándolos hacia el PC por la puerta de retorno.

Además del controlador, el PC también contiene el concentrador raíz. Este es el primer concentrador de toda la cadena que permite a los datos y a la energía pasar a uno o dos conectores USB del PC, y de allí a los 127 periféricos que, como máximo, puede soportar el sistema . Esto es posible añadiendo concentradores adicionales . Por ejemplo, si el PC tiene una única puerta USB y a ella le conectamos un hub o concentrador de 4 puertas, el PC se queda sin más puertas disponibles . Sin embargo, el hub de 4 puertas permite realizar 4 conexiones descendentes . Conectando otro hub de 4 puertas a una de las 4 puertas del primero, habremos creado un total de 7 puertas a partir de una puerta del PC . De esta forma, es decir, añadiendo concentradores, el PC puede soportar hasta 127 periféricos USB.

La mayoría de los concentradores se encontrarán incorporados en los periféricos . Por ejemplo, un monitor USB puede contener un concentrador de 7 puertas incluido dentro de su chasis . El monitor utilizará una de ellas para sus datos y control y le quedarán 6 para conectar allí otros periféricos .

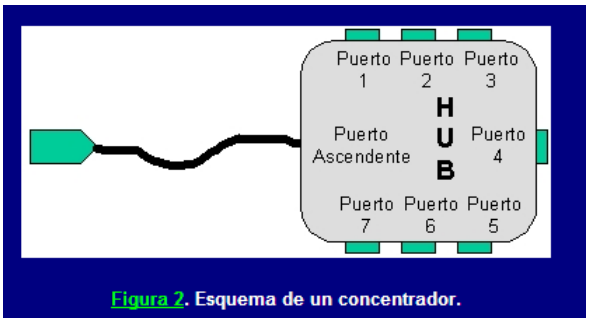


Figura 2.- Esquema de un controlador HUB

Periféricos

USB soporta periféricos de baja y media velocidad . Empleando dos velocidades para la transmisión de datos de 1 . 5 y 12 Mbps se consigue una utilización más eficiente de sus recursos . Los periféricos de baja velocidad tales como teclados, ratones, joysticks, y otros periféricos para juegos, no requieren 12 Mbps . Empleando para ellos 1,5 Mbps, se puede dedicar más recursos del sistema a periféricos tales como monitores, impresoras, módems, scanner, equipos de audio, que precisan de velocidades más altas para transmitir mayor volumen de datos o datos cuya dependencia temporal es más estricta.

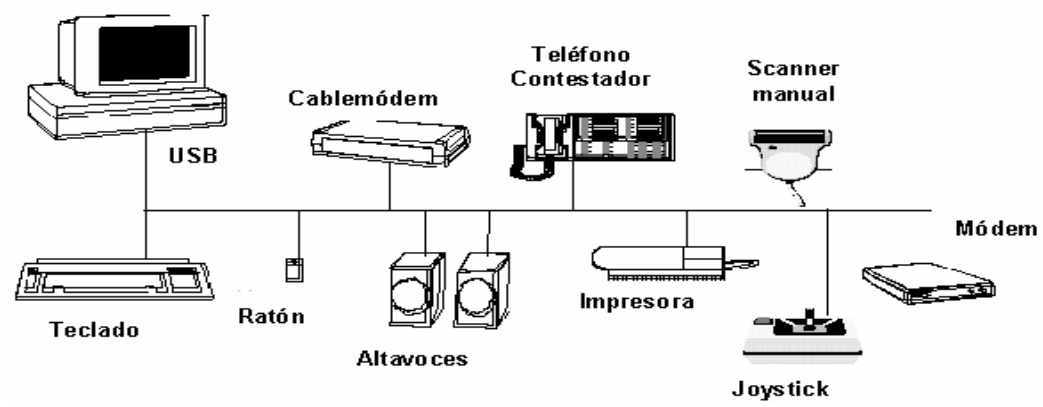


Figura 3.- Periféricos de baja y media velocidad

Diagrama de capas

En el diagrama de capas podemos ver cómo fluye la información entre las diferentes capas a nivel real y a nivel lógico.

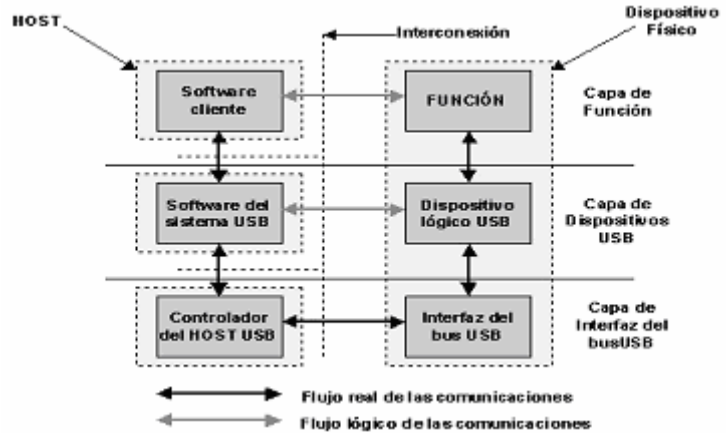


Figura 4.- Diagrama de capas entre el HOST y un dispositivo

En dicha figura está materializada la conexión entre el controlador anfitrión o host y un dispositivo o periférico . Este está constituido por hardware al final de un cable USB y realiza alguna función útil para el usuario.

El software cliente se ejecuta en el host y corresponde a un dispositivo USB; se suministra con el sistema operativo o con el dispositivo USB . El software del sistema USB, es el que soporta USB en un determinado sistema operativo y se suministra con el sistema operativo independientemente de los dispositivos USB o del software cliente.

El controlador anfitrión USB está constituido por el hardware y el software que permite a los dispositivos USB ser conectados al anfitrión. La conexión entre un host y un dispositivo requiere la interacción entre las capas. La capa de interfaz de bus USB proporciona la conexión física entre el host y el

dispositivo. La capa de dispositivo USB es la que permite que el software del sistema USB realice operaciones genéricas USB con el dispositivo.

La capa de función proporciona capacidades adicionales al host vía una adecuada capa de software cliente. Las capas de función y dispositivos USB tienen cada una de ellas una visión de la comunicación lógica dentro de su nivel, aunque la comunicación entre ellas se hace realmente por la capa de interfaz de bus USB.

1.2.1.2 VENTAJAS DEL USB

Conexión más sencilla:

Un Solo tipo de Cable

Gracias al USB prácticamente no se registrarán errores al momento de instalar la impresora,

cámara digital o scanner, etc. Sólo existe un tipo de cable (USA A-B) con conectores distintos en cada extremo, de manera que es imposible conectarlo erróneamente.

Plug and Play :

Cuando se conecta una impresora, cámara fotográfica, o scanner a través de la interfase USB, no es necesario apagar el equipo ni hacer que el sistema busque el nuevo Hardware ya que el sistema automáticamente reconoce el dispositivo conectado e instala los controladores adecuados.

Hot Pluggable:

El usuario podrá conectar y desconectar los dispositivos USB las veces que quiera sin que tenga que apagar y encender la máquina

Mayor Rendimiento:

Velocidad. La gran ventaja de usar el puerto USB en las Mac y PC es la velocidad de transferencia de los datos desde el ordenador a la impresora, cámaras digitales, scanner y entre otros hasta 12 Mbps.
Mucho más rápido que un puerto serial - casi 3 veces más rápido.
Más rápido que un puerto paralelo.
Mayor capacidad de expansión.

Múltiples Dispositivos Conectados de Manera Simultánea.

La tecnología USB permite conexiones en funcionamiento, para que los usuarios puedan incorporar una impresora fácilmente y cuando lo necesiten, en USB, es posible conectar hasta 127 dispositivos a nuestra computadora.
USB ha sido diseñado para las futuras generaciones de PC y deja la puerta abierta a un gran número de aplicaciones tales como audio digital y telefonía de banda ancha.
La compatibilidad universal de USB elimina los riesgos en las ofertas de una gama de productos, posibilitando a los fabricantes (OEMs; Original Equipment Manufacturers) la creación de combinaciones innovadoras de PC, periféricos y software que cubran las necesidades de determinados segmentos de mercado. La norma USB simplifica los procesos de validación y los test de compatibilidad de diferentes combinaciones de hardware y software, de forma que los OEM puedan desarrollar con anticipación

determinados segmentos de mercado y responder con más agilidad a los mercados emergentes.

La tecnología USB contribuirá de forma notable al desarrollo de la telefonía mediante PC. Tanto para las grandes como para las pequeñas empresas, la arquitectura de USB hace posible la fácil conexión a los PC de PBX y teléfonos digitales, sin requerir la instalación de tarjetas especiales de expansión. El ancho de banda de USB permite la conexión de interfaces de alta velocidad (RDSI, PRI, T1, E1) y posibilita la adaptación a normas de telefonía específicas de un país, sin tener que añadir tarjetas adicionales

1.2.2 PROTOCOLOS DE CONTROL DE TRANSMISIÓN Y PROTOCOLO DE INTERNET (TCP/IP)

Una red tiene dos tipos de conexiones, las conexiones físicas que permiten a los ordenadores transmitir y recibir señales directamente y conexiones lógicas, o virtuales, que permiten intercambiar información a las aplicaciones informáticas, por ejemplo a un procesador de textos.

Las conexiones físicas están definidas por el medio empleado para transmitir la señal, por la disposición geométrica de los ordenadores (topología) y por el método usado para compartir información. Las conexiones lógicas, son creadas por los protocolos de red y permiten compartir datos a través de la red entre aplicaciones correspondientes a ordenadores de distinto tipo. Algunas conexiones lógicas emplean software de tipo cliente-servidor y están destinadas principalmente a compartir archivos e impresoras.

El conjunto de Protocolos de Control de Transmisión y Protocolo de Internet (TCP/IP, siglas en inglés), desarrollado originalmente por el Departamento de Defensa estadounidense, es el conjunto de conexiones lógicas empleado por Internet, la red de redes planetaria. El TCP/IP, basado en software de aplicación de igual a igual, crea una conexión entre dos computadoras cualesquiera.

TCP se refiere a un gran cúmulo de protocolos de comunicación que se han desarrollado desde 1970 a partir de su predecesor ARPANET. TCP, podríamos decir que es el que se encarga de transmitir la información y el IP el que se encarga de enrutarla (se encarga de decirnos donde está el destinatario). Todas las máquinas que están conectadas a Internet tienen asignadas un número que se forma con 4 cifras de 3 dígitos (que no pueden superar al número 255). Ejemplo del IP máximo que se puede encontrar: 255.255.255.255

Pero el TCP/IP sólo no es suficiente ya que se puede atacar a una gran variedad de Host en las diversas máquinas, podríamos ir al Host de Finger o al de Telnet y para eso este protocolo necesita una información adicional que es lo que llamamos puerto.

Si tienes una conexión TCP y acceso a los diversos puertos que usan las aplicaciones de Internet como Browsers, FTP, POP, etc.… puedes dialogar con ellas.

Con Visual Studio y en concreto con Visual Basic viene un control llamado Winsock para manejo de sockets.

1.2.2.1 PUERTOS Y SOCKETS

El puerto se usa para identificar un servicio o aplicación concreta dentro de una máquina. Esto es necesario porque en una misma máquina pueden estar corriendo diversos servicios. El puerto le dice al servicio de que hay un cliente que quiere conectarse.

Cuando se diseñaron se llegó a un acuerdo sobre la reserva de una serie de puertos

Puerto	Aplicación
80	http
20 y 21	FTP
70	Gopher
25	SMTP
110	POP 3

Tabla 1.- Aplicación de los puertos en Windows

Los sockets nos llevan al campo más bajo de la programación en Internet. Como habrás podido suponer, programar sockets es algo bastante complejo. Sin embargo, el control para el manejo de sockets de Microsoft te da un acceso muy fácil al Windows Socket API

El API de Windows para sockets también llamado WinSock se basa en el estándar que marcó la Universidad de Berkeley que se usa también en las plataformas UNYS. Mientras que un socket se usa para comunicar dos aplicaciones. El Winsock encapsula todas las acciones de bajo nivel y da a los programadores que lo usen la posibilidad de manejar desde un nivel más alto estos sockets.

Con el Winsock te puedes comunicar con otra aplicación e intercambiar datos ya sea usando el protocolo UDP (User Datagrama Protocol) ó el TCP (Transmisión Control Protocol) los protocolos UDP y TCP son los protocolos

fundamentales de Internet. Por ejemplo, el http correo sobre transferencias TCP.

Con el Winsock se pueden crear aplicaciones que coleccionen información y las envíen a un servidor central o que coleccionen datos de diversos clientes.

Una de las primeras decisiones que tendrás que hacer a la hora de usar el control de Winsock es determinar si vas a usar el protocolo TCP ó el de UDP.

1.2.2.2 CARACTERÍSTICAS DEL PROTOCOLO TCP

1.- Es una conexión basada en el protocolo. Esto quiere decir que el cliente debe empezar la comunicación conectándose a un servidor.

2.- No hay límite en el tamaño de los mensajes. Si es necesario el protocolo romperá el mensaje en trozos más pequeños.

3.- Sin embargo, se basa en cadenas (como opuesto a registros) lo que quiere decir que muchas veces serán necesarias varias lecturas del socket hasta completar el mensaje.

4.- Garantiza que se envía el mensaje desde el cliente al servidor y si no saldrá un error.

En cambio el protocolo UDP, tiene las siguientes características:

1.- No es un protocolo basado en la conexión por lo tanto no es necesario que exista un servidor esperado una llamada

2.- Por tanto no se garantiza que el mensaje sea recibido. La aplicación envía el mensaje al servidor y si no lo recibe no pasa nada.

3.- No hay garantía en el orden de los mensajes.

4.- El tamaño máximo de los mensajes está limitado por la configuración de la red y de los servidores.

Las analogías que se suelen utilizar para estos dos protocolos son el del teléfono y el de la radio. El teléfono necesita (Protocolo TCP) necesita a

alguien al otro lado de la línea pues de lo contrario no se puede establecer la comunicación. La radio, por el contrario, (Protocolo UDP) emite y le da igual que exista alguien al otro lado para recibir las señales.

Si tu aplicación necesita que la otra aplicación reciba la información deberás de usar el TCP y lo mismo si vas a enviar grandes cantidades de datos. No obstante, si son cantidades pequeñas y no hay nadie necesariamente esperándolo podrás usar UDP.

1.3 HERRAMIENTAS PARA GRAFICACIÓN: CST TREND ACTIVEX CONTROL

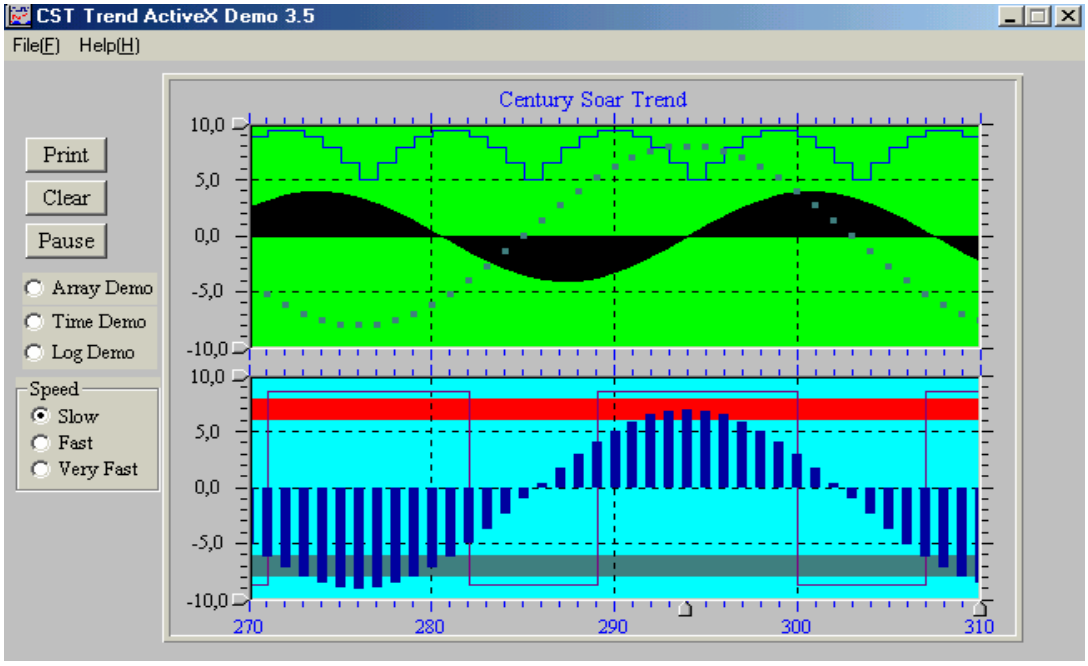


Figura 5.-Pantalla CST Trend ActiveX Control (ecuaciones vs. tiempo en segundos)

Para visualización de graficas, actualmente se utilizan diversidad de programas. Los controles **ActiveX** se pueden implementar en diferentes lenguajes de programación y deben descargarse al disco duro del computador para que los documentos que los utilizan puedan visualizarse en este caso nosotros hemos optado por el visual Basic, teniendo un control llamado "**CST Trend ActiveX Control"** donde su última versión es la 3.6

Este control Trend es multi-propósito , ya que es altamente productivo, en diversas areas tal como en la estadística y en industria, es flexible, dinámico y fácil de aprender, la dimensión en al que se trabaja es en 2D, Con este control podemos visualizar barras y pasteles estadísticos , líneas sólidas y quebradas, y con la ayuda de las propiedades , eventos y métodos de visual,

podemos lograr la interfase RS232, USB, Paralelo de tal forma que podremos apreciar con el TREND CONTROL las señales de afuera.
Otra ventaja es la facilidad del trabajo, ya que este TREND ya viene prediseñada para gráficos, y no necesitamos utilizar los métodos gráficos de visual Basic, y no complicarnos tomando escalas, ni dibujando línea por línea o llamando módulos o funciones para la graficación. Con este TREND podemos ir verificando los valores graficados y sacar resultados estadísticos como Valores máximos, mínimos y promedios.

1.4 GENERALIDADES DE LOS MICROCONTROLADORES

Fue a principios de los años 70 cuando apareció en el mercado electrónico, el circuito integrado denominado microprocesador, que revolucionó el campo de la electrónica digital y analógica de una manera rapidísima y eficaz. Se implementaron numerosos sistemas de control e instrumentación industrial en torno a los microprocesadores, que sin duda alguna se imponían, no solamente en precio sino además en rendimiento y nuevas posibilidades, a los sistemas hasta entonces existentes.

Los microprocesadores funcionan básicamente, como una unidad de procesamiento y control de datos. Para llevar a cabo todas las operaciones que son capaces de realizar, necesitan disponer en su entorno de una serie de elementos, sin los cuales les resultaría imposible llevar a buen término ninguna de ellas.

Los fabricantes de este tipo de microcircuitos, dándose cuenta de todo esto, desarrollaron componentes que engloban en un solo chip gran parte de estos

elementos. Estos nuevos microcircuitos especializados generalmente en aplicaciones industriales, constituyen lo que llamamos los **microcontroladores**. Lógicamente, a medida que elevamos el nivel de exigencia o demanda de nuestro nuevo microcircuito, se eleva su complejidad.

1.4.1 ESTRUCTURA DE UN MIROCONTROLADOR

El diagrama de un sistema microcontrolador sería algo así

Los dispositivos de entrada pueden ser un teclado, un interruptor, un sensor, etc.

Los dispositivos de salida pueden ser LED's, pequeños parlantes, zumbadores, interruptores de potencia (tiristores, optoacopladores), u otros dispositivos como relés, luces, motores DC, resistencias, y otros mas. Tenemos una representación en bloques del microcontrolador, para tener una idea, y poder ver que lo adaptamos tal y cual es un ordenador, con su fuente de alimentación, un circuito de reloj y el chip microcontrolador, el cual dispone de su CPU, sus memorias, y por supuesto, sus puertos de comunicación listos para conectarse al mundo exterior.

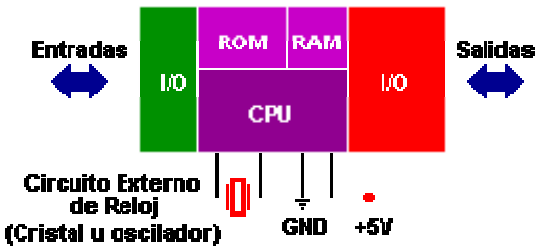


Figura 6.- Representación de un microcontrolador

- **Memoria ROM** (Memoria de sólo lectura)
- **Memoria RAM** (Memoria de acceso aleatorio)
- **Líneas de entrada/salida (I/O)** También llamados puertos
- **Lógica de control** Coordina la interacción entre los demás bloques

Un microcontrolador es un sencillo aunque completo computador con su UCP (Unidad central de proceso), memoria para albergar un programa que es fácil de instalar, memorias para uso general y entradas y salidas para poder ampliarse o comunicarse con el exterior, sistemas de control de tiempo internos y externos, puertos serie y paralelo, conversores A/D y D/A etc. todo ello contenido en un mismo circuito integrado.

Según el tipo empleado pueden diferenciarse en la cantidad y tipo de memoria, cantidad y tipo de entradas y salidas, temporizadores, módulos de control internos y externos, etc. Cada tipo de microcontrolador sirve para una serie de casos y es el creador del producto el que debe de seleccionar que microcontrolador es el idóneo para cada uso.

La aplicación de un microcontrolador en un circuito reduce el número de averías, al reducirse en número de componentes, así como el volumen, el stock y el trabajo. Prácticamente todos los mas importantes fabricantes de componentes del mundo, Intel, Motorola, Philips, Texas, Microchip, etc. fabrican microcontroladores.

1.4.2 DESCRIPCIÓN GENERAL DEL PIC16F877

El microcontrolador PIC16F877 de Microchip pertenece a una gran familia de microcontroladores de 8 bits (bus de datos) que tienen las siguientes características que los distinguen de otras familias:

- Arquitectura Harvard
- Tecnología RISC
- Tecnología CMOS

Estas características se conjugan para lograr un dispositivo altamente eficiente en el uso de la memoria de datos y programa y por lo tanto en la velocidad de ejecución.

Microchip ha dividido sus microcontroladores en tres grandes subfamilias de acuerdo al número de bits de su bus de instrucciones:

Subfamilia	Bits del bus de instrucciones	nomenclatura
Base - Line	12	PIC12XXX y PIC14XXX
Mid – Range	14	PIC16XXX
High - End	16	PIC17XXX y PIC18XXX

Tabla 2.- Subfamilia de los microcontroladores

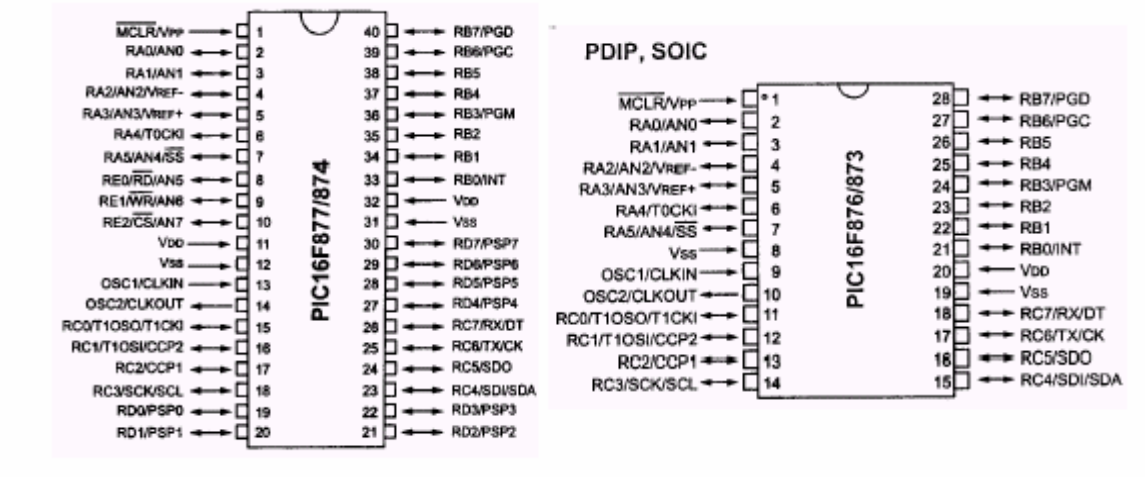


Figura 7.- Empaquetados de los microcontroladores

Nomenclatura

Además de lo mostrado en la tabla anterior, en el nombre específico del microcontrolador pueden aparecer algunas siglas como se muestra en la siguiente tabla:

Tipo de memoria	Rango de voltaje	
	Estándar	Extendido
EPROM	PIC16**C**XXX	PIC16**LC**XXX
ROM	PIC16**CR**XXX	PIC16**LCR**XXX
Flash	PIC16**F**XXX	PIC16**LF**XXX

Tabla 3.- Tabla de Nomenclatura de los microcontroladores

Oscilador

Los PIC de rango medio permiten hasta 8 diferentes modos para el oscilador. El usuario puede seleccionar alguno de estos 8 modos programando 3 bits de configuración del dispositivo denominados:

FOSC2, FOSC1 y FOSC0. En algunos de estos modos el usuario puede indicar que se genere o no una salida del oscilador (CLKOUT) a través de una patita de Entrada/Salida. Los modos de operación se muestran en la siguiente lista:

- LP Baja frecuencia (y bajo consumo de potencia)
- XT Cristal / Resonador cerámico externos, (Media frecuencia)
- HS Alta velocidad (y alta potencia) Cristal/resonador
- RC Resistencia / capacitor externos (mismo que EXTRC con
- CLKOUT)
- EXTRC Resistencia / capacitor externos
- EXTRC Resistencia / Capacitor externos con CLCKOUT
- INTRC Resistencia / Capacitor internos para 4 MHz
- INTRC Resistencia / Capacitor internos para 4 MHz con CLKOUT

Los tres modos LP, XT y HS usan un cristal o resonador externo, la diferencia sin embargo es la ganancia de los drivers internos, lo cual se ve reflejado en el rango de frecuencia admitido y la potencia consumida. En la siguiente tabla se muestran los rangos de frecuencia así como los capacitores recomendados para un oscilador en base a cristal.

Modo	Frecuencia típica	Capacitores recomendados	
		C1	C2
LP	32 khz	68 a 100 pf	68 a 100 pf
	200 khz	15 a 30 pf	15 a 30 pf
XT	100 khz	68 a 150 pf	150 a 200 pf
	2 Mhz	15 a 30 pf	15 a 30 pf
	4 Mhz	15 a 30 pf	15 a 30 pf
HS	8 Mhz	15 a 30 pf	15 a 30 pf
	10 Mhz	15 a 30 pf	15 a 30 pf
	20 Mhz	15 a 30 pf	15 a 30 pf

Tabla 4.- Rango de Frecuencias y Capacitares para un oscilador en base a cristal

FUNCIONES ESPECIALES

- **Conversores análogo a digital (A/D)** en caso de que se requiera medir señales analógicas, por ejemplo temperatura, voltaje, luminosidad, etc.
- **Temporizadores programables (Timer's)** Si se requiere medir períodos de tiempo entre eventos, generar temporizaciones o salidas con frecuencia específica, etc.
- **Interfaz serial RS-232.** Cuando se necesita establecer comunicación con otro microcontrolador o con un computador.
- **Memoria EEPROM** Para desarrollar una aplicación donde los datos no se alteren a pesar de quitar la alimentación, que es un tipo de memoria ROM que se puede programar o borrar eléctricamente sin necesidad de circuitos especiales.
- **Salidas PWM (modulación por ancho de pulso)** Para quienes requieren el control de motores DC o cargas resistivas, existen microcontroladores que pueden ofrecer varias de ellas.
- **Técnica llamada de "Interrupciones"**, (ésta me gustó) Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que está ejecutando, atiende dicha interrupción, y luego continúa con lo que estaba haciendo.

1.4.3 PUERTOS DE COMUNICACIÓN Y DESARROLLO DEL SOFTWARE

Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras

normas y protocolos. Algunos modelos disponen de recursos que permiten directamente esta tarea, entre los que destacan:
UART, adaptador de comunicación serie asíncrona.
USART, adaptador de comunicación serie síncrona y asíncrona
Puerta paralela esclava para poder conectarse con los buses de otros microprocesadores.
USB (Universal Serial Bus), que es un moderno bus serie para los PC.
Bus I^2C, que es un interfaz serie de dos hilos desarrollado por Philips.
CAN (Controller Area Network), para permitir la adaptación con redes de conexionado multiplexado desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles. En EE.UU. se usa el J185O.
Uno de los factores que más importancia tiene a la hora de seleccionar un microcontrolador entre todos los demás es el soporte tanto software como hardware de que dispone. Un buen conjunto de herramientas de desarrollo puede ser decisivo en la elección, ya que pueden suponer una ayuda inestimable en el desarrollo del proyecto.

Las principales herramientas de ayuda al desarrollo de sistemas basados en microcontroladores son:

Ensamblador. La programación en lenguaje ensamblador puede resultar un tanto ardua para el principiante, pero permite desarrollar programas muy eficientes, ya que otorga al programador el dominio absoluto del sistema. Los fabricantes suelen proporcionar el programa ensamblador de forma gratuita y en cualquier caso siempre se puede encontrar una versión gratuita para los microcontroladores más populares.
Compilador. La programación en un lenguaje de alto nivel (como el C ó el Basic) permite disminuir el tiempo de desarrollo de un producto. No obstante, si no se programa con cuidado, el código resultante puede ser mucho más ineficiente que el programado en ensamblador. Las versiones más potentes

suelen ser muy caras, aunque para los microcontroladores más populares pueden encontrarse versiones demo limitadas e incluso compiladores gratuitos.

Depuración. Debido a que los microcontroladores van a controlar dispositivos físicos, los desarrolladores necesitan herramientas que les permitan comprobar el buen funcionamiento del microcontrolador cuando es conectado al resto de circuitos.

Simulador. Son capaces de ejecutar en un PC programas realizados para el microcontrolador. Los simuladores permiten tener un control absoluto sobre la ejecución de un programa, siendo ideales para la depuración de los mismos. Su gran inconveniente es que es difícil simular la entrada y salida de datos del microcontrolador. Tampoco cuentan con los posibles ruidos en las entradas, pero, al menos, permiten el paso físico de la implementación de un modo más seguro y menos costoso, puesto que ahorraremos en grabaciones de chips para la prueba in-situ.

Placas de evaluación. Se trata de pequeños sistemas con un microcontrolador ya montado y que suelen conectarse a un PC desde el que se cargan los programas que se ejecutan en el microcontrolador. Las placas suelen incluir visualizadores LCD, teclados, LEDs, fácil acceso a los pines de E/S, etc. El sistema operativo de la placa recibe el nombre de programa monitor. El programa monitor de algunas placas de evaluación, aparte de permitir cargar programas y datos en la memoria del microcontrolador, puede permitir en cualquier momento realizar ejecución paso a paso, monitorizar el estado del microcontrolador o modificar los valores almacenados los registros o en la memoria.

Emuladores en circuito. Se trata de un instrumento que se coloca entre el PC anfitrión y el zócalo de la tarjeta de circuito impreso donde se alojará el microcontrolador definitivo. El programa es ejecutado desde el PC, pero para la tarjeta de aplicación es como si lo hiciese el mismo microcontrolador que

luego irá en el zócalo. Presenta en pantalla toda la información tal y como luego sucederá cuando se coloque la cápsula.

1.5 CONDICIONES NORMALES PARA LA INCUBACIÓN DE AVES

Los cambios que tienen lugar en el huevo durante la incubación se presentan ordenados y regidos por leyes naturales. Estos cambios se producen, con normalidad, solamente bajo niveles determinados de temperatura, humedad, contenido químico del aire y posiciones del huevo.

Por otra parte, el mismo huevo incubado modifica el medio que lo rodea al emitir calor, gases y vapor de agua hacia el mismo. Podemos definir al régimen de incubación, por tanto, como el medio externo del desarrollo embrionario, condicionado por niveles establecidos de los factores de ese medio. El régimen de incubación es el conjunto de factores físicos presentes en el medio ambiente que rodea al huevo.

Los factores que lo integran son:

- temperatura
- humedad
- ventilación y volteo de los huevos

De todos ellos la temperatura oficia como el factor de mayor importancia, ya que, inclusive, las variaciones en sus valores pueden resultar letales para muchos embriones.

1.5.1 TEMPERATURA

El calentamiento de los huevos durante la incubación artificial se produce mediante el intercambio de calor entre el aire y los huevos. De ahí se deriva, que la temperatura del aire se constituye en el factor fundamental en este proceso.

La temperatura de trabajo en las incubadoras se enmarca entre 37 y 39 grados centígrados. El nivel de temperatura óptimo a aplicar depende del tipo de incubadoras, la calidad y el tamaño de los huevos, la edad de los embriones, además de la especie de que se trate.

En las incubadoras de etapas simples, la temperatura se mantiene a un nivel más alto durante las dos primeras semanas de incubación (para ser más exactos hasta los 13 días). Con posterioridad se disminuye este nivel de temperatura.

En las incubadoras de etapas múltiples, cuando recién comienzan a recibir huevos se fija una temperatura similar a la de las incubadoras de etapas múltiples hasta tanto el gabinete de incubación no haya recibido más de la mitad de su capacidad en huevos. A partir de este momento se mantiene un nivel de temperatura más bajo y se mantiene estable hasta los 18-l9 días de incubación cuando los huevos ya están en el gabinete de nacimiento.

En todos los casos es necesario disminuir el nivel de temperatura durante los últimos días (2 a 3) de incubación, es decir, que la temperatura se diferencia de acuerdo a las etapas de incubación.

Relación entre la temperatura del aire de la incubadora y los huevos incubados.

Al comienzo de la incubación, los embriones no están preparados funcionalmente (ni orgánicamente) para emitir calor. Por esto reaccionan como los organismos de sangre fría, es decir, cuando la temperatura del aire se eleva, aumenta el metabolismo de los embriones. Si la temperatura disminuye, el metabolismo decrece igualmente. Por tanto, el aumento de la temperatura favorece la multiplicación celular, la formación de las capas y las membranas embrionarias (alantoides, corion, amnios y saco vitelino), así como la nutrición. En resumen, se incrementa el ritmo de crecimiento y desarrollo de los embriones. Al final de la incubación, cuando ya la emisión de calor es alta, la disminución de la temperatura (dentro de los límites normales) actúa, por su parte, de forma completamente inversa; estimula el consumo de los nutrientes ó lo que es lo mismo, acelera el metabolismo y el desarrollo en los embriones.

1.5.2 HUMEDAD

La humedad ideal es de entre el 50% y el 60% los 17 primeros días, y de un 65% los 3 últimos días del período de incubación, según los manuales. El incremento de temperatura, por parte de los propios huevos, facilita la rotura de la cáscara por las codornices al nacer. De hecho, cuando empieza la eclosión se puede incrementar la humedad relativa hasta el 65%. Por el hecho de ser porosa la cáscara del huevo, este va perdiendo agua que se sustituye por aire.

Con la humedad del ambiente se puede regular el ritmo al que el huevo pierde agua, en concreto, mientras mayor sea la humedad del ambiente, menor será el ritmo de pérdida de agua en el huevo.

Se va formando una cámara de aire, que es el primero que respira el pollito y permite que este se pueda girar para ponerse en posición para poder romper la cáscara en el momento de la eclosión. A lo largo del periodo de incubación de agua, es normal una pérdida de peso en el huevo por esta pérdida de agua del 14%.

Cuando los pollitos se han liberado completamente de la cáscara, se puede disminuir la humedad y la temperatura rápidamente, hasta el 50% con el fin de que se seque su plumaje.

La humedad, se regula gracias a la propia temperatura que estamos controlando, debido a que según las pruebas que hemos hecho , cuando la temperatura alcanza 38 a 39 grados centígrados , la humedad esta alrededor del 50% al 65%. Lo cual es una humedad ideal para las codornices u otras aves.

Para esto, nosotros tomamos medidas de la temperatura y humedad inicial con un equipo de medición, lo cual vimos que la humedad es un parámetro secundario, ya que este varia de acuerdo a la temperatura, y según las pruebas hechas obseravmos que la temperatura con el sistema de ventilación, y controlando el sistema de volteo, ayudan a que la humedad tenga un rango aceptable. La cual ustedes podrán apreciarla a través de una pantalla LCD y en la pantalla de visual.

Debemos de tomar en cuenta que en nuestro País la humedad es alta, y por ende no necesitamos calentar agua para generar vapor, es suficiente colocar un recipiente lleno de agua, ya que con la temperatura que controlamos y demás variables, obtenemos la humedad del 50-65%.

1.5.3 VENTILACIÓN Y RENOVACIÓN DE AIRE

El problema de la ventilación debe ser abordado desde dos ángulos: la circulación de aire propiamente dicha y la reventilación o recambio de aire. Mediante el aire que circula en el interior del gabinete de incubación, llega a los huevos el calor y la humedad necesaria.

El aire refresca el medio que rodea los huevos, en algunos casos y en otros contribuye a calentarlo. Por otra parte, el recambio de aire constante es necesario para la extracción del exceso de calor que pudiera acumularse en el interior del gabinete de incubación y asegurar la pureza del aire.

Durante la incubación el huevo absorbe oxígeno y elimina anhídrido carbónico en gran cantidad**.** Solamente una adecuada reventilación garantiza buenos resultados en la incubación.

La correcta circulación de aire en el gabinete se garantiza mediante el funcionamiento de los ventiladores, los inyectores ó los extractores de aire, las compuertas u orificios de entrada y salida, etc. Para que la circulación de aire sea eficiente es importante también un buen funcionamiento del sistema de volteo, ya que el aire se mueve mejor entre las bandejas, cuando las mismas se hallan en posición inclinada

El sistema de renovación del aire puede ser muy simple, basta con realizar unos pequeños agujeros (de unos 12-20 mm.), por la zona baja de incubadora y otros por la parte alta para que la acción del aire caliente cuando sube realice todo el trabajo, (efecto chimenea).

1.5.4 POSICIÓN DE LOS HUEVOS DURANTE LA INCUBACIÓN.

El desarrollo de los embriones transcurre normalmente sólo cuando los huevos son volteados (virados) periódicamente durante los primeros 18 días de incubación.

En la incubación natural, la gallina voltea los huevos que incuba con cierta frecuencia, de ahí que en el proceso de incubación artificial sea necesario repetir este procedimiento mediante medios mecánicos. El huevo, como se ha explicado antes, pierde agua durante todo el período de incubación, es decir, sufre un proceso de desecamiento.

Por este motivo, el embrión está expuesto a pegarse a las membranas internas de la cáscara, lo que puede provocar su muerte, en particular durante los primeros seis días de incubación. A esto contribuye el hecho de que el peso específico del embrión lo lleva a mantenerse en la parte superior de la yema, durante los primeros días, por debajo y muy cercano a la cáscara, en la zona de la cámara de aire. Por otra parte, la posición del huevo influye sobre la posición futura que adoptará el pollito en el momento de prepararse para la eclosión. ¡Esto es de capital importancia para obtener un alto por ciento de nacimiento!

La posición del embrión se define ya desde las 36 a 48 horas de incubación. En este momento el embrión descansa en la yema, de manera transversal, a lo largo del eje menor. Con posterioridad la cabeza del embrión comienza a separarse de la yema y girar hacia la izquierda. Hacia el 5to. día de incubación, el embrión se halla cerca de la cámara de aire. A partir del 11no. día, cuando el cuerpo del embrión pesa más que su cabeza, el mismo efectúa un giro a la izquierda, lo que provoca que el cuerpo descienda en dirección al polo fino del huevo. A los 14 días, el cuerpo del embrión está situado a lo largo del eje mayor del huevo, con la cabeza dirigida hacia el

polo grueso. Esta es la posición correcta y necesaria que debe adoptar el pollito para el nacimiento.

1.6 TRANSDUCTORES DE TEMPERATURA

Es fácil realizar medidas de la temperatura con un sistema de adquisición de datos, pero la realización de medidas de temperatura *exactas y repetibles* no es tan fácil.

La temperatura es un factor de medida engañoso debido a su simplicidad. A menudo pensamos en ella como un simple número, pero en realidad es una estructura estadística cuya exactitud y repetitividad pueden verse afectadas por la masa térmica, el tiempo de medida, el ruido eléctrico y los algoritmos de medida.

Sin embargo existe cuatro tipos más corrientes de transductores de temperatura que se usan en los sistemas de adquisición de datos: detectores de temperatura de resistencia (RTD), termistores, sensores de IC y termopares. Ningún transductor es el mejor en todas las situaciones de medida, por lo que tenemos que saber cuándo debe utilizarse cada uno de ellos.

	RTD	Termistor	Sensor de IC	Termopar
Ventajas	Más estable. Más preciso. Más lineal que Los Termopares	Alto rendimiento Rápido Medida de dos hilos	El más lineal El de más alto rendimiento Económico	Autoalimentado Robusto Económico Amplia gama de temperaturas
Desventajas	Caro. Lento. Precisa fuente de Alimentación Pequeño cambio de resistencia. Medida de 4 hilos Autocalentable	No lineal. Rango de Temperaturas limitado. Frágil. Precisa fuente de Alimentación. Autocalentable	Limitado a < 250 ºC Precisa fuente de alimentación Lento Autocalentable Configuraciones limitadas	No lineal Baja tensión Precisa referencia El menos estable El menos sensible

Tabla 5.- Ventajas y Desventajas de los transductores de Temperatura

1.7 TIPOS DE RESISTENCIA PARA EL PROCESO DE CALENTAMIENTO

Generalmente las resistencias de pueden dividir en 4 tipos:

- Calentamiento de aire, solido y líquido.
- Materiales de Cuarzo y cerámica

PRODUCTOS: CALENTAMIENTO DE AIRE

- Elementos Aleteados
- Resistencias Blindadas
- Espirales de Niquel/Cromo
- Baterías de Calefacción con cualquiera de los elementos citados

Figura 8.- Productos para el calentamiento de Aire

PRODUCTOS: CERÁMICA Y CUARZO

- Resistencias montadas en candela o placa cerámica
- Pantallas de infrarrojos
- Tubos infrarrojos de cuarzo

Figura 9.- Productos de cerámica y Cuarzo para calentamiento

1.8 INCUBADORAS EN EL MERCADO

MODELO G 50
Una bandeja para 50 huevos de gallina o su equivalente a las demás aves. Volteo manual,termostato común simple con bandeja metálica para humedad. Construida en madera multilaminada de Guatambú.

MODELO E 50
Una bandeja para 50 huevos de gallina o su equivalente a las demás aves. Volteo manual Regulador Electrónico de temperatura, con bandeja metálica para humedad. Construida en madera multilaminada de Guatambú.

MODELO G 120
Una bandeja de 120 huevos de gallina o su equivalencia a las demás aves. Volteo mecánico. Termostato común doble. Bandeja metálica para la humedad. Construida en madera multilaminada de Guatambú.

MODELO E 120
Una bandeja de 120 huevos de gallina o su equivalencia a las demás aves. Volteo mecánico. Regulador Electrónico de Temperatura y bandeja metálica para la humedad. Construida en madera multilaminada de Guatambú.

MODELO G 200
Dos secciones independientes con una bandeja por sección, con capacidad de 100 huevos de gallina o su equivalencia a las demás aves. Capacidad total 200 huevos. Volteo mecánico. Termostatos comunes dobles. Bandejas metálicas para la humedad Construida en madera multilaminada de Guatambú.

MODELO G 360

Dos secciones independientes con una bandeja por sección con capacidad de 180 huevos de gallina o su equivalencia a las demás aves. Capacidad total 360 huevos. Volteo mecánico. Termostatos comunes dobles Bandejas metálicas para la humedad. Construida en madera multilaminada de Guatambú.

MODELO E 360

Dos secciones independientes con una bandeja por sección con capacidad de 180 huevos de gallina o su equivalencia a las demás aves. Capacidad total 360 huevos Volteo mecánico. Regulador Electrónico de Temperatura. Bandejas metálicas para la humedad. Construidas en madera multilaminada de Guatambú.

Figura 10.- Modelos de Incubadora

1.9 JUSTIFICACIÓN DE LA TESIS.

Definitivamente, la avicultura ecuatoriana, en especial la del pollo, muestra un futuro alentador, gracias, de un lado, a la buena aceptación que esta carne tiene entre la población local, y de otro, a los esfuerzos que los cultivadores de materias primas -maíz y soya- vienen haciendo por mejorar su productividad, lo que terminará por favorecer la competitividad de la cadena. Futuro promisorio que será realidad en la medida en que los productores de pollo y huevo desarrollen procesos de innovación tecnológica e implementen alianzas estratégicas en toda la cadena, que les permitan competir en mejores condiciones con sus similares del MERCOSUR y del Area de Libre Comercio de las Américas, ALCA.

La avicultura, uno de los pilares fundamentales del sector agropecuario ecuatoriano, ha basado su estrategia de desarrollo en la consolidación de la cadena agroindustrial a través de alianzas estratégicas que involucran a productores de las materias primas, industriales y abastecedoras de las industrias avícolas. Las mayores inversiones en esta cadena durante los últimos tres años han permitido obtener parámetros productivos adecuados en sus diferentes eslabones, gracias a lo cual le ha sido posible abastecer el mercado interno y salir al exterior, especialmente a Colombia.

Basándonos en estas expectativas nosotros escogimos el proceso de incubación de aves, específicamente la incubación de huevos de codornices, que a pesar de su valor proteico, podemos apreciar los diferentes controles analógicos y digitales que realizamos a través de diferentes lenguajes como lo es el Visual Basic, que debido a su fácil manejo, nos beneficia para alcanzar nuestros logros. Hemos elaborado la transmisión con el tan famoso USB, que tiene una gran velocidad y es una interfase plug and play, haciendo este proceso rápido y eficaz. Y añadimos al proyecto el control vía internet, que conjuntamente con Visual Basic encontramos el control

Winsock, el cual pudimos detectar la IP y establecer una comunicación con otra aplicación e intercambiar usando el protocolo TCP/IP (Control de Transmisión y Protocolo de Internet)

CAPITULO 2

2. METODOLOGIA

El proyecto utilizara el puerto USB de la computadora, circuitos de potencia, así como un circuito para la adquisición de datos; este se encarga de mandar datos al puerto USB que son interpretados por una interfaz gráfica realizada en Visual Basic, que nos mostrará el estado de la temperatura y humedad. Estas variables podrán ser monitoreadas por un Cliente que tenga acceso a Internet, debido a que los datos percibidos serán de un Servidor con IP pública.

Entradas/Salidas Del PIC16F877A	**Propósito**	**Procedimiento**
Sensor de temperatura	Medir la temperatura de huevos	Conversión Analógica Monitoreo y Control PWM
Sensor de humedad	Medir humedad de la incubadora	Conversión Analógica Monitoreo
Puerta	Botonera de Seguridad	Verifica el estado de la puerta
Set point de temperatura	SP= 37 00 SP= 38 01 SP= 39 10 SP= 40 11	Ingreso desde teclado para el respectivo control PWM.
Códigos recibidos del USB	Paquetes 1 byte=00,10…FF	Permiten el paso de las variables que pida el software del sistema USB (temperatura, humedad, set point)
Salidas Digitales (1 byte)	Humedad y Temperatura	Señales transmitidas del PIC cuando estas sean pedidas por el sistema USB.

Tabla 6.- Procedimientos de las señales del Microcontrolador.

2.1 HARDWARE

El hardware esta constituido por la electrónica de potencia y el circuito de transmisión de Datos por USB.

Procedimiento de los datos de Humedad y Temperatura

- Los datos sensados de la temperatura pasaran al PIC, para ser convertidos en señales digitales por medio del modulo analógico que posee este microcontrolador. Inmediatamente podrán ser visualizados en la pantalla del computador y en la LCD.
- Los datos de la humedad pasaran al PIC, para ser convertidos en datos digitales por medio del nódulo analógico que posee este microcontrolador. Inmediatamente podrán ser visualizados en la pantalla del computador y en la LCD.
- El dato de la temperatura servirá para el control principal de este proceso.
- Cuando la temperatura este dentro del PIC, esta se comparada con un set point digitalizado por nosotros desde la computadora.
- El set point es la temperatura a la cual nosotros deseamos que la Incubadora se encuentre, por al motivo el PIC dará una serie de pulsos controlados para manejar el prendido y apagado de una resistencia que generara calor para los huevos.

La humedad para este proceso se debe de encontrar en un rango del 70% al 85%. Esta humedad es generada con tan solo controlar la temperatura entre un rango de 37-39 °C. Por causa del aire caliente circulando por la cámara, creara la humedad deseada.

2.1.1 ELECTRONICA DE POTENCIA

2.1.1.1 Transmisor de Temperatura y Humedad

Sensor de Humedad LM35

En el proyecto utilizaremos un termistor para medir la temperatura y un Transmisor de Humedad.

Una de las características del Sensor de temperatura lm35 que utilizaremos es el rango de medición que va 2°C a 150°C, , su salida es lineal y por tal motivo utilizaremos este IC.

V OUT=+1500 mV en +150°C

= +250 mV en +25°C

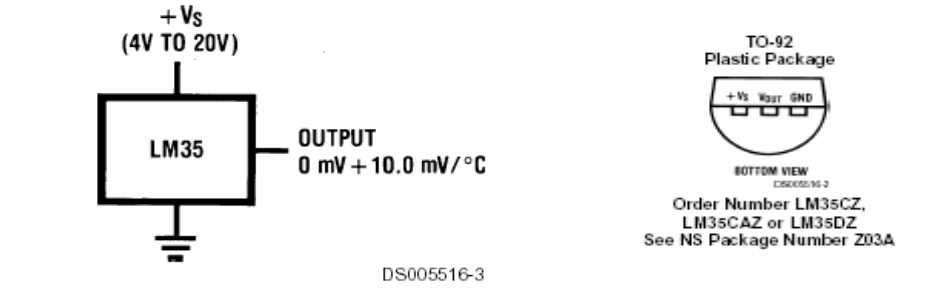


Figura 11.- Sensor de Temperatura LM35

Entonces vemos que el voltaje de salida es de 10mv por grados, el rango de salida va desde 0V hasta 1,5 V. Cuando tengamos 39°C la salida del sensor mostrara 0.39 voltios, es decir que antes de ingresa la señal al pic debemos de amplificarla de tal manera que la salida sea de 0V a 5V.

Sacando la ecuación de amplificación tendremos que

$$\frac{Vo}{Vi} = \frac{R2}{R1} + 1$$

Vo = voltaje de entrada al PIC

Vi = voltaje de salida del sensor

R2 = 2K

R1 = 4,7 K

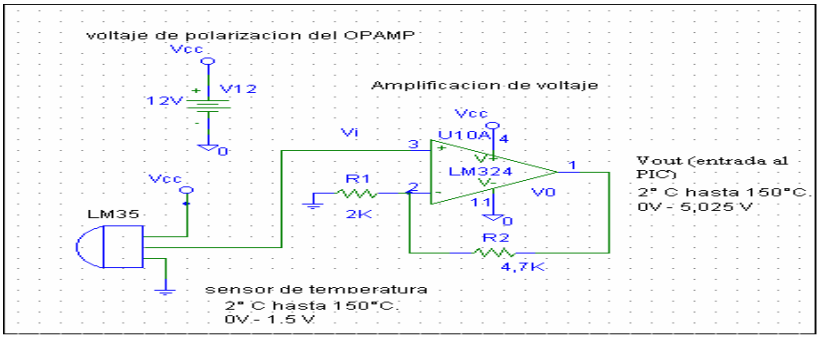


Figura 12.- Amplificación y Acondicionamiento del LM35

Entonces tenemos que el voltaje de salida será de Vo = 3,35 Vi

Vout = 5,025 V en +150°C

0,8325 V en 25°C

1,30 V en 39°C

Este voltaje será la entrada del voltaje analógico al PIC16F877

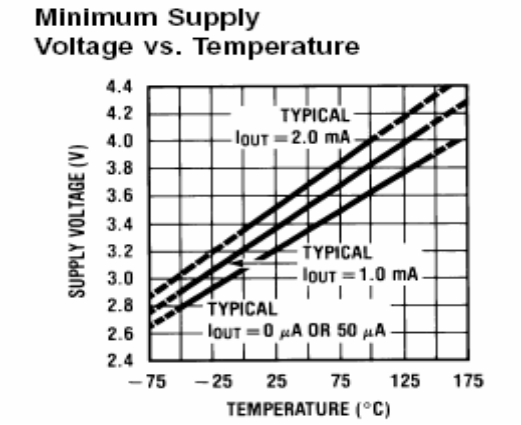


Figura 13.- Voltaje de Salida vs. Temperatura

Transmisor de Humedad C7600A, C Solid State Humidity Sensors

Este sensor es usado en cámaras de aires y son usados con controladores que pueden procesar a 4 a 20 ma

Figura 14.- Transmisor de Humedad C7600A,C

Este sensor en montado en las paredes, tiene una entrada de voltaje de 12 a 40 Vdc, con una salida de 4 a 20 ma. La relación de salida de corriente versus la humedad relativa es presentada en la tabla 6 y en la figura 17 y 18.
Potencia de consumo: 0.30 VA.
Rango de humedad: 10 a 90%RH

Model	Relationship[a]
C7600A	I = -0.16 x RH + 20
C7600C	I = +0.16 x RH + 4

Tabla 7. - Corriente= sensor output (in ma); RH= percent relative humidity

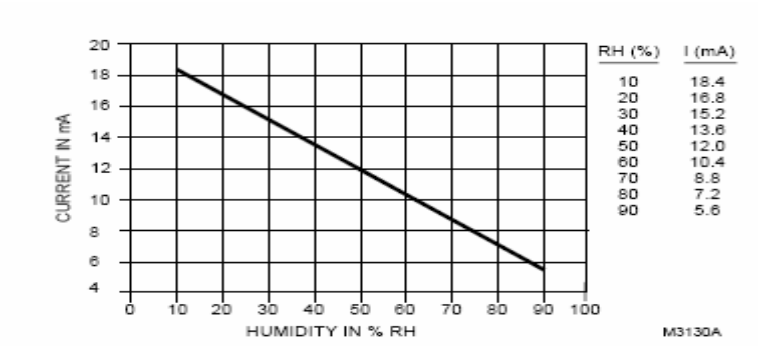


Figura 15.- C7600A output current vs. Relative humidity

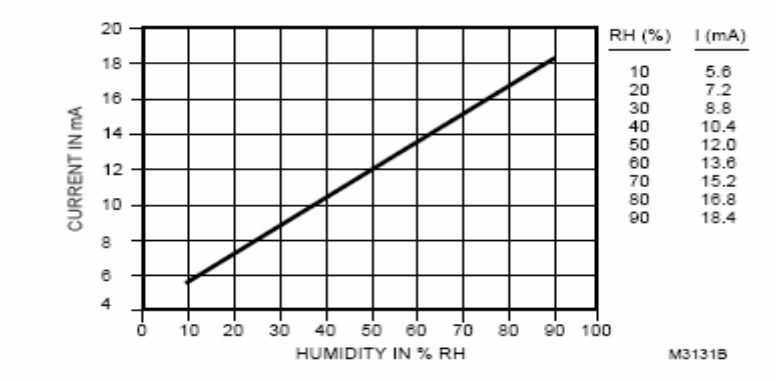


Figura 16.- C7600C output current vs. relative humidity

Esta señal entrara a una entrada analógica del PIC16F8774A, la cual solo recibe voltaje, de tal manera que debemos convertir la corriente a voltaje, colocando una resistencia de 250 ohm, donde su rango será de 1 a 5 V, voltaje adecuado para la lectura del modulo analógico en el microcontrolador

A continuación mostraremos la temperatura y Humedad que requieren las diferentes aves para incubar sus huevos

Aves	Tiempo	Temperatura	Humedad	Recomendaciones	Nacimiento
Gallinas	21 días	38 ºC-39 ºC	50-60%	Voltear 2 veces por día hasta el día 18.	Dura 24 a 48 horas con humedad de 60%.
Patos	28 a 33 días según las razas.	38º C hasta el día 21, luego 39º C.	50-60%	Como necesitan humedad se los puede rociar con agua tibia. Voltear regularmente hasta el día 26.	Dura de 24 a 48 horas a 28ºC y 75% de humedad.
Pavos	Aproxima-damente 28 días.	38-39ºC primera y segunda semana, resto 39	50-65%	Se pueden rociar con agua tibia si es necesario. Voltear tres veces por día y ventilar una vez por día a 29º C.	Dura 48 horas aproximadamente con 70% de humedad.
Codornices	Varía entre 16 y 23 días.	37-39-ºC	50-65%	Ventilación normal. Volteo: a las 48 hs., 2 veces diarias. Rociar con agua tibia.	60% de humedad, no dejar los BB más de 18 hs.

Tabla 8.- Requerimiento para la incubación de las Aves

2.1.1.2 CIRCUITO PARA CALENTAMIENTO DE LA INCUBADORA Y SU FUNCIONAMIENTO

Para nuestro proceso utilizaremos una resistencia de Cuarzo Tubular para el calentamiento de los huevos.

Figura 17.- Resistencia de Calentamiento

Uso de la resistencia: calentamiento del ambiente

Potencia: 600 w

Voltios aplicados: 120 v

Las resistencias tubulares de cuarzo consisten en una bobina de alambre de resistencia, alojada en un tubo de cuarzo (silicio puro vitrificado)

Características:

• Eficiencia Radiante - 50%

• Construcción Robusta de Bajo Costo

• Extremadamente Larga Vida de Operación

• Rango de Longitud de Onda Infrarroja de 3.0 a 6 µm

Para controlar una carga, en este caso la resistencia de cerámica con una entrada digital, necesitamos un **OPTOISOLATORS TRIAC DRIVER OUTPUT**, en este caso utilizaremos un MOC3041M y un **TRIAC BT136500D**

Configuración del MOC3041M

Un optoacoplador de potencia consiste en un circuito electrónico, cuyo principal objetivo es aislar el circuito de la parte de potencia, los componentes a utilizar son los siguientes. El objetivo de este optoacoplador de potencia es activar una carga que tiene que ser alimentada con una corriente alterna, y queremos que esté asilada de nuestro circuito electrónico, ya que este puede ser dañado

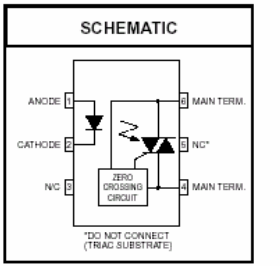


Figura 18.- Top view del MOC3041M

Configuración del TRIAC BT136500D

El triac es un dispositivo semiconductor de tres terminales que se usa para controlar el flujo de corriente promedio a una carga, con la particularidad de que conduce en ambos sentidos y puede ser bloqueado por inversión de la tensión o al disminuir la corriente por debajo del valor de mantenimiento. El triac puede ser disparado independientemente de la polarización de puerta, es decir, mediante una corriente de puerta positiva o negativa.

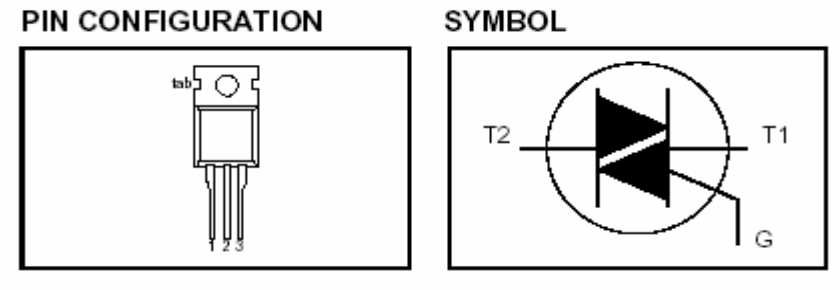


Figura 19.- Configuración de los pines en el Triac BT136500D

Como hemos dicho, el Triac posee dos ánodos denominados (MT1 y MT2) y una compuerta G.

La polaridad de la compuerta G y la polaridad del ánodo 2, se miden con respecto al ánodo 1.

El triac puede ser disparado en cualquiera de los dos cuadrantes I y III mediante la aplicación entre los terminales de compuerta G y MT1 de un impulso positivo o negativo. Esto le da una facilidad de empleo grande y simplifica mucho el circuito de disparo

Funcionamiento:

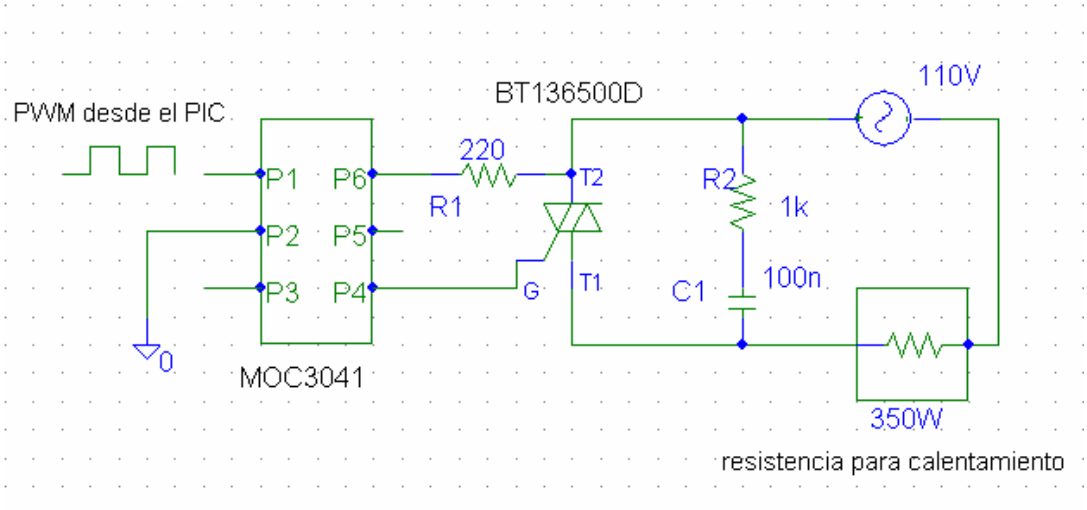


Figura 20.- Calentamiento del ambiente en la incubadora

En la FIG.20 puede verse un circuito que gobierna una resistencia, mediante un triac BT36500D). La señal de control (pwm) llega desde un circuito de mando en este caso, el PIC16F877A, luego circulará corriente a través del diodo emisor perteneciente al MOC3041 (opto acoplador). Dicho diodo emite un haz luminoso que hace conducir al fototriac a través de R2 tomando la tensión del ánodo del triac de potencia. Este proceso produce una tensión de puerta suficiente para excitar al triac principal que pasa al estado de conducción provocando que circule corriente por la resistencia y así provocamos el calor necesario para las codornices.

Debemos recordar que el triac se desactiva automáticamente cada vez que la corriente pasa por cero, es decir, en cada semiciclo, por lo que es necesario redisparar el triac en cada semionda o bien mantenerlo con la señal de control activada durante el tiempo que consideremos oportuno. Por tal motivo con el pulso que sale del PIC (pwm) regulamos el encendido y apagado de la resistencia, generando el calor necesario, sensando la

temperatura el LM35. Como podemos apreciar, entre los terminales de salida del triac se sitúa una red RC cuya misión es proteger al semiconductor de potencia, de las posibles sobrecargas que se puedan producir por las corrientes inductivas de la carga, evitando además cebados no deseados.
Es importante tener en cuenta que el triac debe ir montado sobre un disipador de calor constituido a base de aletas de aluminio de forma que el semiconductor se refrigere adecuadamente.

2.1.1.2 MOTOR DE PASO: CIRCUITO CONTROLADOR

Voltear significa que la cara que estaba hacia arriba debe quedar hacia abajo y viceversa (siempre acostados). Este proceso es fundamental para el éxito de la incubación.
Motor de Paso: Vn: 4.2 V
In : 1,3 amp
Cada dia desde el 2º hasta el 10º día de incubación, se girarán dos veces .El sistema para girar los huevos más barato posible, son las manos, pero es preferible lavárselas bien antes de tocarlos, pues en las manos tenemos una cera que de tocar mucho los huevos se pega en la cáscara, taponando los poros que utiliza el embrión para respirar.

Figura 21.- Motor Paso-Paso

Los motores paso a paso son ideales para la construcción de mecanismos en donde se requieren movimientos muy precisos. Controlamos el giro a través

de Software y lo cableamos al dispositivo USB. El USB envía la secuencia en 4 bits que serán cableados a los transistores NECD1308 para movilizar el motor.

La frecuencia de volteo óptima es de una vez cada 1 ó 2 horas. El giro debe alcanzar los 90 grados y los huevos son mantenidos a 45 grados de una vertical imaginaria.

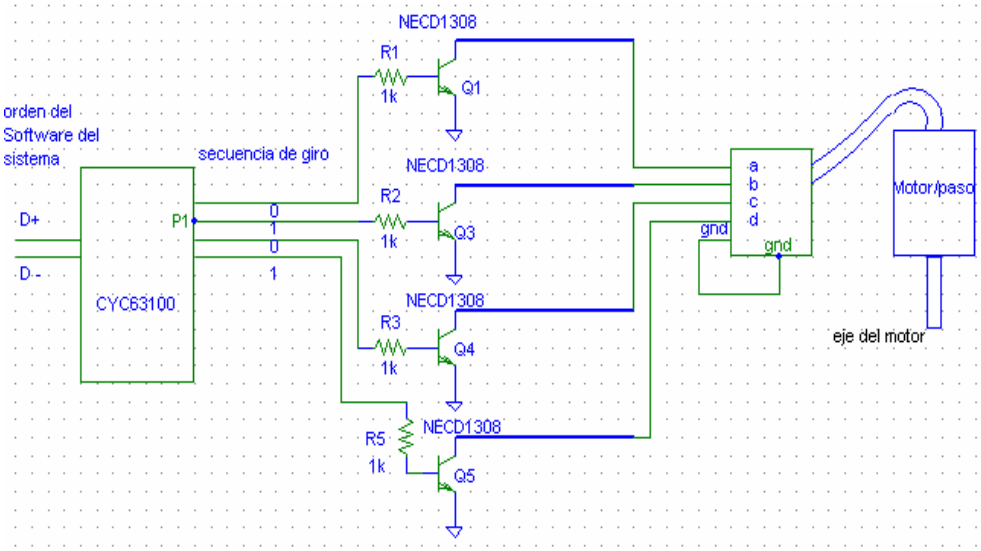


Figura 22.- Controlador del Motor Paso-Paso

La característica principal de estos motores es el hecho de poder moverlos un paso a la vez por cada pulso que se le aplique. Este paso puede variar desde 90° hasta pequeños movimientos de tan solo 1.8°, es decir, que se necesitarán 4 pasos en el primer caso (90°) y 200 para el segundo caso (1.8°), para completar un giro completo de 360°.

Estos motores poseen la habilidad de poder quedar enclavados en una posición o bien totalmente libres. Si una o más de sus bobinas está energizada, el motor estará enclavado en la posición correspondiente y por el contrario quedará completamente libre si no circula corriente por ninguna de sus bobinas.

Cuando se trabaja con motores P-P usados o bien nuevos, pero de los cuales no tenemos hojas de datos. Es posible averiguar la distribución de los cables a los bobinados y el cable común en un motor de paso unipolar de 5 o 6 cables siguiendo las instrucciones que se detallan a continuación:

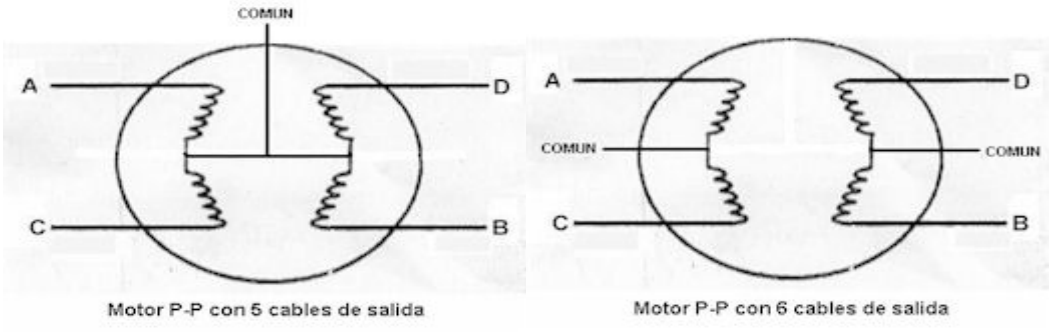


Figura 23.- Motor P-P con 5 y 6 cables de salida

1. **Aislando el cable(s) común que va a la fuente de alimentación:**

Como se aprecia en las figuras anteriores, en el caso de motores con 6 cables, estos poseen dos cables *comunes*, pero generalmente poseen el mismo color, por lo que lo mejor es unirlos antes de comenzar las pruebas.
Usando un tester para chequear la resistencia entre pares de cables, el cable común será el único que tenga la mitad del valor de la resistencia entre ella y el resto de los cables.
Esto es debido a que el cable *común* tiene una bobina entre ella y cualquier otro cable, mientras que cada uno de los otros cables tienen dos bobinas entre ellos. De ahí la mitad de la resistencia medida en el cable *común*.

2. Identificando los cables de las bobinas (A, B, C y D):

Aplicar un voltaje al cable *común* (generalmente 12 volts, pero puede ser más o menos) y manteniendo uno de los otros cables a masa (GND) mientras vamos poniendo a masa cada uno de los demás cables de forma alternada y observando los resultados.

El proceso se puede apreciar en el siguiente cuadro:

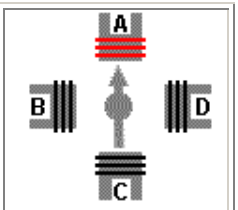
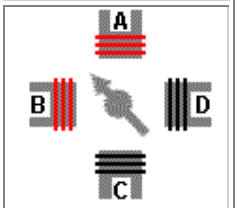
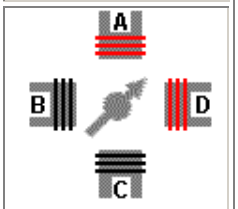
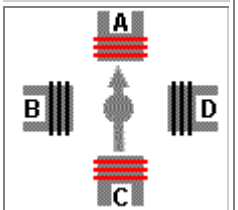
Seleccionar un cable y conectarlo a masa. Ese será llamado cable **A**.	
Manteniendo el cable **A** conectado a masa, probar cuál de los tres cables restantes provoca un paso en sentido antihorario al ser conectado también a masa. Ese será el cable **B**.	
Manteniendo el cable **A** conectado a masa, probar cuál de los dos cables restantes provoca un paso en sentido horario al ser conectado a masa. Ese será el cable **D**.	
El último cable debería ser el cable **C**. Para comprobarlo, basta con conectarlo a masa, lo que no debería generar movimiento alguno debido a que es la bobina opuesta a la **A**.	

Tabla 9.- Identificación de los cables de la bobina del motor

Secuencias para manejar motores paso a paso Unipolares

Existen tres secuencias posibles para este tipo de motores, las cuales se detallan a continuación. Todas las secuencias comienzan nuevamente por el paso 1 una vez alcanzado el paso final (4 u 8). Para revertir el sentido de giro, simplemente se deben ejecutar las secuencias en modo inverso.

Secuencia Normal:

Esta es la secuencia más usada y la que generalmente recomienda el fabricante. Con esta secuencia el motor avanza un paso por vez y debido a que siempre hay al menos dos bobinas activadas, se obtiene un alto torque de paso y de retención.

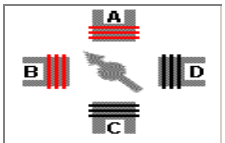
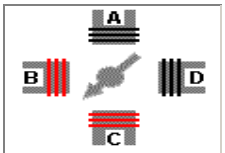
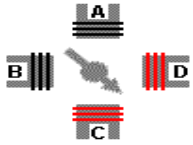
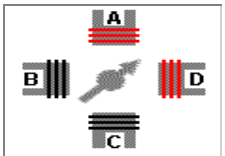
PASO	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

Tabla 10.- Secuencia Normal para manejar el motor

Cabe destacar que debido a que los motores paso a paso son dispositivos mecánicos y como tal deben vencer ciertas inercias, el tiempo de duración y la frecuencia de los pulsos aplicados es un punto muy importante a tener en cuenta. En tal sentido el motor debe alcanzar el paso antes que la próxima secuencia de pulsos comience.

2.1.1.3 CIRCUITO DE I/O DEL PIC16F877A

En este circuito apreciaremos el microcontrolador con todas las entradas y salidas cableadas, conjuntamente con el modulo LCD.

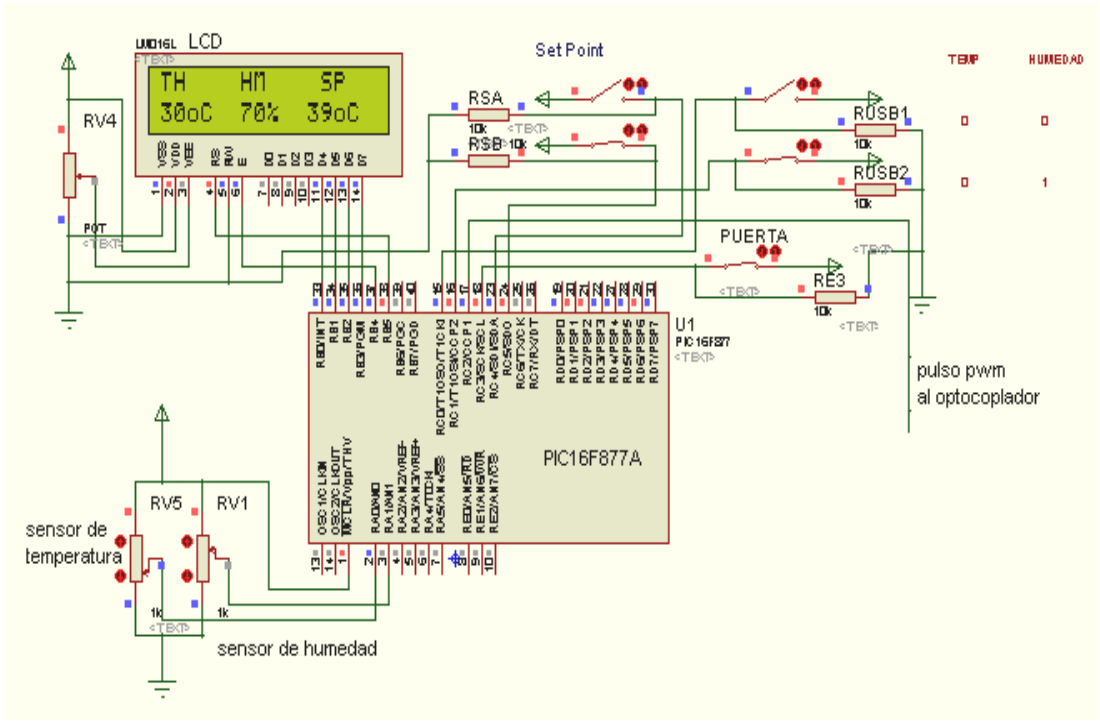


Figura 24.- Microcontrolador PIC16F8774A

Identificación de los pines de conexión de un modulo LCD

Figura 25.- Pantalla de Cristal Liquido

Los pines de conexión de un modulo LCD han sido estandarizados por lo cual en la mayoría de ellos son exactamente iguales siempre y cuando la línea de caracteres no sobrepase los ochenta caracteres por línea. Por otro lado es de suma importancia localizar exactamente cual es el pin Numero 1 ya que en algunos módulos se encuentra hacia la izquierda y en otros módulos se encuentra a la derecha.

Pin	Simbología	Nivel	I/O	Función
1	VSS	-	-	0 Vlts. Tierra (GND).
2	VCC	-	-	+ 5 Vlts. DC.
3	Vee = Vc	-	-	Ajuste del Contraste.
4	RS	0/1	I	0= Escribir en el modulo LCD. 1= Leer del modulo LCD
5	R/W	0/1	I	0= Entrada de una Instrucción. 1= Entrada de un dato.
6	E	1	I	Habilitación del modulo LCD
7	DB0	0/1	I/O	BUS DE DATO LINEA 1 (LSB).
8	DB1	0/1	I/O	BUS DE DATO LINEA 2
9	DB2	0/1	I/O	BUS DE DATO LINEA 3
10	DB3	0/1	I/O	BUS DE DATO LINEA 4
11	DB4	0/1	I/O	BUS DE DATO LINEA 5
12	DB5	0/1	I/O	BUS DE DATO LINEA 6
13	DB6	0/1	I/O	BUS DE DATO LINEA 7
14	DB7	0/1	I/O	BUS DE DATO LINEA 8 (MSB).
15	A	-	-	LED (+) Back Light
16	K	-	-	LED (-) Back Light.

Tabla 11.- Especificaciones de los pines del LCD

2.1.2 CIRCUITO DE TRANSMISIÓN USB DE DATOS INCUBADORA-COMPUTADORA.

En nuestro proyecto hemos utilizado comunicación USB para la transmisión de datos, donde la comunicación de divide en capas, como la interfaz física donde hablaremos de el Concentrador o HUB. En nuestro caso es el CYPRESS CY7C63000A

2.1.2.1 CONCENTRADOR DEL USB (HUB): PIC CYPRESS CY7C63000A

Los Chips basados en Cypress CYC63001A y el CY7C63101A proveen una solución para comunicación USB a bajo costo, entre estos tenemos el 802600, 802300,802200 que son chips USB I/O.

El 802600 y el 802200 son programados para aceptar un surtido conjunto de comandos. Los Chips conforman al Standard USB 1.1. El chips USB acepta un resonador cerámico de 6 Mhz, e internamente se duplica a 12MHz.

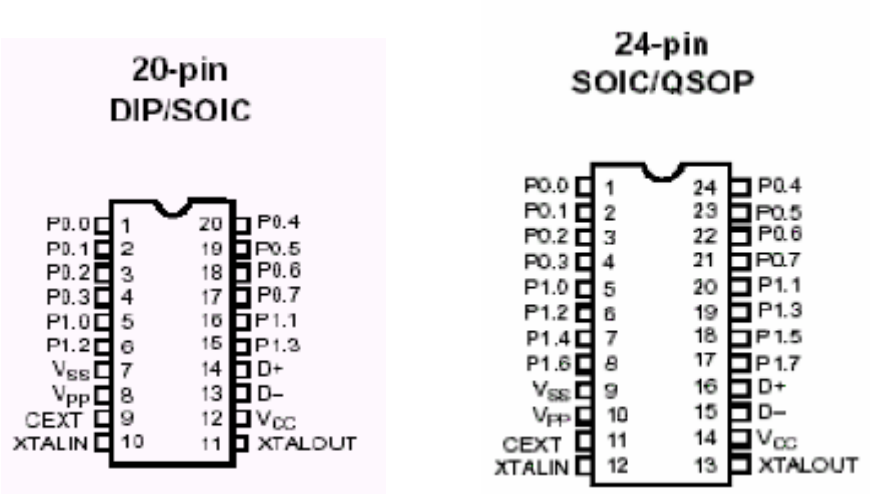


Figura 26.- Chips de 20-24 pines para comunicación USB.

En la figura podemos apreciar dos diferentes chips:

802600(24-pin): 16 I/O (entrada/salidas)

803300(20-pin): 12 I/O (entrada/salidas)

Ambos chips tienen 8 pines de lógica baja en el puerto0 y de 4 a 8 pines de lógica alta en el puerto1.

En la siguiente tabla podemos apreciar la descripción y definición de cada uno de los pines basados en CYC63001A y el CY7C63101A

Name	Description
Vcc	Voltage Supply. Nominal 5V, Range 4.0Volts to 5.25Volts
Vss	Ground. Connect to ground
XtalIn	Clock Input
XtalOut	Clock Output
P0.0-7	Port 0. Low Current GPIO. Programmable sink current & pullup.
P1.0-7	Port 1. High Current GPIO. Programmable sink current & pullup.
D+,D-	USB data lines. Requires an external 7.5K resistor connected to D- to Vcc.
Vpp, Cext	Unused pins. Vpp connect to ground. Cext leave open.

Tabla 12.- Nomenclatura del CYC63100

Name	I/O	802200 802300 20-Pin	802600 24-Pin	Description
P0.0	I/O	1	1	Port 0 bit 0
P0.1	I/O	2	2	Port 0 bit 1
P0.2	I/O	3	3	Port 0 bit 2
P0.3	I/O	4	4	Port 0 bit 3
P0.4	I/O	20	24	Port 0 bit 4
P0.5	I/O	19	23	Port 0 bit 5
P0.6	I/O	18	22	Port 0 bit 6
P0.7	I/O	17	21	Port 0 bit 7
P1.0	I/O	5	5	Port 1 bit 0
P1.1	I/O	16	20	Port 1 bit 1
P1.2	I/O	6	6	Port 1 bit 2
P1.3	I/O	15	19	Port 1 bit 3
P1.4	I/O	-	7	Port 1 bit 4
P1.5	I/O	-	18	Port 1 bit 5
P1.6	I/O	-	8	Port 1 bit 6
P1.7	I/O	-	17	Port 1 bit 7
XTALIN	I	10	12	Clock In
XTALOUT	O	11	13	Clock Out
CEXT	I/O	9	11	Wake Up Pin
D+	I/O	14	16	USB Data +
D-	I/O	13	15	USB Data -
Vpp	-	8	10	Programming voltage, Connect to Vss
Vcc	-	12	14	Voltage Supply
Vss	-	7	9	Ground

Tabla 13.- Descripción de los pines del CYC63100

El chips provee comandos de entrada / salida de 8 bits y comandos individuales para setear y resetear cada uno de los pines

Write Strobe: permite comunicarse al chips USB I/O con otro dispositivo usando como interface un bus de datos de 8 bits con un pin de strobe. El dato es ubicado en el puerto0 y el unos de los pin del puerto1 es el pin de strobe, este habilita la escritura de los Datos.

Clock generador: Este función genera una fuente reloj con frecuencia y periodo variable, habilitado para tener 4 configuraciones separadas de reloj. La salida del reloj pueden ser seleccionada del port1 pines 0-3.

Port Setup: esta característica permite setear la salida de corriente y habilita o deshabilita la resistencia de pull-up.

Read Buffer: Esta característica permite comunicarse con un dispositivo usando un estándar de un bus de datos de 8 bits. El dato es leido en el puerto0 con un pulso en el pin read strobe , que es seleccionado de los pines del puerto1.

Stratch pad: Permite al usuario escribir 8 bytes de información definida por el usuario en el dispositivo USB. Esta area puede ser usada para almacenar variables de usuario, estados u otra información.

Event Counter: Permite el conteo de los eventos en uno de los pines del puerto0. La resolución del contador es de 4 bytes.

Status led: Activa el pin 3 del puerto1 cuando existe actividad en el bus. Este se desactiva cuando el comando es procesado. Para visualizar la actividad de este pulso se necesito un circuito de sostenimiento.

64 bit Read/write commands : Estos comandos permiten al usuario leer o escribir 64 bits (8 bytes) de datos con un solo comando. Estos comandos requieren hardware extras.

2.1.2.2 CIRCUITO BASICO DE COMUNICACIÓN

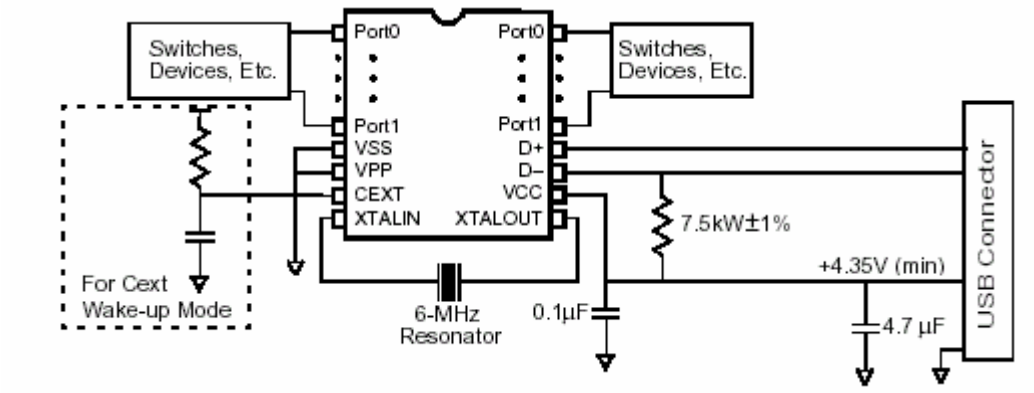


Figura 27.- Circuito de comunicación básica USB

Tenemos las especificaciones de los valores máximos de operación

Storage Temperature	-65C to +150C
Operating Temperature	-0C to +70C
Vss relative to Vcc	-0.5V to +7.0V
DC Input Voltage	-0.5V to Vcc+0.5V
DC voltage on HiZ pins	-0.5V to Vcc+0.5V
Max Current Summed on Port1 pins	60ma
Max Current Summed on Port0 pins	10ma
Power Dissipation	300mW
Static Discharge Voltage	>2000V
Latch Up Current	200mA

Y las especificaciones de las características eléctricas

Vcc Operating Current	25mA
Vcc Limits	4 to 5.25V
Port 0 Max Current Sink	1.5mA
Port 0 Min Current Sink	0.3mA
Port 1 Max Current Sink	24mA
Port 1 Min Current Sink	4.8mA
Pull Up Resistor	16Kohms
Input Hysteresis Voltages P0 &P1	Min6% Max12% Vcc
Bandwidth	3000 Cmd/sec*

2.1.2.3 CABLES Y CONECTORES QUE UTILIZA EL USB.

USB transfiere señales y energía a los periféricos utilizando un cable de 4 hilos, apantallado para transmisiones a 12 Mbps y no apantallado para transmisiones a 1.5 Mbps . En la figura 6 se muestra un esquema del cable, con dos conductores para alimentación y los otros dos para señal, debiendo estos últimos ser trenzados o no según la velocidad de transmisión.

Figura 28.- Esquema del cable para la comunicación USB

El calibre de los conductores destinados a alimentación de los periféricos varía desde 20 a 26 AWG, mientras que el de los conductores de señal es de 28 AWG. La longitud máxima de los cables es de 5 metros.

Por lo que respecta a los conectores hay que decir que son del tipo ficha (o conector) y receptáculo, y son de dos tipos: serie A y serie B . Los primeros presentan las cuatro patillas correspondientes a los cuatro conductores alineadas en un plano. El color recomendado es blanco sucio y los receptáculos se presentan en cuatro variantes: vertical, en ángulo recto, panel y apilado en ángulo recto. Se emplean en aquellos dispositivos en los que el cable externo, está permanentemente unido a los mismos, tales como teclados, ratones, y hubs o concentradores.

Los conectores de la serie B presentan los contactos distribuidos en dos planos paralelos, dos en cada plano, y se emplean en los dispositivos que deban tener un receptáculo al que poder conectar un cable USB. Por ejemplo impresoras, scanner, y módems.

2.2 FIRMWARE DEL MICROCONTROLADOR

Se conoce como firmware del microcontrolador al programa que se ejecuta dentro del integrado. Este programa es desarrollado en PICBASIC PRO y simulado en PROTEUS. Para la realización del programa se debe tener conocimiento de las instrucciones básicas de programación de microcontroladores. A continuación se detalla la clasificación de las instrucciones y configuraciones para la Conversión analógica y Modulación de ancho de pulso. Para un mayor entendimiento puede revisar el anexo .

2.2.1 INSTRUCCIONES

@ : Insert one line of assembly language code.

ADCIN: Read on-chip analogue to digital converter.

ASM..ENDASM: Insert assembly language code section.

BRANCH: Computed GOTO (equivalent to ON..GOTO).

BRANCHL: BRANCH out of page (long BRANCH).

BUTTON: Debounce and auto-repeat input on specified pin.

CALL: Call assembly language subroutine.

CLEAR: Zero all variables.

CLEARWDT: Clear (tickle) Watchdog Timer.
COUNT: Count number of pulses on a pin.

DATA: Define initial contents of on-chip EEPROM.

DEBUG: Asynchronous serial output to fixed pin and baud.

DEBUGIN: Asynchronous serial input from fixed pin and baud.

DISABLE: Disable ON DEBUG and ON INTERRUPT processing.

DISABLE DEBUG: Disable ON DEBUG processing.

DISABLE INTERRUPT: Disable ON INTERRUPT processing.

DTMFOUT: Produce touch-tones on a pin.

EEPROM: Define initial contents of on-chip EEPROM.

ENABLE: Enable ON DEBUG and ON INTERRUPT processing.

ENABLE DEBUG: Enable ON DEBUG processing.

ENABLE INTERRUPT: Disable ON INTERRUPT processing.

END: Stop execution and enter low power mode.

FOR..NEXT: Repeatedly execute statements.

FREQOUT: Produce up to 2 frequencies on a pin.

GOSUB: Call BASIC subroutine at specified label.

GOTO: Continue execution at specified label.

HIGH: Make pin output high.

HPWM: Output hardware pulse width modulated pulse train.

IF..THEN..ELSE..ENDIF: Conditionally execute statements.

INPUT: Make pin an input.

LCDIN: Read from LCD RAM.

LCDOUT: Display characters on LCD.

{LET}: Assign result of an expression to a variable.

LOW: Make pin output low.

NAP: Power down processor for short period of time.

OWIN: One-wire input.

OWOUT: One-wire output.

OUTPUT: Make pin an output.

PAUSE: Delay (1ms resolution).

PAUSEUS: Delay (1us resolution).

PEEK: Read byte from register.

POKE: Write byte to register.

POT: Read potentiometer on specified pin.

PULSIN: Measure pulse width on a pin.

PULSOUT: Generate pulse to a pin.

PWM: Generate pulse to a pin.

RANDOM: Generate pseudo-random number.

RCTIME: Measure pulse width on a pin.

READ: Read byte from on-chip EEPROM.

READCODE: Read word from code memory.

RETURN: Continue at statement following last GOSUB.

REVERSE: Make output pin an input or an input pin an output.

SELECT CASE: Compare a variable with different values.

SERIN: Asynchronous serial input (BS1 style).

SERIN2: Asynchronous serial input (BS2 style).

SEROUT: Asynchronous serial input (BS2 style).

SEROUT2: Asynchronous serial output (BS2 style).

SHIFTIN: Synchronous serial input.

SHIFTOUT: Synchronous serial output.

SLEEP: Power down processor for a period of time.

SOUND: Generate tone or white-noise on specified pin.

STOP: Stop program execution.
SWAP: Exchange the values of two variables.
TOGGLE: Make pin output and toggle state.
USBIN: USB input.
USBINIT: Initialize USB.
USBOUT: USB output.
WHILE..WEND: Execute statements while condition is true.
WRITE: Write byte to on-chip EEPROM.
WRITECODE: Write word to code memory.
XIN: X-10 input.
XOUT: X-10 output.

2.2.2 CONVERSIÓN ANALÓGICA

Los microcontrolador PIC16F877A poseen un conversor A/D de 10 bits de resolucion y 5 canales de entrada en los modelos con 28 patita y 8 canales en los que tienen 40 patitas.

La resolución que tiene cada bit procedente de la conversión tiene un valor que es función de la tensión de referencia Vref, de acuerdo con la formula siguiente:

Resolución = (Vref+-Vref-)/1.024= Vref/1.024

Si el voltaje Vref+=5V y la Vref- es tierra, la resolución es de 4,8mv/bit. Por tanto, a la entrada analógica de 0V le corresponde una digital de 00 0000 0000 y para la de 5V una de 11 1111 1111. La tensión de referencia determina los límites máximo y mínimo de la tensión analógica que se puede convertir. El voltaje diferencial es de 2V.

A través del canal de entrada seleccionado, se aplica la señal analógica a un condensador de captura y mantenimiento (simple and hola) y luego se

introduce al conversor, el cual proporciona un resultado digital de 10 bits de longitud usando la técnica de aproximaciones sucesivas.

El conversor A/D es el único dispositivo que puede funcionar en modo reposo (SLEEP), para ello el reloj del conversor deberá conectarse al oscilador RC interno.

La tensión de referencia puede implementarse con la tensión interna de alimentación VDD, o bien, con una externa que se introduce por la patita RA3/AN3/Vref+, en suyo caso la polaridad negativa se aplica por la patita RA2/AN2/Vref-.

Para la conversión se definen los siguientes parámetros en el PBP:

Define ADC_BITS 8 : Fija numero de BITS del resultado (5,8,10)

Define ADC_CLOCK 3 : Fija EL CLOCK (rc=3)

Define ADC_SAMPLEUS 50 : Fija el tiempo de muestreo en Us.

ADC_SAMPLEUS, es el numero de microsegundos que el programa espera entre fijar el canal y comenzar la conversión analogica/digital.

Para la activacion de canales utilizamos la siguiente instrucción:

ADCON0= %1000001 : activa canal 0 a Fosc/8

ADCON==%1001001 : activa canal 1 a Fosc/8

ADCIN 0,dato : leer el canal 0 (A0) y guardarlo en Dato.

2.2.3 MODULACIÓN DE ANCHO DE PULSO

El PWM(Pulse width Modulation) o modulacion en ancho del pulso, tiene muchas aplicaciones. Cada ciclo de PWM está compuesto de 256 pasos
El ciclo útil para cada ciclo varía de 0 (0%) a 255 (100%).
La forma de la señal que sale por el PIC es de acuerdo a las diferentes instrucciones:

PWM portb.0, 228,3 genera 3 ciclos al 90% alto y 10% bajo
PWM portb.0, 127,3 genera 3 ciclos al 50% alto y 50% bajo
PWM portb.0, 25,3 genera 3 ciclos al 10% alto y 90% bajo

2.2.4 MANEJO DE LA LCD.

Los Lcd se puede conectar con el PIC con un bus de 4 u 8 bits, la diferencia está en el tiempo que se demora, pues la comunicación a 4 bits, primero envía los 4 bits más altos y luego los 4 bits más bajos, mientras que la de 8 bits envía todo al mismo tiempo, esto no es un inconveniente si consideramos que el LCD trabaja en microsegundos.

La configuración de los pines del PIC hacia el LCD, los podemos definir de la siguiente manera:

```
DEFINE LCD_DREG PORTB   ; define pines del LCD B4 a B7
DEFINE LCD_DBIT 4   ; empezando desde el Puerto B4 hasta el B7
DEFINE LCD_RSREG PORTB   ; define pin para conectar el bit Rs
DEFINE LCD_RSBIT 3    ; en el Puerto B3
DEFINE LCD_EREG PORTB;  define pin para conectar el bit enable
DEFINE LCD_EBIT 2 ; en el Puerto B2
```

Una vez que se define la nueva configuración de pines para el LCD, programamos de la misma forma que las ocasiones anteriores, es importante ademas saber que los 4 bits de datos solo se pueden configurar en los 4 bits más bajos (B0 al B3) o los 4 bits más altos (B4 al B7) de un puerto del PIC, y si deseamos hacer una comunicación a 8 bits con el LCD, estos deben estar en un solo puerto, ademas debemos definir en el PBP que vamos a utilizar un bus de 8 bits, esto es de la siguiente manera:

```
DEFINE LCD_BITS 8   ; define comunicación a 8 bits con el LCD
```

Y si nuestro LCD posee 4 líneas, también debemos definirlo de la siguiente forma:

DEFINE LCD_LINES 4 ; define un lcd de 4 lineas

2.3 INTERFAZ LÓGICA DE LA COMUNICACIÓN USB

Nuestro proyecto tiene la segunda etapa que es la comunicación USB, este disposito llamado CY7C63100 tiene un firmware el cual permite la transmisión de datos Incubadora ⇔ Computadora y tiene un software el cual permite que a través de nuestra programación en Visual Basic podamos realizar las entradas de las variables externas y ejecutar acciones hacia la incubadora.

2.3.1 DISPOSITIVO LÓGICO USB: FIRMWARE

El Firmware del dispositivo esta elaborado en Assembler, el objetivo final del programa es pasar los datos recibidos en paralelos en las entradas (P0.1, P0.2, P03, P04, P05, P06, P07) para luego ser enviados en serie a través de D+ y D- (pines 15 y 16) hacia la computadora

Así mismo los datos enviados desde la Computadora vienen en serie a través del D+ y D- , el firmware los pasa en datos paralelo hacia las salidas (P1.0, P1.1, P1.2, P1.3, P1.4, P1.5, P1.6, P1.7) para ser llevados al PIC16F877A.

Para empezar a programar nuestro Software de sistema USB, debemos tomar en cuenta que al colocar en el puerto USB nuestra placa electrónica, inmediatamente el Sistema Operativo reconoce que existe un dispositivo nuevo. Luego nosotros procedemos a instalar el archivo de instalación para el USB driver y el driver del USB cuyo nombre son respectivamente USBIODS.INF y USBIODS.SYS

Requerimientos:

- Poseer el dispositivo y los driver
- Tener Sistema Operativo Win 98,2000, Me, Xp
- Visual Basic mínimo versión 4.0

2.3.2 SOFTWARE DEL SISTEMA USB EN LA COMPUTADORA

Nuestro proyecto es manejado desde el software Servidor. Dentro de este existen funciones y subrutinas que permiten cada 100 ms pedir datos de la incubadora. Además hemos incluido un modulo donde definimos las funciones que utiliza el dispositivo USB para abrirlo, leer, escribir y cerrarlo. Podemos leer los manuales de Delcom Dll en los anexos.

Las funciones están divididas en tres grupos:

- Funciones comunes
- Indicadores Visuales
- Funciones de USB I/O

La programación puede ser en C o Visual Basic.

A continuación detallaremos las funciones principales para realizar la comunicación Computadora – USB.

Función para Abrir el dispositivo

Esta función lee el nombre del dispositivo y luego abre el dispositivo y envía un aviso en una variable global hdevice . Retorna 0 si existe algun error. Y tambien nos devuelve el nombre completo del device en DEviceName.

```
Function OpenDevice() As Boolean
On Error GoTo ERROR_HANDLER
```

Obtiene el nombre del dispositivo:

```
lpDeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _

"System\CurrentControlSet\Services\Delcom\USBIODS\Parameters\", _
                    "DeviceName", "")
If lpDeviceName = "" Then   ' exit on error
   MsgBox "Unable to open device, check connection and power."
   lpDeviceName = "Device Not Found!"
   OpenDevice = False
   Exit Function
End If
```

Trata de abrir el dispositivo. Este fallara si el dispositivo no esta presente

```
hDevice = CreateFile(lpDeviceName, GENERIC_READ Or
GENERIC_WRITE, _
                FILE_SHARE_WRITE Or FILE_SHARE_READ, 0, _
                OPEN_EXISTING, 0, 0)
If hDevice <= 0 Then    ' check for error
   MsgBox "Unable to open device, check connection and power"
   lpDeviceName = "Device Not Found!"
   OpenDevice = False
Else
   OpenDevice = True
End If
Exit Function
ERROR_HANDLER:
   MsgBox "OpenDevice() ERROR #" & Str$(Err) & " : " & Error
End Function
```

Función para cerrar el dispositivo

Esta función cierra el dispositivo después de haberlo usado.
Si no cerramos el dispositivo después de usarlo, no podremos abrirlo nuevamente sin sacar el cable USB. Para cerrar el dispositivo este verifica hdevice, ya que este nos indica si el dispositivo esta abierto.

```
Function CloseDevice() As Boolean
On Error GoTo ERROR_HANDLER
CloseDevice = CloseHandle(hDevice)  ' Close the device
hDevice = 0                         ' Null the handle
If CloseDevice = False Then         ' Check for errors
  MsgBox "Error closing file"     ' Display errors
End If
Exit Function
ERROR_HANDLER:
   MsgBox "CloseDevice() ERROR #" & Str$(Err) & " : " & Error
End Function
```

Función para enviar y recibir paquetes al dispositivo

Esta función recibe y envía paquetes desde el dispositivo USB. El dispositivo USB debe estar abierto. El primer parámetro es el hdevice, el segundo parámetro es el paquete a enviar al dispositivo USB y el último parámetro es el paquete a recibir por parte del dispositivo USB. El paquete recibido es únicamente requerido cuando el comando es de lectura, de lo contrario el parámetro es cero. El paquete enviado puede ser 8 a 16 bytes de longitud. El paquete recibido es siempre 8 bytes de longitud.

```
VB: Public Declare Function DelcomSendPacket Lib "DelcomDLL.dll" _
(ByVal DeviceHandle As Long, ByRef PacketOut As PacketStructure, ByRef
PacketIn As PacketStructure) As Long
PacketStruct Packet;
char DeviceName[MaxDeviceLen];
if(!DelcomGetNthDevice(USBIODS, 0, DeviceName)) return(0);
```

```
HANDLE hUsb = DelcomOpenDevice((char*)DeviceName,0);
```

Escribir un paquete:

```
Packet.Recipient = 8          ; // always 8
Packet.DeviceModel = 18   ; // always 18
Packet.MajorCmd = 10
Packet.MinorCmd = 10       ; // escribe  port0 & port1
Packet.DataLSB = 0xFF      ; // set port0 to all high
Packet.DataMSB = 0x00      ; // set port1 to all low
Packet.Length = 0              ; // DataExt not used
DelcomSendPacket(hUsb,&Packet,NULL)
```

Leer un paquete:

```
Packet.Recipient = 8            ; // always 8
Packet.DeviceModel = 18     ; // always 18
Packet.MajorCmd = 11;
Packet.MinorCmd = 0          ; // read port0 & port1
Packet.Length = 0               ; // DataExt not used
DelcomSendPacket(hUsb,&Packet,&Packet);
printf("Port0=%X Port1=%X\n",((char*)&Packet)[0],((char*)&Packet)[1]);
DelcomCloseDevice(hUsb)    ; // close the device
return(0);
```

Variables globales

Variable que nos dirá si el dispositivo fue abierto

Public hDevice As Long

Variable donde estará el nombre del dispositivo para abrirlo

Public lpDeviceName As String’

Paquetes enviados y recibidos a la Incubadora

En el software Servidor seteamos un timer de 100ms de tal forma que este enviando y recibiendo paquetes al Dispositivo. Nosotros enviamos códigos que van al PIC, donde el PIC compara los códigos asignados a la Temperatura, Humedad, Sensor de nivel y Puerta. Inmediatamente envía la variable pedida a la entrada del dispositivo esperando que hagamos una lectura.

Motor	Set Point	Temperatura	Humedad
1001 1100 0110 0011	00 01 10 11	00	01

Tabla 14.- Códigos enviados al microcontrolador

2.4 SOFTWARE DE COMUNICACIÓN VIA INTERNET

Para poder controlar el proceso de Incubación de Codornices vía Internet utilizamos La arquitectura Cliente-Servidor con el protocolo TCP/IP. Por lo tanto tenemos un software Servidor y un software cliente

2.4.1 PROPIEDADES DEL CONTROL DE COMUNICACIÓN WINSOCK

- **LocalIP**: Devuelve la dirección IP de la máquina local en el formato de cadena con puntos de dirección IP (xxx.xxx.xxx.xxx).
- **LocalHostName**: Devuelve el nombre de la máquina local.
- **RemoteHost**: Establece el equipo remoto al que se quiere solicitar la conexión.
- **LocalPort**: Establece el puerto que se quiere dejar a la escucha.
- **RemotePort**: Establece el número del puerto remoto al que se quiere conectar.
- **State**: Verifica si el Control WinSock esta siendo utilizado o no.

Lista de Métodos más importantes.

- **Accept**: Sólo para las aplicaciones de servidor TCP. Este método se utiliza para aceptar una conexión entrante cuando se está tratando un evento ConnectionRequest.
- **GetData**: Recupera el bloque actual de datos y lo almacena en una variable de tipo Variant.
- **Listen**: Crea un socket y lo establece a modo de escucha.
- **SendData**: Envía datos a un equipo remoto

Lista de Eventos más importantes.

- **ConnectionRequest**: Se produce cuando el equipo remoto solicita una conexión. Sin este evento no se puede llevar a cabo la conección.
- **Connect**: Se produce cuando el equipo local se conecta al equipo remoto y se establece una conexión.

- **Close**: Se produce cuando el equipo remoto cierra la conexión. Las aplicaciones deben usar el método Close para cerrar correctamente una conexión TCP.
- **DataArrival**: Se produce cuando llegan nuevos datos. Este evento es importante, ya que debemos hacer algo con la información que llega.

2.4.2 SOFTWARE SERVIDOR

Figura 29.- Pantalla del Software Servidor

Es el que recibe la información de nuestra Incubadora a través de la interfaz lógica USB . Y la trasmite a nuestro software cliente utilizando un control Winsock

El proceso del software Servidor tiene diferentes etapas importantes:

- Abre el dispositivo USB.
- Luego setea un reloj para pedir datos a la incubadora cada 100 ms por medio del Software del sistema USB,
- Inmediatamente coloca su dirección IP, Nombre de la maquina y comienza a esperar conexión (EsperarConexion ( )) de parte del cliente por medio del control Winsock que lo pone en estado de escuha (winsock1.listen) siendo el estado 0. (Winsock1.State = 0).

  Tenemos la función Ver estado, donde podemos analizar la conexión

```
Public Function VerEstado(Estado As Byte) As String
Select Case Estado
    Case 0
    VerEstado = "Sin Conexiones"
    MainForm.EsperarConexion
    With MainForm
     End With
    Case 1
    VerEstado = "Abierto"
   Case 2
   VerEstado = "Esperando Conexion"
   With MainForm
    End With
   Case 3
   VerEstado = "Conexión Pendiente"
   Case 4
   VerEstado = "Resolviendo Host"
   Case 5
   VerEstado = "Host Resuelto"
   Case 6
   VerEstado = "Conectando"
   Case 7
   VerEstado = "Conectado"
   With MainForm
   .LblIpLocal.Caption = "IP = " & .Winsock1.RemoteHostIP
   End With
```

```
        Case 8
          VerEstado = "Cerrando Conexion"
          Winsock1.Close
          Case 9
          VerEstado = "Error"
    End Select
    End Function
```

Dentro de esta función usaremos el caso 0 y 7 para otras acciones y para validar el sistema. Debido a que son estados de “Sin conexiones” y “Conectado”.

- Cada 100 msg muestra los datos recibidos de la incubadora por medio de etiquetas.
- En el instante que recibe un requerimiento de conexión (Winsock1_ConnectionRequest) por parte del software cliente, se coloca la dirección IP del cliente y el estado cambia a “Conectado” Case 7. Inmediatamente se crea un sockets y lo establece a modo de escucha winsock1.listen (espera requerimientos por parte del cliente). Estos requerimientos pueden ser para enviar datos o para recibir mensajes (Chat) .
- Todo pedido que llega produce el evento Winsock1_DataArrival, y a través del método Winsock1.GetData datos, obtenemos los datos y identificamos (Ident) si desea Humedad, Temperatura, giro o Chat.
- En caso de ser Humedad, Temperatura, giro , la información se la empaqueta enviándola dentro de esta variable

```
s = valor_temph.Caption & "&" & valor_humedad & "&" &
      velocidad.Caption & "&" & Trim(sp_th.Text)
If Ident = "DATA" Then Call Send(s)
If Ident = "CHAT" Then List1.AddItem datos
```

Y utilizamos la función Send(s), la cual contiene (Winsock1.SendData xDato), la cual envía datos a un equipo remoto.

- Y si identificamos que es CHAT, lo añadimos a una lista. El usuario del Servidor puede responder al mensaje escribiendo en una caja de texto y lo envía haciendo uso de la función Winsock1.SendData xDato

2.4.3 SOFTWARE CLIENTE

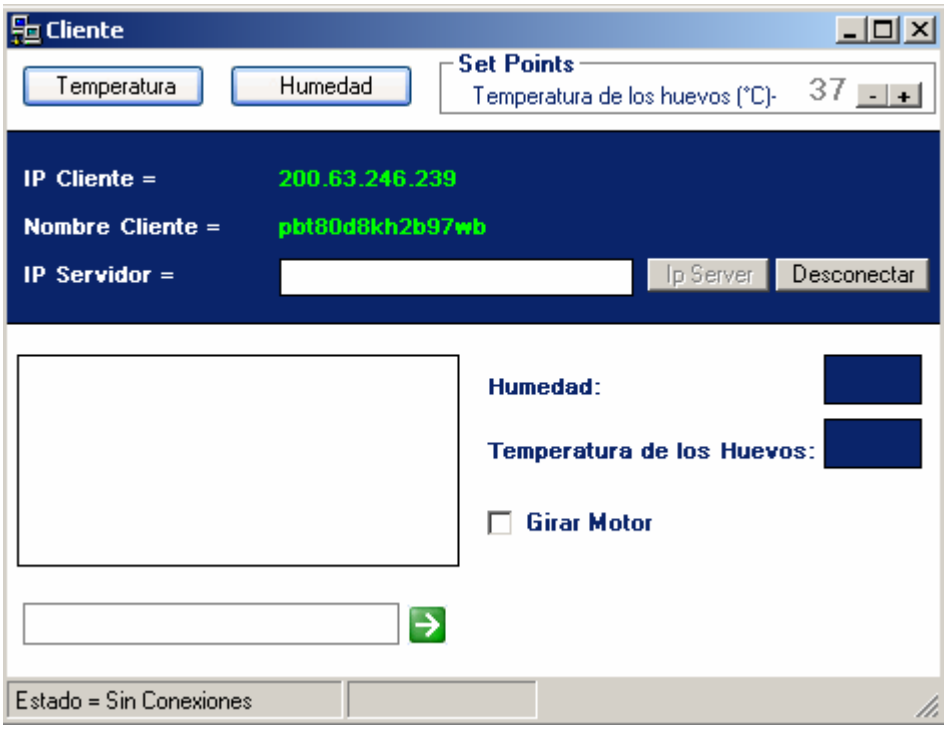


Figura 30.- Pantalla del Software Cliente

Es el que recibe la información del Servidor utilizando un control Winsock
El proceso del software Cliente tiene diferentes etapas importantes:

- El cliente inicia su operación en el Form_Activate, seteando su dirección IP

- Inmediatamente nosotros coloquemos la IP del servidor , dando clic en IP SERVER , se establece una conexión (WSocket.Connect)

Cuando se conecta produce un evento Wsockets_ConnectionRequest y cambia el estado a “Conectado” Case 7, siempre y cuando el Servidor este levantado.

```
Private Sub CmdConectar_Click()
WSocket.RemoteHost = TxtIpServidor.Text
WSocket.Connect
```

- Se activa el timer para hacer requerimientos de Datos (Temperatura, Humedad y giro) o Chat al servidor cada segundo, utilizando WSocket.SendData xDato
- Cuando llega la imformacion del Servidor produce el evento Data_arrival y por medio de la funcion WSocket.GetData sdatos obtenemos los datos y los desempaquetamos e identificamos si es DATO o CHAT los cuales son mostrados al usuario por medio de etiquetas.

```
WSocket.GetData sdatos
npos = Val(Mid(sdatos, 1, 1))
If npos <> 4 Then
    s = sdatos
    'MsgBox s
    columna = InStr(1, s, "&", 1)
    cad1 = Left(s, columna - 1)
    valor_temph = cad1
    'MsgBox cad1
    s = Trim(Right(s, Len(s) - columna))
    columna = InStr(1, s, "&", 1)
```

```
        cad2 = Left(s, columna - 1)
        'MsgBox cad2
        valor_humed.Caption = cad2
        s = Trim(Right(s, Len(s) - columna))
        columna = InStr(1, s, "&", 1)
        cad3 = Left(s, columna - 1)
        'MsgBox cad3
        velocidad.Caption = cad3
        cad4 = Trim(Right(s, Len(s) - columna))
        Text2.Text = cad4
    End If
    If npos = 4 Then
        ident = Trim(UCase(Mid(sdatos, 2, npos)))
        valor = Trim(Mid(sdatos, 6, 250))
        MsgBox ident
         MsgBox valor
        If ident = "CHAT" Then List1.AddItem sdatos
```

- Ademas, el cliente puede enviar mensajes al administrador del servidor utilizando WSocket.SendData xDato.

CAPITULO 3

3. DIAGRAMAS DE BLOQUE

Podemos apreciar en la figura, el diagrama del Control Remoto Vía Internet en su totalidad, donde podemos distinguir dos etapas importantes:

- Diagrama de bloque de un Cliente-Servidor
- Diagrama de bloque comunicación USB y Firmware del microcontrolador

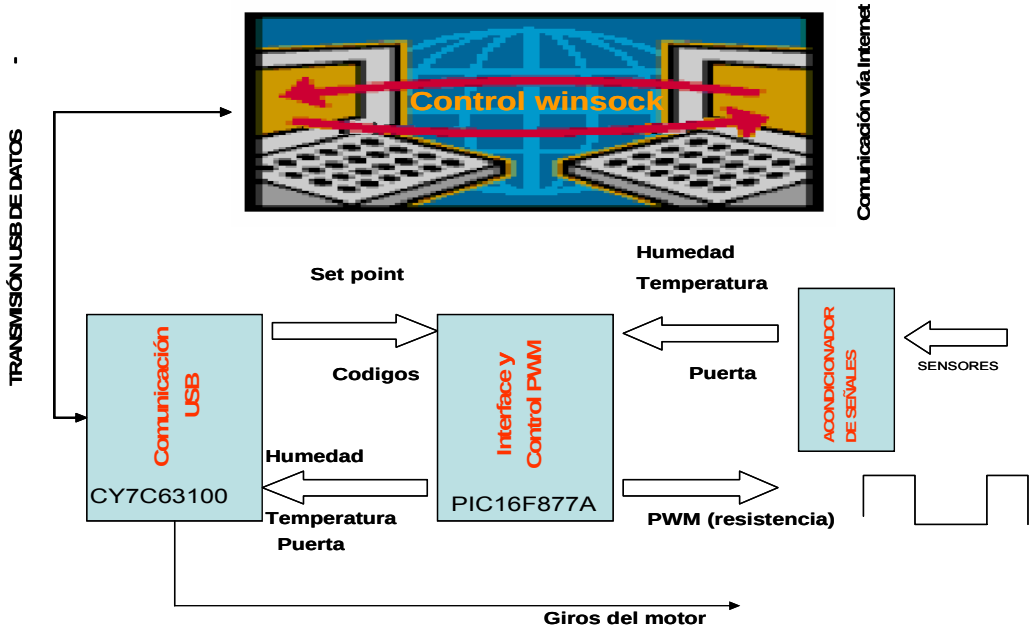


Figura 31.- Diagrama de Bloque General

3.1 DIAGRAMA DE BLOQUES CLIENTE-SERVIDOR

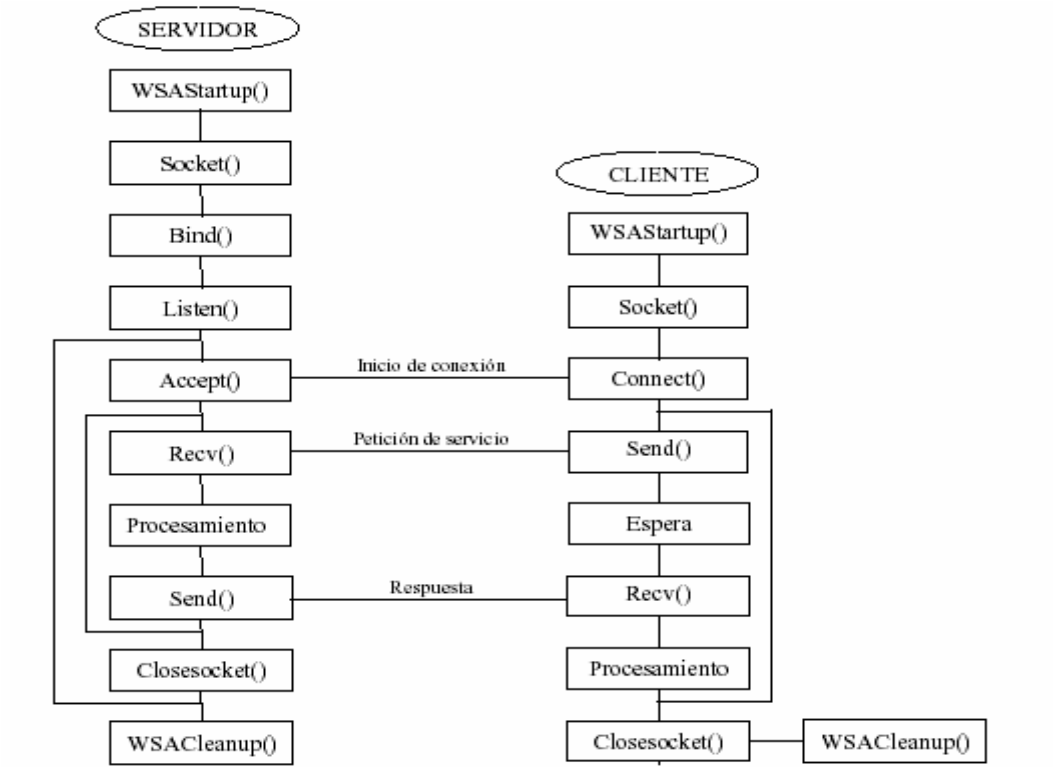


Figura 32.- Diagrama de Bloques Cliente-Servidor

Pasos seguidos por un Servidor en la realización de una conexión

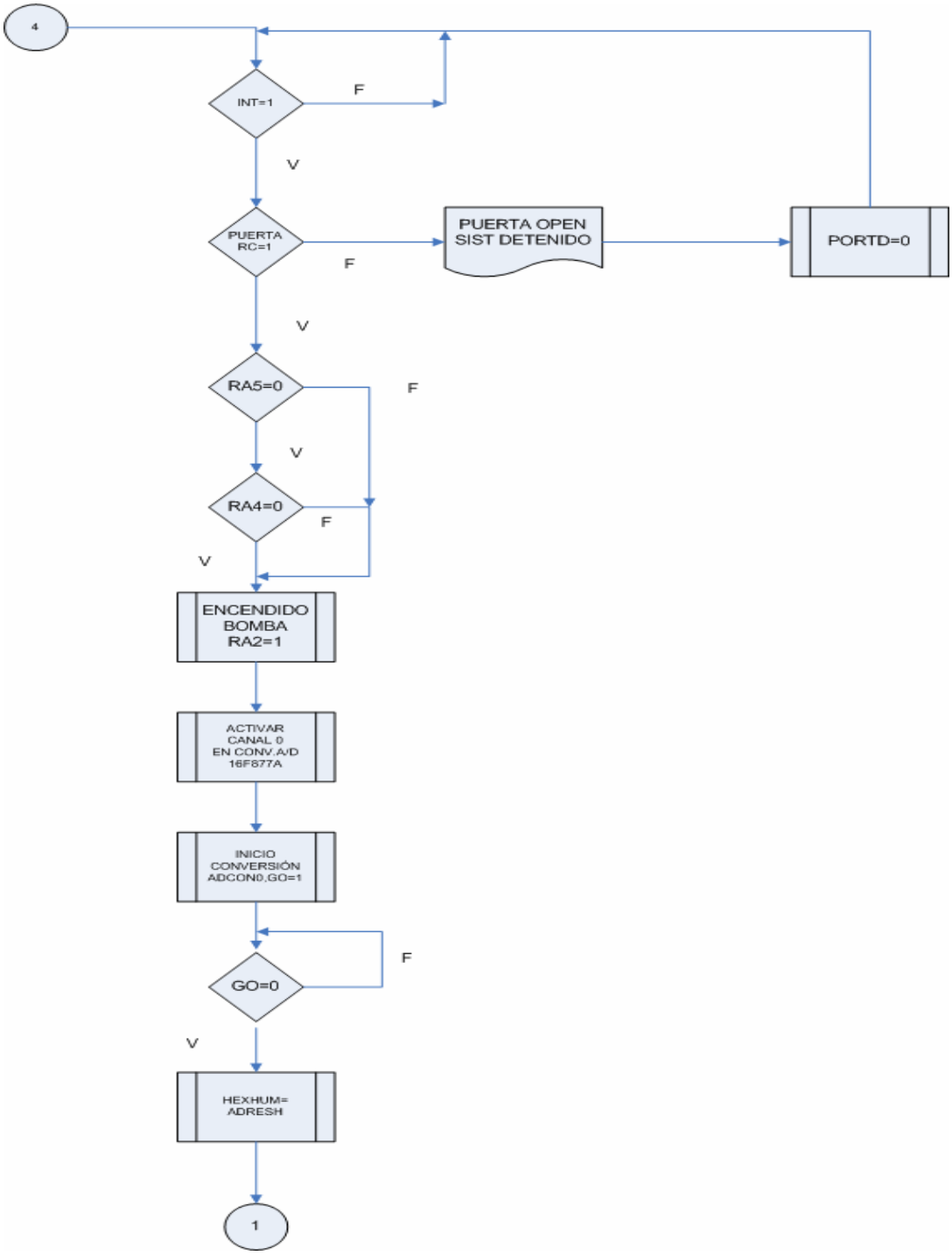
1. Inicializacion del sistema de sockets(solo en windows: WSAStartup()).
2. Creación del socket de escucha (socket()).
3. Vincualción del socket con una dirección local (blnd()).
4. Puesta del socket a la escucha y creación de la cola de peticiones (listen()).

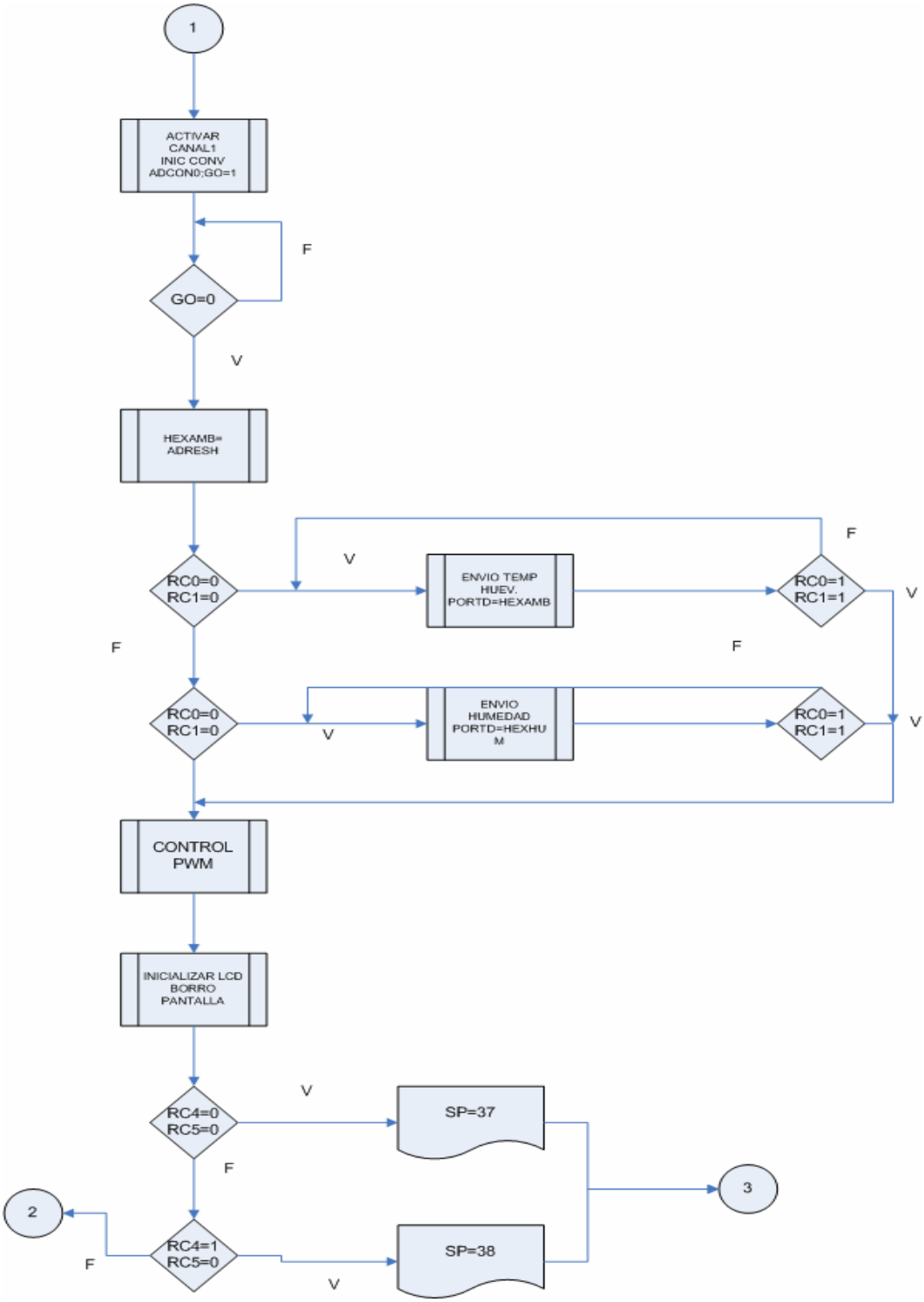
5. Aceptación de conexiones y creación del socket de servicio (accept()).
6. Lectura de la petición de recurso del cliente (recv() o write()).
7. Procesamiento de la petición
8. Envío de datos al cliente (send() o read()).
9. Vuelta al paso 6 si es necesario
10. Cierre del sockets de servicio (closesocket() o close()).
11. Vuelta al paso 4
12. Liberación del sistema de sockets(solo en windows: WSACleanup()).

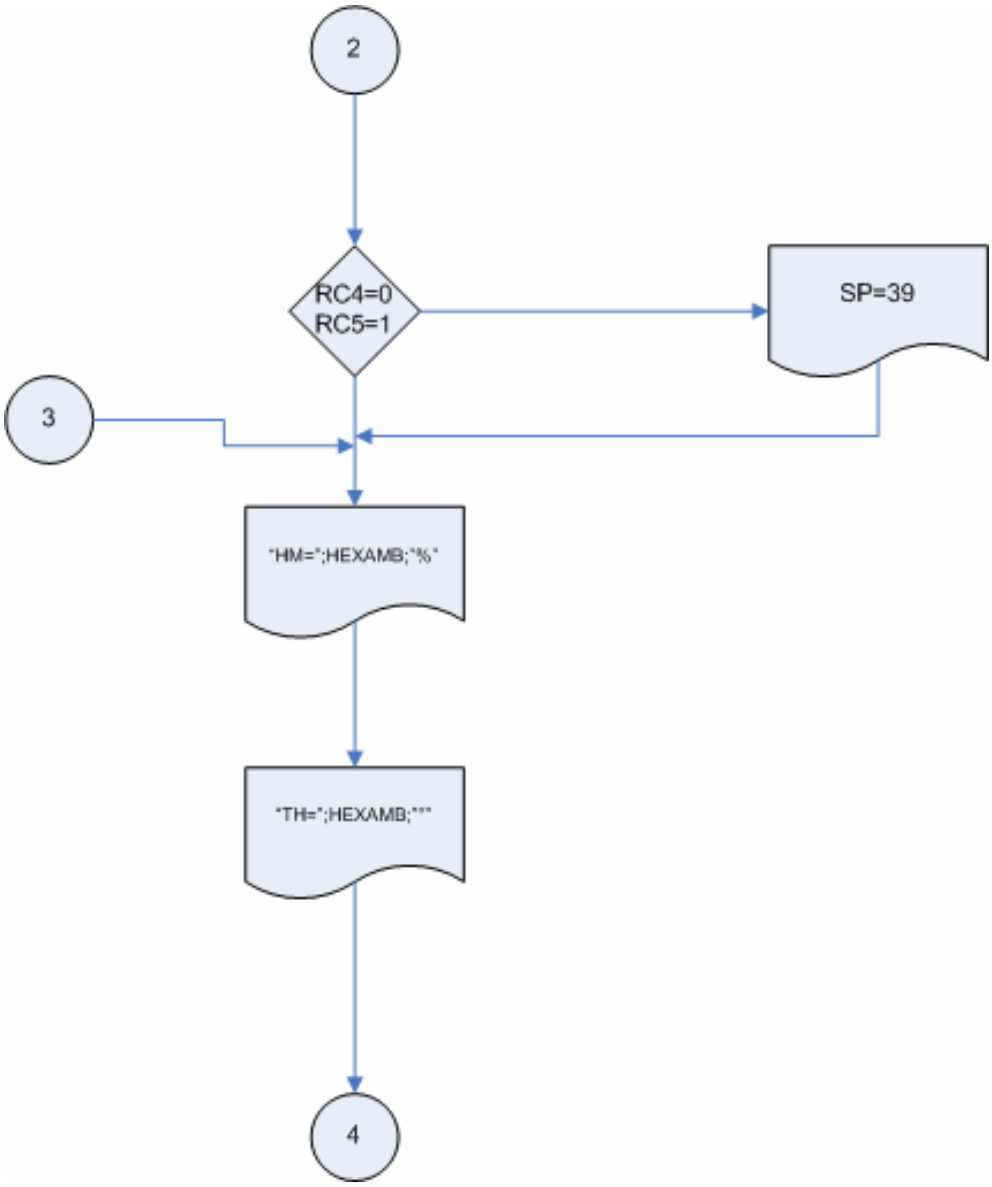
Pasos seguidos por un Cliente para la realización de una conexión

1. Inicializacion del sistema de sockets(solo en windows: WSAStartup()).
2. Creación del socket a conectar (socket()).
3. Conexión del socket con el Servidor.
4. Envió de petición de recursos al servidor (send() o read()).
5. Espera de la Respuesta del Servidor.
6. Recepción de datos del servidor.
7. Procesamiento de la imformacion
8. Vuelta al paso 6 si es necesario
9. Cierre del sockets (closesocket() o close()).
10. Liberación del sistema de sockets(solo en windows: WSACleanup()).

3.2 DIAGRAMA DE BLOQUES FIRMWARE DEL MICROCONTROLADOR Y COMUNICACIÓN USB.




1
ACTIVAR CANAL1 INIC CONV ADCON0;GO=1
F
GO=0
V
HEXAMB= ADRESH
RC0=0 RC1=0
V
ENVIO TEMP HUEV. PORTD=HEXAMB
RC0=1 RC1=1
F
V
F
RC0=0 RC1=0
V
ENVIO HUMEDAD PORTD=HEXHUM
RC0=1 RC1=1
V
CONTROL PWM
INICIALIZAR LCD BORRO PANTALLA
RC4=0 RC5=0
V
SP=37
F
2
RC4=1 RC5=0
F
V
SP=38
3


2
RC4=0
RC5=1
SP=39
3
"HM=";HEXAMB;"%"
"TH=";HEXAMB;""
4

3.3 ESTRATEGIA DE CONTROL

En nuestro estudio de controladores encontramos
Las formas estándar de controladores PID y el Control ON-OFF:

Proporcional	$K_P(s) = K_p$
Proporcional e Integral	$K_{PI}(s) = K_p\left(1 + \frac{1}{T_r s}\right)$
Proporcional y Derivativo	$K_{PD} = K_p\left(1 + \frac{T_d s}{\tau_d s + 1}\right)$
Proporcional, Integral y Derivativo	$K_{PID}(s) = K_p\left(1 + \frac{1}{T_r s} + \frac{T_d s}{\tau_d s + 1}\right)$

ANALISIS DEL CONTROLADOR:
Variable sensada: temperatura
Elemento a manejar: TRIAC
Procedimiento a actuar: Prender y Apagar el TRIAC a través de un optoacoplador para mantener una temperatura deseada en la cámara de incubación
De acuerdo a estos requerimientos, nosotros hemos escogido un control ON-OFF, debido a que nuestro elemento para generar calor es una resistencia manejada por un TRIAC, y no es un elemento proporcional que se abrirá o encenderá proporcionalmente , ya que si fuese el caso, utilizaríamos un control PI.

CONTROL ON-OFF

1. El control On-Off es la forma más simple de controlar.
2. Es comúnmente utilizado en la industria

3. Muestra muchos de los compromisos fundamentales inherentes a *todas* las soluciones de control.

Control en realimentación con ganancia elevada

- El control en realimentación con ganancia elevada posee ventajas.
- Un controlador On-Off es una forma sencilla de implementar un control en realimentación con alta ganancia.

Tenemos dos variables en nuestro proceso:

u(t): La entrada (variable manipulada) es el calor entregado por la resistencia de cerámica

y(t): La salida (variable de proceso) es la temperatura medida en la incubadora.

Tenemos el Sistema a lazo abierto, donde cualquier perturbación provocada causara un descenso de la temperatura en la cámara de incubación, aun cuando la calefacción este fija.

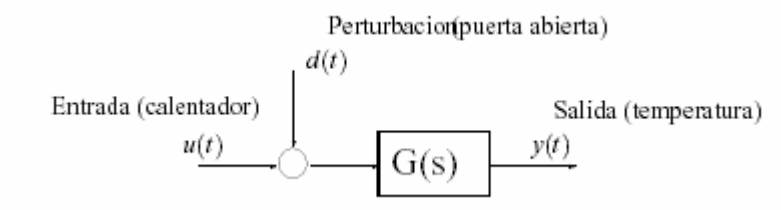


Figura 33.- Sistema de lazo abierto

Como podemos apreciar el controlador a Lazo Abierto es muy sensible a perturbaciones

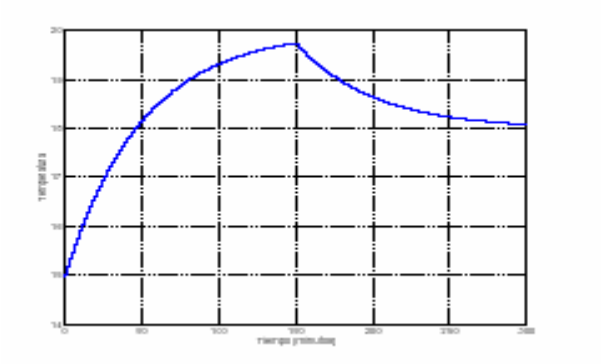


Figura 34.- Lazo abierto sensible a perturbaciones

Llevemos ahora al sistema a un Lazo Cerrado utilizando un controlador On-Off, como muestran las figuras

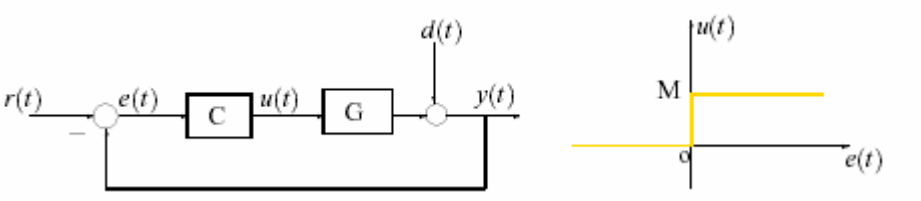


Figura 34.- Lazo cerrado utilizando control On-Off

- La respuesta se estabiliza en el valor deseado de la temperatura mucho
  mas rápido que cuando utilizamos el control a Lazo Abierto.
- La perturbación ahora solo afecta un poco a la respuesta.
- Una vez que la temperatura deseada es alcanzada el controlador continua variando entre On y Off rápidamente.

Nuestro Control ON-OFF , lo haremos a traves de una modulacion por ancho de pulso
Que se explico en el capitulo anterior.
Podemos apreciar el prendido y apagado del TRIAC en la siguiente simulación en Proteus.

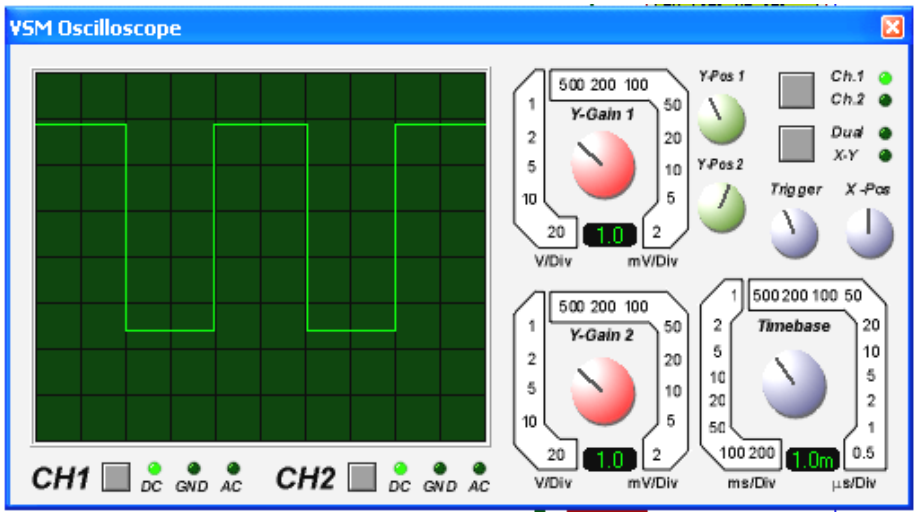


Figura 35.- Grafico del PWM a traves del VSM Oscilloscope

La temperatura y la humedad se podrá visualizar en la pantalla. En este caso podemos apreciar como la temperatura alcanza su set point.

Como sabemos el LM35 es lineal, de tal forma la temperatura va a crecer linealmente.

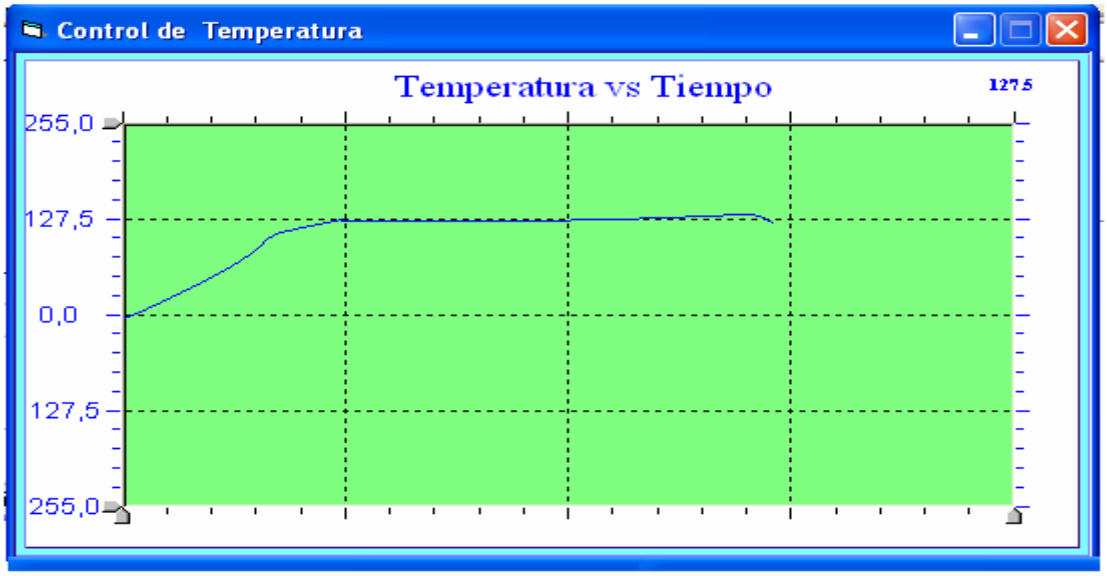


Figura 36.- Grafico de la Temperatura de Incubación vs Tiempo.

CAPITULO 4

4. IMPLEMENTACION Y COSTOS

4.1 Materiales y costos en la implementación del proyecto

Para el diseño y la implementacion de la Incubadora se utilizaron sensores tanto de humedad y temperatura, elaborando una tarjeta de adquisición de datos elaborada con los microcontoladores y ademas sus respectivas resistencias y capacitores.

Se describirá a continuación la cantidad de elementos utilizados y el costo del proyecto a implementar.

ITEM	CANTIDAD	DESCRIPCION	VALOR UNITARIO	VALOR TOTAL
1	2	resistencias de 1k 1/4 w	0,05	$ 0,10
2	9	resistencias de 10K	0,05	$ 0,45
3	2	resistencias de 0.39k	0,05	$ 0,10
4	2	resistencias de 0.25k	0,05	$ 0,10
5	1	resistencias de 0.22k	0,05	$ 0,05
6	1	resistencias de 7.5k	0,05	$ 0,05
7	1	Capacitores de 100 n	0,15	$ 0,15
8	2	Capacitores de 22pf	0,2	$ 0,40
9	4	Capacitores electrolíticos 0.1uF	0,15	$ 0,60
12	1	PIC16F877A	16	$ 16,00
13	1	LCD 2x16	10	$ 10,00
14	1	CYC7C63100 DELCOM	18	$ 18,00
15	1	USBDSC (CABLE)	2	$ 2,00
16	1	MOC3041(OPTOCOPLADOR)	1,5	$ 1,50
17	1	BT136500D(TRIAC)	0,75	$ 0,75
18	1	RES. CERAMICA 600 W	6,5	$ 6,50
19	1	VENTILADOR PEQUEÑO	1,5	$ 1,50
20	1	MOTOR DE PASO	30	$30
21	1	CRISTAL 4MHZ	2,5	$ 2,50
22	1	CRISTAL 6MHZ	2,5	$ 2,50
23	1	TRANSMISOR DE HUMEDAD	PRESTADO	
24	1	LM358	1	$ 0.50
25		PASTA Y SOLDADURA	5	$ 5,00
26		CAJA PARA INCUBADORA	80	$ 80,00
27		SENSOR LM35	1	$3.50
			TOTAL	$ 105.00

Tabla 15.- Costos en la implementación del proyecto

4.2 Esquemático General

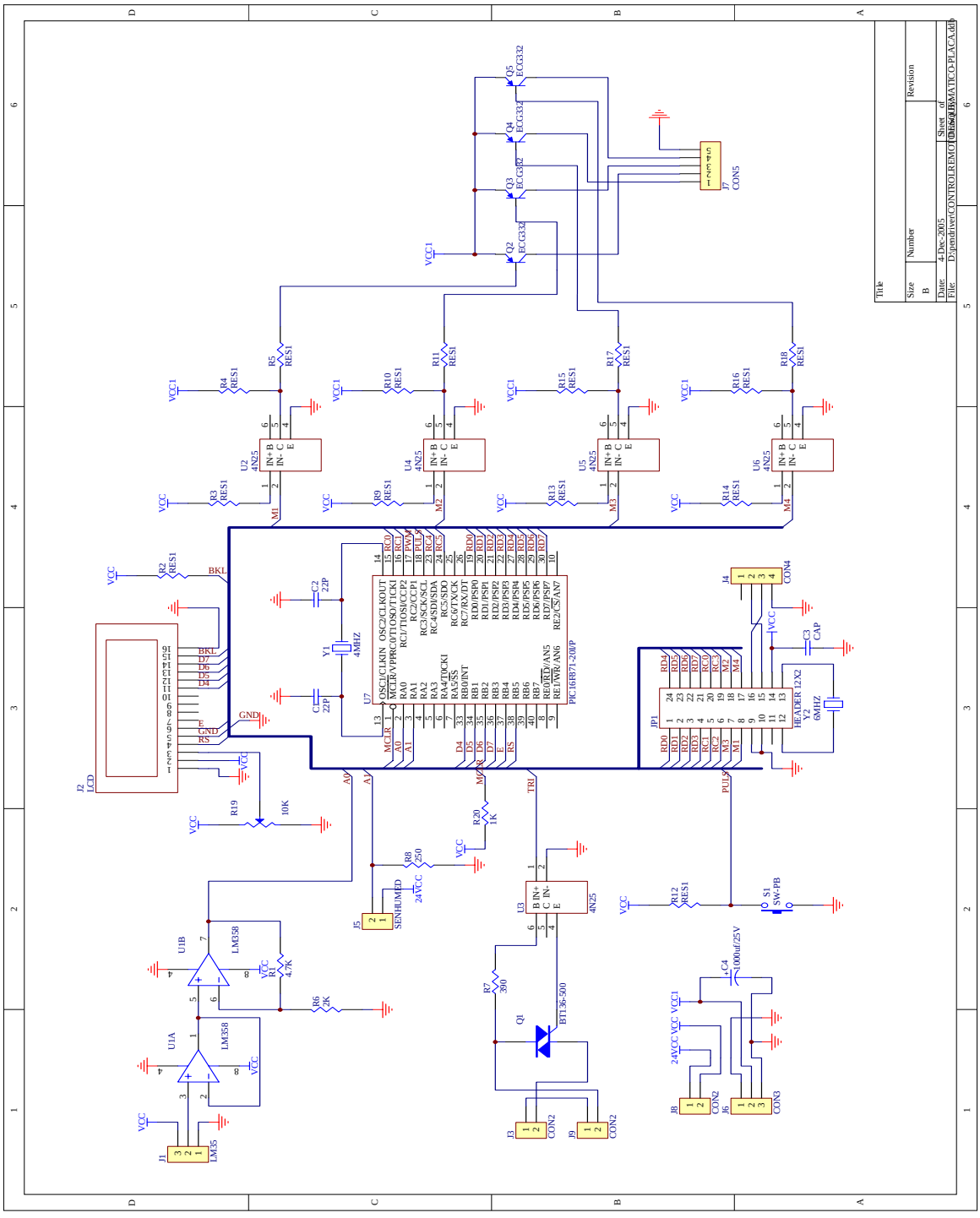


Figura 37.-Esquemático General

4.3 Diseño y Fotos de La Placa Electrónica

Tenemos dos pistas:

- Control General
- Control del Motor de Paso

Pistas del Control General

Podemos apreciar Las pistas de la Placa del control General

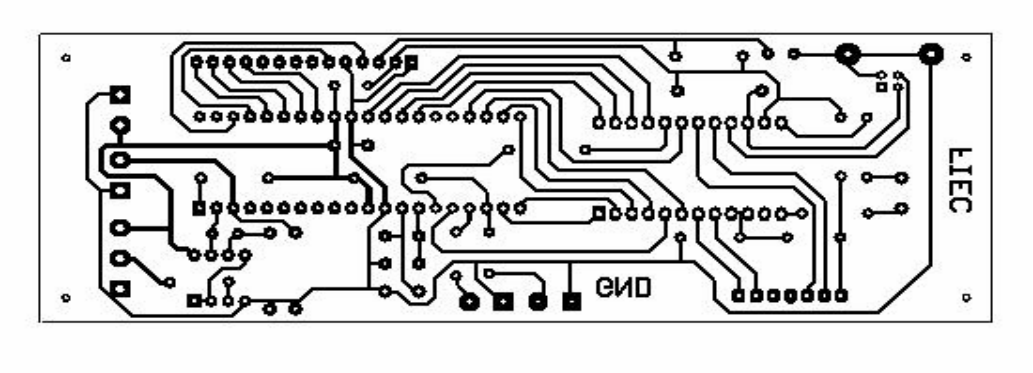


Figura 38.- Pista del Control General

Conjuntamente con la Placa de la Posición de los elementos, podemos apreciar las conexiones de los voltajes y sensores.

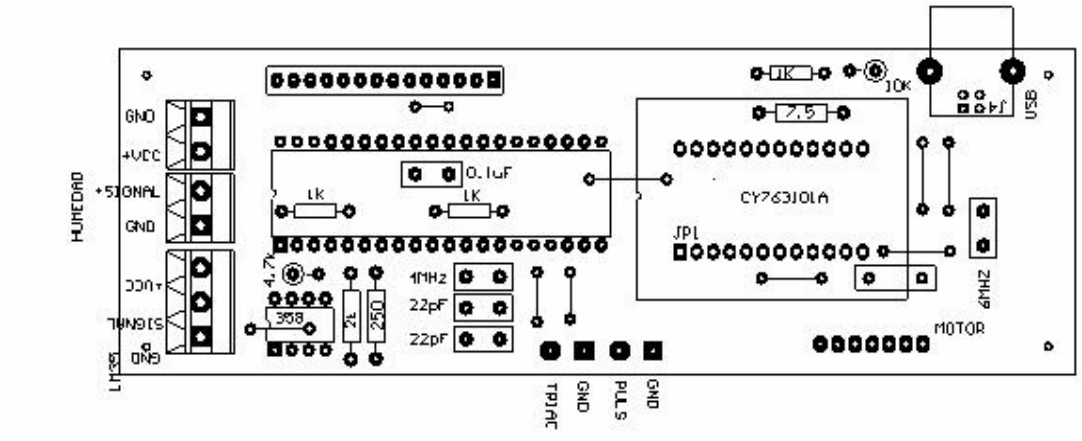


Figura 39.- Posición de los Elementos de la Tarjeta de Control

Podemos apreciar las fotos de la placa construida

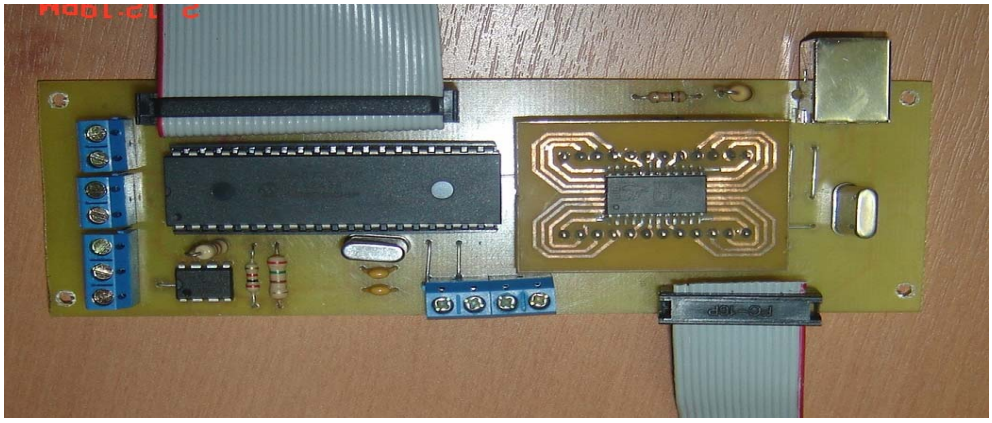


Figura 40.- Foto del Control General

Conjuntamente con los sensores y pantalla LCD acoplados

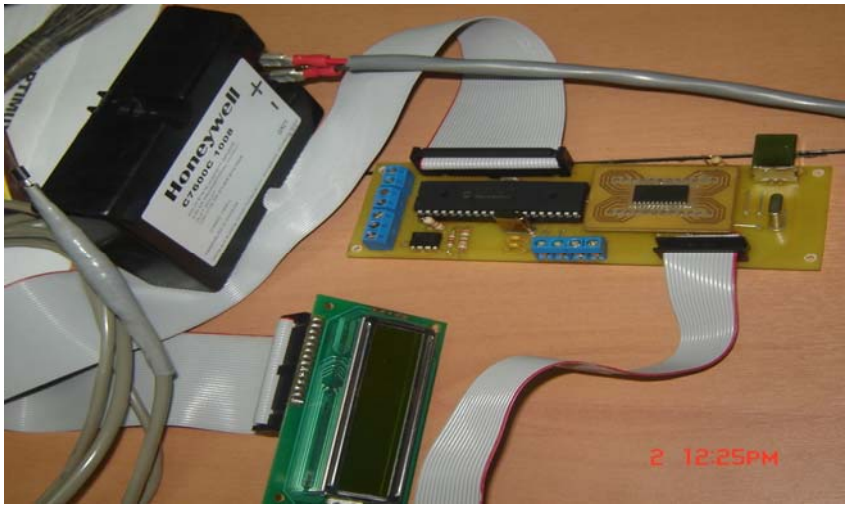


Figura 41.- Foto de la Placa del Control y sensores

Pistas del Control del Motor de Paso

Tenemos las Pistas de la Placa del Control del Motor de Paso

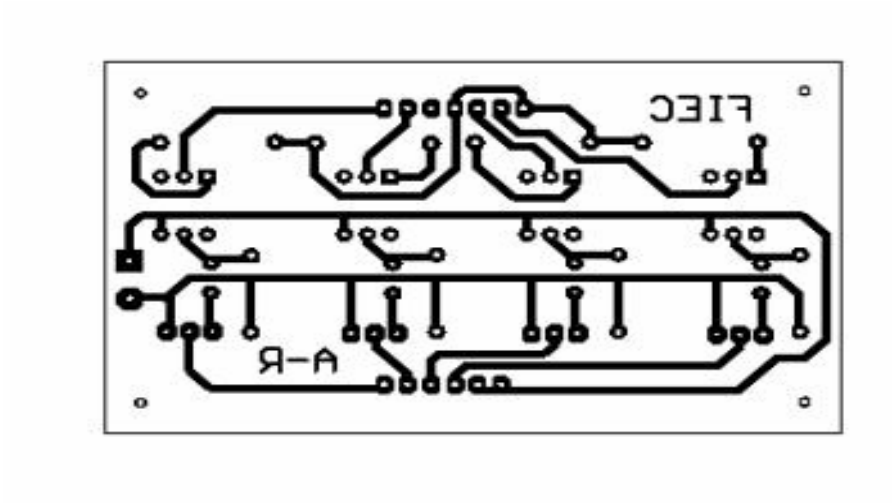


Figura 42.- Pistas del Control del Motor de Paso

Posición de los Elementos de la Placa del Control del Motor

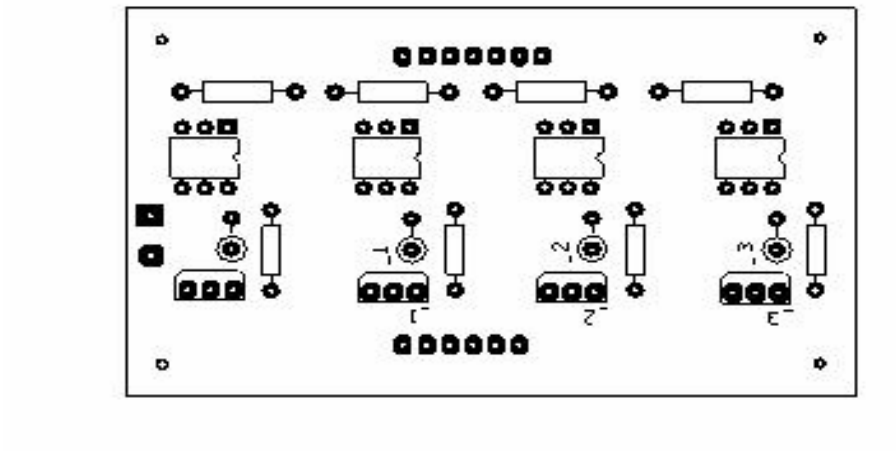


Figura 43.- Posición de los elementos del Control del Motor de Paso

Podemos apreciar las fotos de la placa construida del motor

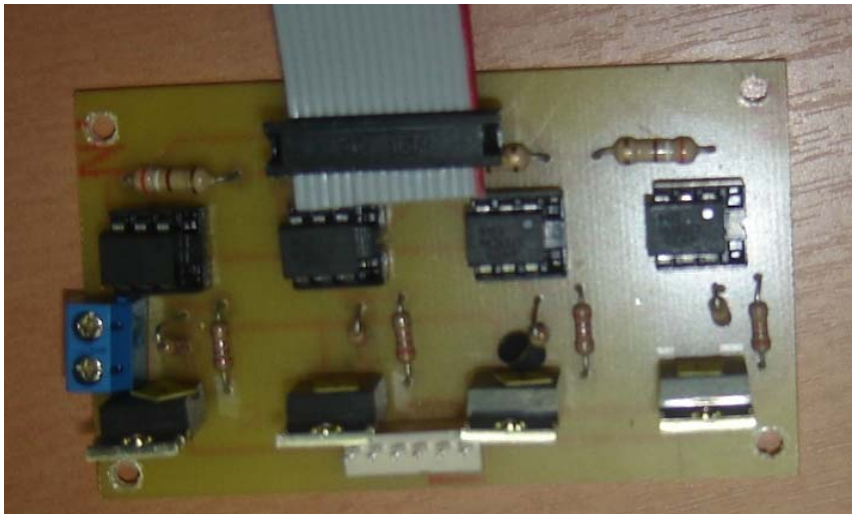


Figura 44.- Foto de la placa del Motor

Y podemos apreciar el acople del motor conjuntamente a la Placa de Control, ya que la tarjeta de Control manda los disparos hacia la Tarjeta Del Motor.

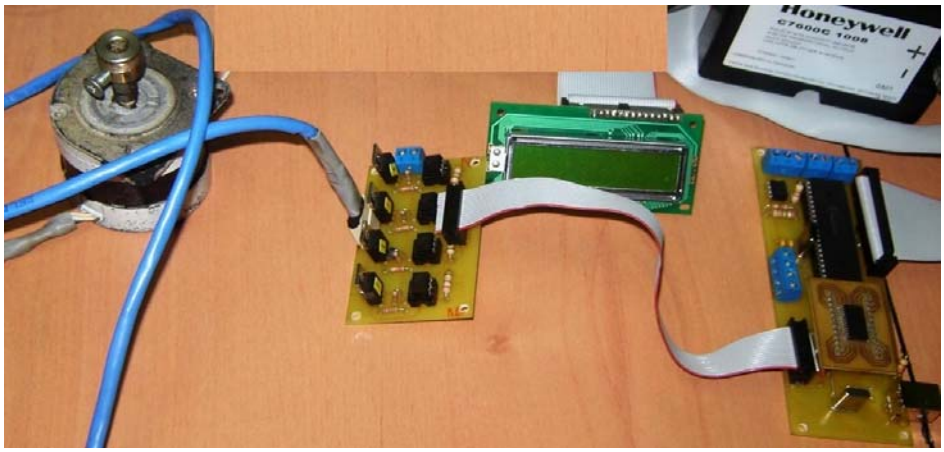


Figura 45.- Foto del acople de la placa del Motor

4.4 Diseño de la Incubadora

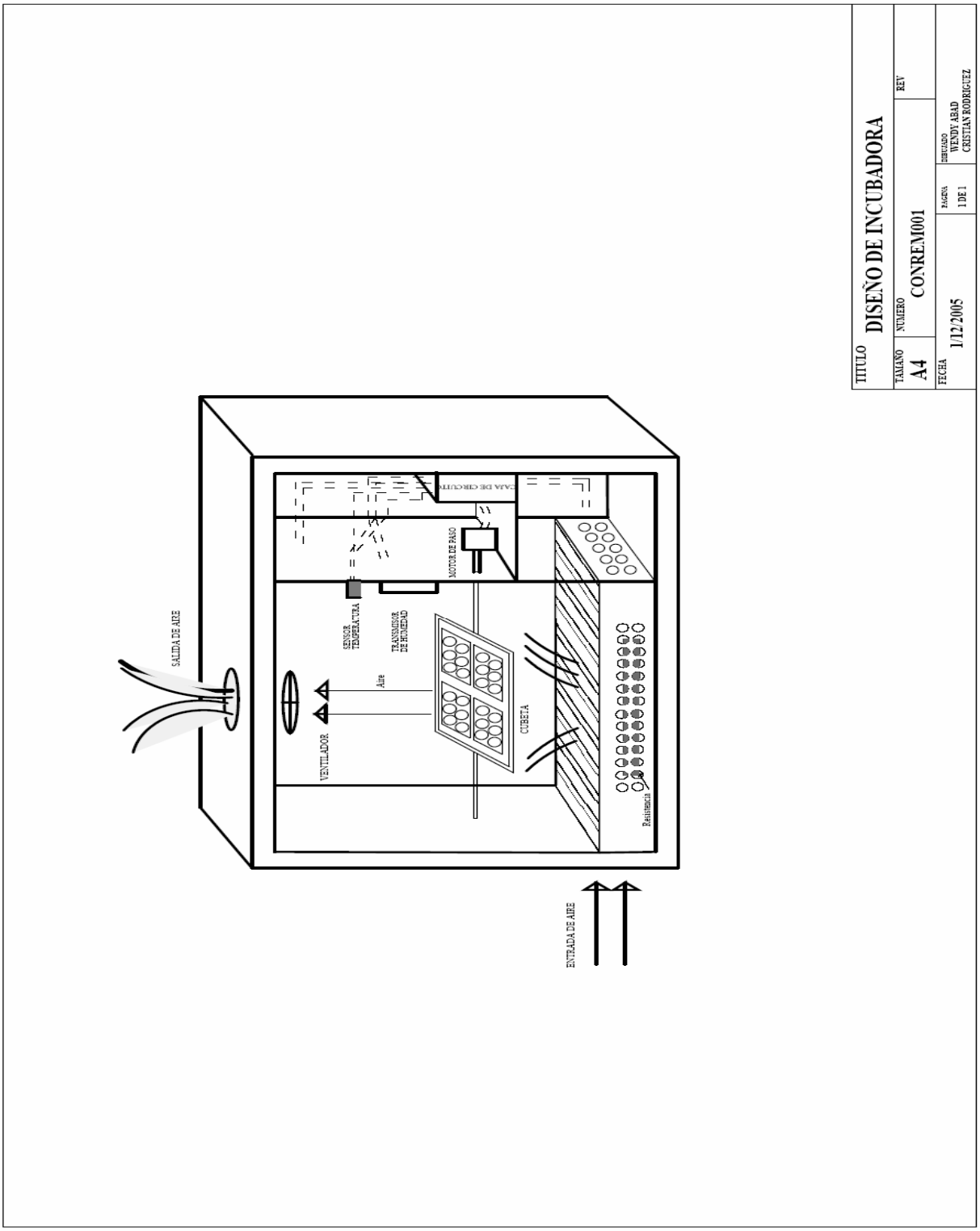


Figura 46.- Diseño de la Incubadora

CAPITULO 5

5.1 CONCLUSIONES Y RECOMENDACIONES

Humedad y Temperatura

Concluimos que la humedad en nuestro país alcanza entre el 70 al 80% sin necesidad de utilizar algún elemento de control, y en cuartos con aire acondicionados alcanza entre 50 al 60 %. La humedad para la incubación de aves debe ser mayor a 50 % y menor al 80% para una cría optima. Lo fundamental es la temperatura, ya que si es mayor a 40 °, los huevos se queman por dentro y el embrión no desarrolla, además debemos tener un buen UPS, ya que los apagones causan que la temperatura baje de 39 a 27 grados, y estas variaciones de temperatura causan deformaciones en el embrión o simplemente se mueren.

Circuito de Comunicación USB

1. Si Windows no reconoce el dispositivo USB o si aparece como desconocido en la lista de dispositivo, es porque usted tiene un problema de hardware. Los errores mas comunes son:

- Inversión de cables del pin D+ con el D-
- La resistencia de 7.5KW esta en el pin incorrecto
- Hay un error en al alimentación de 5V.
- El pin Vpp no esta tierra.
- Asegúrese que el circuito este igual que el esquemático simple.

2. Cuando el Chips USB se usa para interface con otro circuiteria debería tener cuidado de no sobrecargar la corriente de los pines y no exceder el voltaje de los pines. Si el voltaje o corriente sobrepasa los niveles de los chips, tendrá que añadir algún buffer u optoacoplador. Por ejemplo los relays necesitan más de 25 miliamperios y un dispositivo usb puede solo abarcar 25ma, por eso se necesita un amplificador de corriente como un transistor.

MOTOR DE PASO: CIRCUITO CONTROLADOR

1. Con respecto al giro de los huevos , si la frecuencia de pulsos es muy elevada, el motor puede reaccionar en alguna de las siguientes formas:

- Puede que no realice ningún movimiento en absoluto.
- Puede comenzar a vibrar pero sin llegar a girar.
- Puede girar erráticamente.
- O puede llegar a girar en sentido opuesto.

Para obtener un arranque suave y preciso, es recomendable comenzar con una frecuencia de pulso baja y gradualmente ir aumentándola hasta la velocidad deseada sin superar la máxima tolerada. El giro en reversa debería también ser realizado previamente bajando la velocidad de giro y luego cambiar el sentido de rotación.

2. Un motor de paso con 5 cables es casi seguro de 4 fases y unipolar.
3. Un motor de paso con 6 cables también puede ser de 4 fases y unipolar, pero con
4. 2 cables comunes para alimentación. Pueden ser del mismo color.
5. Un motor de pasos con solo 4 cables es comúnmente bipolar.

PROBLEMAS COMUNES EN EL PROCESO DE INCUBACIÓN

Nosotros hemos tomado en cuenta los problemas que se dieron a lo largo del desarrollo del proceso de incubación tanto manual y automático. Con un buen controlador y cuidado de los huevos al momento de sacarlos de la granja no tendremos los siguientes problemas

1. **Los huevos no eran frescos, fueron mal almacenados o no eran fértiles.**

- Los huevos no deberían tener más de dos semanas entre su postura y si incubación.
- Si se encuentra en una estación cálida, guárdelos en la parte baja de la heladera.
- Si en el gallinero del que provenían los huevos había demasiadas gallinas por gallo, es probable que el gallo no llegue a "pisarlas" a todas.

- Si quiere asegurarse que todos los huevos sean fértiles, separe a los reproductores y no tenga más de 6 gallinas por gallo.

2. La temperatura no era la adecuada o no se mantuvo constante durante todo el período de incubación.

- Controle que la temperatura sea la adecuada cada vez que voltea los huevos.
- Si no fuere el caso, regule el termostato, controlador o llame al servicio técnico.
- Si sufre de un corte de energía eléctrica prolongado (más de una hora) mantenga caliente la incubadora agregando agua caliente en la bandeja (cuidando de no mojar los huevos). O tenga un buen UPS.

3. Los embriones se desarrollan dentro del huevo pero mueren antes de romper la cáscara.

- La membrana se pegó por falta de volteo. La incubadora fue limpiada incorrectamente y los huevos se contaminaron con salmonella.
- La humedad fue insuficiente

4. Las codornices nacen antes o después de tiempo.

- La temperatura de incubación no fue la correcta. Esto aumenta el índice de mortandad.

BIBLIOGRAFIA

- Carlos A. Reyes , Manual de Microcontrolador Microcode Studio, Prentice-Hall, Ecuador.
- WWW.mecanique.co.uk Descarga de Microcode y Pbp demo
- WWW.IC-prog.com Descarga de Ic-prog105C.zip
- WWW.todopic.com.ar Descarga manual de pbp, teoria del PIC
- Antonio Creus, Instrumentación Industrial, Editorial Alfaomega, Mexico,1997
- WWW.monografias.com Todo referente a PIC y USB.
- Pagina para información del CYPRESS www.delcom-eng.com
- Revista Internacional de Campo y Agricultura
- Manuales electrónicos: www.roso-control.com \ELEPOT.htm
- Diseños electronicos:www.power.designer.com
- Información del Universal Serial Comunication :www.usb.com

ANEXOS

ANEXO A

MANUAL DE USUARIO

A.1 SERVIDOR

1. CONECTAR LA INCUBADORA A 110 VOLTIOS 60 HZ
2. PRESIONE EL BOTON DE ENCENDIDO 0N/0FF
3. INMEDIATAMENTE, USTED PODRA VISUALIZAR LA TEMPERATURA Y HUMEDAD EN LA LCD, SIMEPRE Y CUANDO LA PUERTA DE LA INCUBADORA ESTE CERRADA
4. APARECERA UN SET POINT DE 37ªC, QUE SERA EL SET POINT DE FABRICA, HASTA QUE USTED LE PROPORCIONE UN NUEVO SET POINT POR MEDIO DE LA COMPUTADORA
5. CONECTE LA INCUBADORA A LA PC POR MEDIO DEL CABLE USB
6. INMEDIATAMENTE SE LE PEDIRA LOS DRIVERS, QUE ESTARAN EN EL CD DE INSTALACION “CONTROL REMOTO”.
7. UNA VES RECONOCIDA LA INCUBADORA, PROCEDA A EJECUTAR EL SOFTWARE DEL SISTEMA “SERVIDOR” QUE ESTA EN EL CD DE INSTALACION
8. APARECERA LA PANTALLA DEL SOFTWARE SERVIDOR , DONDE VISUALIZARA LA TEMPERATURA, HUMEDAD.
9. ESTAMOS LISTOS ….. AHORA PODRA GIRAR EL MOTOR
10. Y SETEAR SU SET POINT DE 37-40 GRADOS
11. SI USTED SE DA CUENTA, APARECERA UNA IP SERVIDOR
12. SI USTED TIENE UNA RED PODRA ACCEDER A EL SERVIDOR DESDE UN CLIENTE.

13. SI USTED TIENE LA INCUBADORA CONECTADA EN UNA COMPUTADORA QUE TIENE UNA IP PUBLICA, PODRA ACCEDER DESDE UN CLIENTE DE CUALQUIER PARTE A TRAVES DE INTERNET

A.2 CLIENTE

1. EJECUTE EL PROGRAMA “CLIENTE”, QUE ESE ENCUENTRA EN EL CD DE INSTALACION
2. APARECERA LA PANTALLA CLIENTE.
3. COLOQUE EL IP DEL SERVIDOR
4. PODRA VISUALIZAR LOS VALORES DE TEMPERATURA, HUMEDAD Y LAS FRAFICAS.
5. PODRA ENVIAR MENSAJES A SU SERVIDOR.
6. PODRA CONTROLAR EL ON/OFF DEL MOTOR Y LA TEMPERATURA.

ANEXO B

DOCUMENTACIÓN DE LA PROGRAMACIÓN

B.1 PROGRAMACIÓN DEL MICROCONTROLADOR PIC16F8774A

```
Define LCD_DREG   PORTB       ;bit de datos del LCD empezando
Define LCD_DBIT   0           ;por B.0 ,B.1, B.2 y B.3
Define LCD_RSREG  PORTB       ;bit de registro del LCD conectar
Define LCD_RSBIT  5           ;en el puerto B.5
Define LCD_EREG   PORTB       ;bit de Enable conectar en el
Define LCD_EBIT   4           ;puerto B.4
SP      var BYTE
TEMP    var byte
HUME    var byte
TRISD =%0:ADCON1 =%100

Inicio:Pause 300
     Lcdout $fe, 2,"TH    HM    SP "
     ADCON0 =%1000001
     ADCON0.2 = 1:Pauseus 50
     IF ADRESH<=67 THEN PWM PORTC.2,3*(SP-ADRESH),30
     if ADRESH>67 THEN PWM PORTC.2,0,30
     TEMP=(ADRESH/5)*3
     ADCON0 =%1001001
```

```
      ADCON0.2 = 1:Pauseus 50:HUME=(ADRESH/5)*2
      if PORTC.0=0 AND PORTC.1=0 THEN  PORTD=TEMP
      if PORTC.0=0 AND PORTC.1=1 THEN  PORTD=HUME
      if PORTC.4=0 AND PORTC.5=0 THEN  Lcdout $fe,$c0,# TEMP,"oC
", #
      HUME,"%  ","37oC ":sp=62
      if PORTC.4=1 AND PORTC.5=0 THEN  Lcdout $fe,$c0,# TEMP,"oC
", #
      HUME,"%  ","38oC ":sp=64
      if PORTC.4=0 AND PORTC.5=1 THEN  Lcdout $fe,$c0,# TEMP,"oC
", #
      HUME,"%  ","39oC ":sp=65
      if PORTC.4=1 AND PORTC.5=1 THEN  Lcdout $fe,$c0,# TEMP,"oC
", #
      HUME,"%  ","40oC ":sp=67
      If PORTC.3=0 THEN Lcdout $fe,2," Puerta abierta "
GOTO   inicio
      Return
End
```

B.2 DISPOSITIVO LÓGICO USB: FIRMWARE

```
;Chip: Cypress Semiconductor CY7C63001 USB Microcontroller
;Assembler: cyasm.exe
;Purpose: demonstrates USB communications with an HID-class device
;Description:
;Handles all required standard USB and HID-class requests.
;Receives data from the host in output reports
;using interrupt transfers on Endpoint 1.
;Sends data to the host in input reports
;using control transfers on Endpoint 0.
;(The chip doesn't support OUT transfers on Endpoint 1.)

;Changes:

;V1.3: 11/20/99
;The length of the string descriptors is now correct.
;Changed the control_read routine to prevent error when sending a
;multiple of 8 bytes.
;(Thanks, Dave Wright)
```

```
;V1.2:
;added watchdog resets in wait loops,
;took out the watchdog reset in the 1-msec. timer ISR.

;V1.1:
;Clears the watchdog only in the main routine
;(so the watchdog will detect if the main routine crashes).
;Additions to the comments.

;V1.0:
;Clears the Watchdog timer on 1-msec. interrupt.
;(Not needed on the development board, but needed for stand-alone.)
;The Endpoint 1 ISR now sets bit 7 of Endpoint 1 TX config to 1.
;(The bit is cleared on EP1 interrupt.)

;=======================================================
================
;assembler directives (equates)
;=======================================================
================

;-----------------------------------------------------------------------
;I/O registers
;-----------------------------------------------------------------------

;I/O ports
Port0_Data:          equ     00h      ; GPIO data port 0
Port1_Data:          equ     01h      ; GPIO data port 1
Port0_Interrupt:      equ     04h      ; Interrupt enable for port 0
Port1_Interrupt:      equ     05h      ; Interrupt enable for port 1
Port0_Pullup:         equ     08h      ; Pullup resistor control for port 0
Port1_Pullup:         equ     09h      ; Pullup resistor control for port 1

;USB ports
USB_EP0_TX_Config:      equ     10h     ; USB EP0 transmit configuration
USB_EP1_TX_Config:      equ     11h     ; USB EP1 transmit configuration
USB_Device_Address:     equ     12h     ; USB device address assigned by
host
USB_Status_Control:     equ     13h     ; USB status and control register
USB_EP0_RX_Status:      equ     14h     ; USB EP0 receive status

;Control ports
Global_Interrupt:       equ     20h     ; Global interrupt enable
```

```
Watchdog:              equ     21h     ; clear watchdog Timer
Timer:                 equ     23h     ; free-running Timer

;GPIO Isink registers
Port0_Isink:           equ     30h
Port0_Isink0:          equ     30h
Port0_Isink1:          equ     31h
Port0_Isink2:          equ     32h
Port0_Isink3:          equ     33h
Port0_Isink4:          equ     34h
Port0_Isink5:          equ     35h
Port0_Isink6:          equ     36h
Port0_Isink7:          equ     37h
Port1_Isink:           equ     38h
Port1_Isink0:          equ     38h
Port1_Isink1:          equ     39h
Port1_Isink2:          equ     3Ah
Port1_Isink3:          equ     3Bh

;Control port
Status_Control:        equ     FFh

;-------------------------------------------------------------------------
;Register bit values
;-------------------------------------------------------------------------

;CPU Status and Control (Status_Control)
RunBit:                equ     1h     ; CPU Run bit
USBReset:              equ     20h    ; USB Bus Reset bit
WatchDogReset:         equ     40h    ; Watchdog Reset bit

; USB EP1 transmit configuration (USB_EP1_TX_Config)
DataToggle:            equ     40h    ; Data 0/1 bit

DISABLE_REMOTE_WAKEUP:  equ   0        ; bit[1] = 0
ENABLE_REMOTE_WAKEUP:   equ   2        ; bit[1] = 1

;-------------------------------------------------------------------------
;Interrupt masks
;-------------------------------------------------------------------------

;The timer-only mask enables the 1-millisecond timer interrupt.
TIMER_ONLY:            equ      4h
```

```
;The enumerate mask enables the following interrupts:
;1-millisecond timer, USB Endpoint 0
ENUMERATE_MASK:         equ     0Ch

;The runtime mask enables the following interrupts:
;1-millisecond timer, USB Endpoint 0, USB Endpoint 1, GPIO
RUNTIME_MASK:           equ     5Ch

;------------------------------------------------------------------------
; USB Constants
; from the USB Spec v1.1
;------------------------------------------------------------------------

;standard request codes
get_status:             equ   0
clear_feature:          equ   1
set_feature:            equ   3
set_address:            equ   5
get_descriptor:         equ   6
set_descriptor:         equ   7
get_configuration:      equ   8
set_configuration:      equ   9
get_interface:          equ  10
set_interface:          equ  11
synch_frame:            equ  12

; standard descriptor types
device:             equ  1
configuration:      equ  2
string:             equ  3
interface:          equ  4
endpoint:           equ  5

; standard feature selectors
endpoint_stalled:       equ  0              ; recipient endpoint
device_remote_wakeup:   equ  1              ; recipient device

;------------------------------------------------------------------------
;HID-class descriptors
;from HID Class Definition v1.1 Draft
;------------------------------------------------------------------------

;Class-specific descriptor types from section 7.1 Standard Requests
HID:                    equ  21h
```

```
report:             equ  22h
physical:           equ  23h

;Class-specific request codes from section 7.2 Class Specific Requests
get_report:         equ   1
get_idle:           equ   2
get_protocol:       equ   3
set_report:         equ   9
set_idle:           equ  10
set_protocol:       equ  11

;-----------------------------------------------------------------------
;USB buffer bytes
;-----------------------------------------------------------------------

;Control Endpoint 0 buffer
Endpoint_0:         equ  70h      ; control endpoint
Endpoint0_Byte0:    equ  70h      ; Endpoint 0, byte 0
Endpoint0_Byte1:    equ  71h      ; Endpoint 0 byte 1
Endpoint0_Byte2:    equ  72h      ; Endpoint 0 byte 2
Endpoint0_Byte3:    equ  73h      ; Endpoint 0 byte 3
Endpoint0_Byte4:    equ  74h      ; Endpoint 0 byte 4
Endpoint0_Byte5:    equ  75h      ; Endpoint 0 byte 5
Endpoint0_Byte6:    equ  76h      ; Endpoint 0 byte 6
Endpoint0_Byte7:    equ  77h      ; Endpoint 0 byte 7


;Endpoint 0 SETUP packet bytes
bmRequestType:      equ  70h
bRequest:           equ  71h
wValue:             equ  72h      ; default wValue (8 bits)
wValueHi:           equ  73h
wIndex:             equ  74h      ; default wIndex (8 bits)
wIndexHi:           equ  75h
wLength:            equ  76h      ; default wLength (8 bits)
wLengthHi:          equ  77h

;Endpoint 1 buffer
endpoint_1:         equ  78h
Endpoint1_Byte0:    equ  78h      ; Endpoint 1, byte 0
Endpoint1_Byte1:    equ  79h      ; Endpoint 1 byte 1
Endpoint1_Byte2:    equ  7Ah      ; Endpoint 1 byte 2
Endpoint1_Byte3:    equ  7Bh      ; Endpoint 1 byte 3
Endpoint1_Byte4:    equ  7Ch      ; Endpoint 1 byte 4
```

```
Endpoint1_Byte5:       equ  7Dh       ; Endpoint 1 byte 5
Endpoint1_Byte6:       equ  7Eh       ; Endpoint 1 byte 6
Endpoint1_Byte7:       equ  7Fh       ; Endpoint 1 byte 7

;-------------------------------------------------------------------
; Variables stored in data memory
;-------------------------------------------------------------------
;USB status
remote_wakeup_status:   equ  30h         ;0=disabled, 2-enabled
configuration_status:   equ  31h         ;0=unconfigured, 1=configured
;idle_status:          equ  33h         ;support SetIdle and GetIdle
protocol_status:       equ  34h         ;0=boot protocol, 1=report protocol

;Other variables:
suspend_counter:       equ  35h         ;number of idle bus milliseconds
loop_temp:             equ  37h         ;temporary loop variable
start_send:            equ  32h         ;0=false, 1=true

;Received data:
Data_Byte0:            equ  38h
Data_Byte1:            equ  39h
Data_Byte2:            equ  3Ah
Data_Byte3:            equ  3Bh
Data_Byte4:            equ  3Ch
Data_Byte5:            equ  3Dh
Data_Byte6:            equ  3Eh
Data_Byte7:            equ  3Fh

temp:                  equ  25h
start_time:            equ  21h
testbit:               equ  22h
interrupt_mask:        equ  20h
endp0_data_toggle:     equ  23h
loop_counter:          equ  24h
data_start:            equ  27h
data_count:            equ  28h
endpoint_stall:        equ  29h

;=================================================================
;interrupt vectors
;=================================================================
```

```
org  00h              ; Reset vector; begin here after a reset.
jmp  Reset

org  02h              ; 128-microsecond interrupt
jmp  DoNothing_ISR

org  04h              ; 1024-millisecond interrupt
jmp  One_mSec_ISR

org  06h              ; Endpoint 0 interrupt
jmp  USB_EP0_ISR

org  08h              ; Endpoint 1 interrupt
jmp  USB_EP1_ISR

org  0Ah               ; reserved interrupt
jmp  Reset

org  0Ch               ; general purpose I/O interrupt
jmp  GPIO_ISR          ; not used

org  0Eh               ; Wakeup_ISR or resume interrupt
jmp  DoNothing_ISR     ; not used

ORG  10h

;=======================================================
===============
;Interrupt routines
;=======================================================
===============

;-----------------------------------------------------------------
; 128-microsecond interrupt, Cext
; Unused. If this interrupt occurs, just push the accumulator (because
; ipret pops it) and re-enable the interrupts.
;-----------------------------------------------------------------

DoNothing_ISR:
    push A
    ;Enable interrupts and return
    mov A,[interrupt_mask]
    ipret Global_Interrupt
```

```
;-----------------------------------------------------------------------
; 1-millisecond interrupt
; Check to see if the chip is in suspend mode and take appropriate action.
; Copy values to Endpoint 1's buffer for sending.
;-----------------------------------------------------------------------

One_mSec_ISR:
    push A
iowr Watchdog
;Find out if enumeration is complete.
;If enumerating is in progress, loop_temp = 0.
    mov A, [loop_temp]
    cmp A, 0h
;If enumeration is still in progress, jump.
    jz not_main
;Enumeration has ended, so decrement the loop counter
;(so it no longer = 0).
    dec [loop_temp]

not_main:
;Check for bus activity.
    iord USB_Status_Control
    and A, 01h
    cmp A,0h
;If no bus activity, increment the suspend counter.
    jz Inc_counter
;If bus activity detected, clear the bus-activity bit,
    iord USB_Status_Control
    and A, 0FEh
    iowr USB_Status_Control
;and clear the suspend counter.
    mov A, 0h
    mov [suspend_counter], A
    jmp Suspend_end

Inc_counter:
;Keep track of the amount of time with no bus activity.
    inc [suspend_counter]
;Get the number of milliseconds the bus has been idle.
    mov A, [suspend_counter]
;Has it been 3 milliseconds?
    cmp A, 03h
;If no, there's nothing else to do.
    jnz Suspend_end
```

```
;If yes, put the chip in Suspend mode.
;Clear the Suspend counter.
    mov A, 0h
    mov [suspend_counter], A
;Enable pullups on Port 1; disable the output DAC.
    mov A, 0h
    iowr Port1_Pullup
    mov A, 0ffh
    iowr Port1_Data
;Set the Suspend bit.
    iord Status_Control
    or A, 08h
    iowr Status_Control
;The chip is now in Suspend mode.
;On exiting Suspend mode, the chip will begin
;executing instructions here:
    nop
;Disable pullups on Port 1. Enable the output DAC.
    mov A, 0ffh
    iowr Port1_Pullup
    mov A, 0h
    iowr Port1_Data

Suspend_end:
;Is endpoint 1 enabled?
    iord USB_EP1_TX_Config
    cmp A,0
;If no, do nothing.
    jz Select
;If yes, is start_send = 1?
;(Start_send adds a short delay after enumeration.)
    mov A, [start_send]
    cmp A, 01h
;If no, do nothing
    jnz Select
;If yes, send data:
    jmp send_value

send_value:
;Copies values from RAM into Endpoint 1's buffer
;and enables sending the bytes on the next poll.

;disable Endpoint 1 interrupts
    mov A,[interrupt_mask]
```

```
    and A, EFh
    mov [interrupt_mask],A
    iowr Global_Interrupt

;Copy values from RAM to Endpoint 1's buffer for transmitting to the host.
;Two bytes:
    mov A, [Data_Byte0]
    mov [Endpoint1_Byte0], A
    mov A, [Data_Byte1]
    mov [Endpoint1_Byte1], A
;Add more bytes if the report format specifies it:
;    mov A, [Data_Byte2]

;    mov [Endpoint1_Byte2], A
;    mov A, [Data_Byte3]
;    mov [Endpoint1_Byte3], A
;    mov A, [Data_Byte4]
;    mov [Endpoint1_Byte4], A
;    mov A, [Data_Byte5]
;    mov [Endpoint1_Byte5], A
;    mov A, [Data_Byte6]
;    mov [Endpoint1_Byte6], A
;    mov A, [Data_Byte7]
;    mov [Endpoint1_Byte7], A

;Other things to try:
;Set the value at Port 0 to equal byte 0 in Endpoint 1's buffer:
;    iord Port0_Data
;    mov [Endpoint1_Byte0], A

;Or set a value here and copy to Endpoint 1's buffer, byte 1:
;    mov A, A5h
;     mov [Endpoint1_Byte1], A

;Configure Endpoint 1's transmit register
;so that the bytes will transmit on the next poll.
    iord USB_EP1_TX_Config
;Don't change the Data 0/1 bit.
    and A,40h
;Set bits 4 and 7 to 1 enable transmitting.
;The low nibble is the number of data bytes (2).
    or A,92h
    iowr USB_EP1_TX_Config
```

```
Select:
;Enable Endpoint 1 interrupts.
    mov A,[interrupt_mask]
    or A, 10h
    mov [interrupt_mask],A
    ipret Global_Interrupt

;-----------------------------------------------------------------------
;GPIO interrupt
;Can be configured to trigger when a port bit toggles.
;Unused here.
;-----------------------------------------------------------------------

GPIO_ISR:
    push A
    push X

    pop X
    mov [interrupt_mask],A
    ipret Global_Interrupt

;-----------------------------------------------------------------------
;Endpoint 1 ISR
;Endpoint 1 can do IN (device to host) transfers only.
;This interrupt triggers when the host acknowledges
;receiving data from Endpoint 1.
;The ISR toggles the data 0/1 bit for the next transaction and
;sets the EnableRespondToIN bit so the chip will respond to the
;next poll of the endpoint.
;-----------------------------------------------------------------------

USB_EP1_ISR:
    push A
;Toggle the data 0/1 bit so it's correct for the next transaction.
    iord USB_EP1_TX_Config
    xor A,40h
;The interrupt clears the EnableRespondToIN bit (bit 7) in the TX Config.
;Set this bit to 1 so data will go out on the next poll.
;This will ensure that a ReadFile API call in a Windows application
;won't hang, waiting for the device to send something.
    or A, 92h
    iowr USB_EP1_TX_Config
;Enable interrupts and return.
    mov A, [interrupt_mask]
```

```
    ipret Global_Interrupt

;-----------------------------------------------------------------------
; Reset processing
; Triggers on Reset or "reserved" interrupt.
;To be safe, initialize everything.
;-----------------------------------------------------------------------

Reset:
;Place the data stack pointer at the lowest address of Endpoint 0's buffer.
;This keeps the stack from writing over the USB buffers.
;The USB buffers are in high RAM;
;the data stack pointer pre-decrements on a Push instruction.

    mov A, Endpoint_0
    swap A, dsp

;Initialize to FFh
    mov A, 0ffh
    iowr Port0_Data             ; output ones to port 0
    iowr Port1_Pullup            ; disable port 1 pullups
                            ; select rising edge interrupts
    iowr Port1_Isink0           ; maximum isink current Port1 bit 0
    iowr Port1_Isink1           ; maximum isink current Port1 bit 1
    iowr Port1_Isink2           ; maximum isink current Port1 bit 2
    iowr Port1_Isink3           ; maximum isink current Port1 bit 3

;Initialize to 00h
    mov A, 0h
    iowr Port1_Data             ; output zeros to port 1
    iowr Port0_Interrupt         ; disable port 0 interrupts
    iowr Port0_Pullup           ; enable port 0 pullups
    iowr Port0_Isink0           ; minimum sink current Port0 bit 0
    iowr Port0_Isink1           ; minimum sink current Port0 bit 1
    iowr Port0_Isink2           ; minimum sink current Port0 bit 2
    iowr Port0_Isink3           ; minimum sink current Port0 bit 3
    iowr Port0_Isink4           ; minimum sink current Port0 bit 4
    iowr Port0_Isink5           ; minimum sink current Port0 bit 5
    iowr Port0_Isink6           ; minimum sink current Port0 bit 6
    iowr Port0_Isink7           ; minimum sink current Port0 bit 7
    mov [Endpoint1_Byte0],A
    mov [Endpoint1_Byte1],A
    mov [Endpoint1_Byte2],A
    mov [endpoint_stall], A
```

```
    mov [remote_wakeup_status], A
    mov [configuration_status], A
    mov [loop_temp], A
    mov [start_send], A
    iowr Watchdog               ; clear watchdog timer

;Initialize values to transmit at Endpoint 1.
;    mov A, A5h
;    mov [Data_Byte0], A
;    mov A, F0h
;    mov [Data_Byte1], A

;Enable Port 1, bit 0 interrupts.
;    mov A, 01h
;    iowr Port1_Interrupt
;
;Test what kind of reset occurred: bus or watchdog?
    iord Status_Control
;Was it a bus reset?
    and A, USBReset
;If yes, jump to handle it.
    jnz BusReset
    iord Status_Control
;Was it a watchdog reset?
    and A, WatchDogReset
;If no, continue to wait for a bus reset
    jz suspendReset
;
;Watchog reset:
;A watchdog reset means that the watchdog timer
;wasn't cleared for 8.192 milliseconds.
;Wait for a bus reset to bring the system alive again.
;Enable 1-millisecond interrupt only
    mov A, TIMER_ONLY
    mov [interrupt_mask],A
    iowr Global_Interrupt
;Wait for a bus reset.
WatchdogHandler:
    jmp WatchdogHandler

suspendReset:
;Return to suspend mode to wait for a USB bus reset.
    mov A, 09h
    iowr Status_Control
```

```
    nop
    jmp suspendReset

BusReset:
;Clear all reset bits.
;Set bit 0 (the run bit).
    mov A, RunBit
    iowr Status_Control
;Set up for enumeration (Endpoint 0 and 1-millisecond interrupts enabled)
    mov A, ENUMERATE_MASK
    mov [interrupt_mask],A
    iowr Global_Interrupt

wait:
;Wait until configured.
    iord USB_EP1_TX_Config
    cmp A, 0
;Clear the watchdog timer
    iowr Watchdog
;If not configured, continue to wait.
    jz wait
;When configured, initialize loop_temp.
;Loop_temp adds a delay in the start of transmission of data.
;The chip will respond to the first IN packet no sooner than
;230 milliseconds after enumeration is complete.
;The delay was included in Cypress' joystick code to prevent problems
;that occurred when power cycled off and on or the joystick was plugged
;in before the host powered up.
;I've left it in because it does no harm and
;other hardware might have similar behavior.
;During the delay, the chip sends a NAK in response to any IN packet.
    mov A, 0ffh
    mov [loop_temp], A

;Enable endpoint 1
    iord USB_EP1_TX_Config
    or A, 92h
    iowr USB_EP1_TX_Config

;=========================================================
===============
; The main program loop.
;=========================================================
===============
```

```
main:
;Find out if the loop_temp delay has timed out.
;Loop_temp =0 if not timed out, FFh if timed out.
    mov A, [loop_temp]
    cmp A, 0Ah
;If no, don't enable transmitting.
    jnc no_set
;If yes, enable transmitting.
    mov A, 01h
    mov [start_send], A
no_set:
;Clear the watchdog timer.
;This has to be done at least once every 8 milliseconds!
    iowr Watchdog
    iord Port0_Data
nochange:
    jmp main

;-----------------------------------------------------------------------
;The Endpoint 0 ISR supports the control endpoint.
;This code enumerates and configures the hardware.
;It also responds to Set Report requests that receive data from the host.
;-----------------------------------------------------------------------

USB_EP0_ISR:
    push A
    iord USB_EP0_RX_Status
;Has a Setup packet been received?
    and A, 01h
;If no, find out if it's an OUT packet.
    jz check_for_out_packet
;If yes, handle it.
;Disable Endpoint 0 interrupts.
    mov A,[interrupt_mask]
    and A, 0F7h
    mov [interrupt_mask], A
    iowr Global_Interrupt
;Find out what the Setup packet contains and handle the request.
     call StageOne
;Re-enable Endpoint 0 interrupts.
    mov A, [interrupt_mask]
    or A, 08h
    mov [interrupt_mask], A
```

```
    jmp done_with_packet

check_for_out_packet:
    iord USB_EP0_RX_Status
;Is it an OUT packet?
    and A, 02h
;If no, ignore it.
    jz done_with_packet
;If yes, process the received data.
;Disable Endpoint 0 interrupts.
    mov A,[interrupt_mask]
    and A, 0F7h
    mov [interrupt_mask], A
    iowr Global_Interrupt

;For debugging: set Port 0, bit 1 to show that we're here.
;    iord Port0_Data
;    or a, 2
;    iowr Port0_Data

;Read the first byte in the buffer
    mov a, [Endpoint_0]
;For debugging: if the first byte =12h, bring Port 0, bit 0 high
;      cmp a, 12h
;      jnz not_a_match
;      iord Port0_Data
;      or a, 4
;      iowr Port0_Data

not_a_match:

;For debugging, add 1 to each byte read
;and copy the bytes to RAM.
;These bytes will be sent back to the host.

      push X
      ;data_count holds the number of bytes left to read.
      ;X holds the index of the address to read
      ;and the index of the address to store the received data.
      ;Initialize the X register.
      mov X, 0

      Get_Received_Data:
      ;Find out if there are any bytes to read.
```

```
    mov A, 0
    cmp A, [data_count]
    ;Jump if nothing to read.
    jz DoneWithReceivedData

    ;Get a byte.
    mov A, [X + Endpoint_0]
    ;For debugging, increment the received value.
    ;(Endpoint 1 will send it back to the host.)
    ;If the value is 255, reset to 0.
    ;Otherwise increment it.
    cmp A, 255
    jz ResetToZero
    inc A
    jmp NewValueSet
    ResetToZero:
    mov A, 0
    NewValueSet:
    ;Save the value.
    mov [X + Data_Byte0], A
    ;Decrement the number of bytes left to read.
    dec [data_count]
    ;Increment the address to read.
    inc X
    ;Do another
    jmp Get_Received_Data

    DoneWithReceivedData:
    pop X

;For debugging, set Port 0 to match the value written.
;    iowr Port0_Data

;Handshake by sending a 0-byte data packet.
call Send0ByteDataPacket

done_with_packet:
;Re-enable Endpoint 0 interrupts.
    mov A,[interrupt_mask]
    or A, 08h
    mov [interrupt_mask], A
    ipret Global_Interrupt
```

```
;=======================================================
==================
;Control transfers
;=======================================================
==================

;-----------------------------------------------------------------------
;Control transfer, stage one.
;Find out whether the request is a standard device or HID-class request,
;the direction of data transfer,
;and whether the request is to a device, interface, or endpoint.
;(from Table 9.2 in the USB spec)
;-----------------------------------------------------------------------

StageOne:
;Clear the Setup flag
    mov A, 00h
    iowr USB_EP0_RX_Status
;Set the StatusOuts bit to cause auto-handshake after receiving a data packet.
    mov A, 8
    iowr USB_Status_Control
;bmRequestType contains the request.
     mov A, [bmRequestType]

;Standard device requests. From the USB spec.
; host to device requests
       cmp A, 00h
       jz RequestType00             ; bmRequestType = 00000000 device
;      cmp A, 01h                  *** not required ***
;      jz RequestType01             ; bmRequestType = 00000001 interface
       cmp A, 02h
       jz RequestType02             ; bmRequestType = 00000010 endpoint
       cmp A, 80h
; device to host requests
       jz RequestType80             ; bmRequestType = 10000000 device
       cmp A, 81h
       jz RequestType81             ; bmRequestType = 10000001 interface
       cmp A, 82h
       jz RequestType82             ; bmRequestType = 10000010 endpoint

;HID-class device requests. From the HID spec
; host to device requests
       cmp A, 21h
       jz RequestType21             ; bmRequestType = 00100001 interface
```

```
        cmp A, 22h                  ; *** not in HID spec ***
        jz RequestType22              ; bmRequestType = 00100010 endpoint
; device to host requests
        cmp A, A1h
        jz RequestTypeA1              ; bmRequestType = 10100001 interface

; Stall unsupported requests
SendStall:
      mov A, A0h
     iowr USB_EP0_TX_Config
      ret

;-------------------------------------------------------------------
;Control transfer, stage two
;Find out which request it is.
;-------------------------------------------------------------------

;Host to device with device as recipient
RequestType00:

;The Remote Wakeup feature is disabled on reset.
    mov A, [bRequest]     ; load bRequest
; Clear Feature                 bRequest = 1
    cmp A, clear_feature
    jz ClearRemoteWakeup
; Set Feature                   bRequest = 3
    cmp A, set_feature
    jz SetRemoteWakeup

; Set the device address to a non-zero value.
; Set Address                   bRequest = 5
    cmp A, set_address
    jz SetAddress

; Set Descriptor is optional.
; Set Descriptor              bRequest = 7    *** not supported ***

;If wValue is zero, the device is not configured.
;The only other legal value for this firmware is 1.
;Set Configuration           bRequest = 9
   cmp A, set_configuration
   jz SetConfiguration

;Stall unsupported requests.
```

```
   jmp SendStall


;Host to device with interface as recipient    *** not required ***
; RequestType01:
;       mov A, [bRequest]       ; load bRequest

; There are no interface features defined in the spec.
; Clear Feature              bRequest = 1    *** not supported ***
; Set Feature                bRequest = 3    *** not supported ***

; Set Interface is optional.
; Set Interface              bRequest = 11   *** not supported ***

;Stall unsupported requests.
;       jmp SendStall

;Host to device with endpoint as recipient
RequestType02:
    mov A, [bRequest]     ; load bRequest

; The only standard feature defined for an endpoint is endpoint_stalled.
; Clear Feature              bRequest = 1
    cmp A, clear_feature
    jz ClearEndpointStall
; Set Feature              bRequest = 3
    cmp A, set_feature
    jz SetEndpointStall

;Stall unsupported functions.
   jmp SendStall

;Device to host with device as recipient
RequestType80:
    mov A, [bRequest]         ; load bRequest

; Get Status             bRequest = 0
    cmp A, get_status
    jz GetDeviceStatus

; Get Descriptor             bRequest = 6
    cmp A, get_descriptor
    jz GetDescriptor
```

```
; Get Configuration          bRequest = 8
    cmp A, get_configuration
    jz GetConfiguration

;Stall unsupported requests.
    jmp SendStall

;Device to host with interface as recipient
RequestType81:
    mov A, [bRequest]     ; load bRequest

; Get Status               bRequest = 0
     cmp A, get_status
     jz GetInterfaceStatus

; Get Interface returns the selected alternate setting.
;  This firmware supports no alternate settings.
; Get Interface                bRequest = 10   *** not supported ***

;The HID class defines one more request for bmRequestType=10000001
; Get Descriptor               bRequest = 6
     cmp A, get_descriptor
     jz GetDescriptor

;Stall unsupported functions
     jmp SendStall

;Device to host with endpoint as recipient
RequestType82:
    mov A, [bRequest]          ; load bRequest
; Get Status               bRequest = 0
    cmp A, get_status
     jz GetEndpointStatus

; Get Descriptor               bRequest = 6
    cmp A, get_descriptor
    jz GetDescriptor
; Sync Frame                   bRequest = 12   *** not supported ***

;Stall unsupported functions.
    jmp SendStall


;Check for HID class requests
```

```
;Host to device with endpoint as recipient
RequestType21:
     mov A, [bRequest]      ; load bRequest

; Set Report              bRequest = 9
    cmp A, set_report
    jz SetReport

; Set Idle                bRequest = 10
     cmp A, set_idle
     jz SetIdle

; Set Protocol            bRequest = 11
     cmp A, set_protocol
     jz SetProtocol

;Stall unsupported requests
    jmp SendStall

RequestType22:
       mov A, [bRequest]      ; load bRequest

; Set Report              bRequest = 9
    cmp A, set_report
    jz SetReport

;Stall unsupported requests
    jmp SendStall


;Device to host with endpoint as recipient
RequestTypeA1:
       mov A, [bRequest]      ; load bRequest

; Get Report              bRequest = 1
       cmp A, get_report
       jz GetReport

; Get Idle                bRequest = 2
       cmp A, get_idle
       jz GetIdle

; Get Protocol            bRequest = 3
```

```
        cmp A, get_protocol
        jz GetProtocol

;Stall unsupported requests
        jmp SendStall

;-------------------------------------------------------------------
;Control transfer, stage three
;Process the request.
;-------------------------------------------------------------------

;The host controls whether or not a device can request a remote wakeup.

; Disable the remote wakeup capability.
ClearRemoteWakeup:
        mov A, [wValue]
        cmp A, device_remote_wakeup
        jnz SendStall
;Handshake by sending a data packet
        call Send0ByteDataPacket
        mov A, DISABLE_REMOTE_WAKEUP
        mov [remote_wakeup_status], A
        ret

; Enable the remote wakeup capability.
SetRemoteWakeup:
        mov A, [wValue]
        cmp A, device_remote_wakeup
;If not a match, stall.
        jnz SendStall
;Handshake by sending a 0-byte data packet
        call Send0ByteDataPacket
;Perform the request.
        mov A, ENABLE_REMOTE_WAKEUP
        mov [remote_wakeup_status], A
        ret

SetAddress:
; Set the device address to match wValue in the Setup packet.
;Complete the requested action after completing the transaction.
;Handshake by sending a 0-byte data packet.
        call Send0ByteDataPacket
;Perform the request
        mov A, [wValue]
```

```
        iowr USB_Device_Address
        ret

SetConfiguration:
;Unconfigured: wValue=0, configured: wValue=1.
;Also clear any stall condition and set Data 0/1 to Data0.
;Handshake by sending a 0-byte data packet.
      call Send0ByteDataPacket
;Save the configuration status.
      mov A, [wValue]
      mov [configuration_status], A
;Clear any stall condtion
      mov A, 0
      mov [endpoint_stall], A
;Set data 0/1 to Data0
      iord USB_EP1_TX_Config
      and A, ~DataToggle

;Set the configuration status.
     iowr USB_EP1_TX_Config
     mov A, [configuration_status]
     cmp A, 0
;If configured, jump.
     jnz device_configured

;If unconfigured:
;Disable Endpoint 1
     iord USB_EP1_TX_Config
     and A, EFh
     iowr USB_EP1_TX_Config
;Disable Endpoint 1 interrupts.
     mov A, [interrupt_mask]
     and A, EFh
     mov [interrupt_mask], A
     jmp done_configuration

;If configured:
device_configured:
;Send NAK in response to IN packets
     iord USB_EP1_TX_Config
     and A,7Fh
;Enable Endpoint 1
     or A, 10h
     iowr USB_EP1_TX_Config
```

```
;Enable interrupts: Endpoint 1 and GPIO
    mov A, [interrupt_mask]
    or A, 50h
    mov [interrupt_mask], A
;Send NAK in response to Endpoint 0 OUT packets.
    iord USB_Status_Control
    and A,0EFh
    iowr USB_Status_Control
done_configuration:
      ret

ClearEndpointStall:
;Clear the stall (halt) condition for Endpoint 1.
;wValue = 0.
      mov A, [wValue]
      cmp A, endpoint_stalled
;If endpoint_stalled = 0, the endpoint isn't stalled
;and there's nothing to clear. Return a Stall for the request.
      jnz SendStall
;
;Clear Endpoint 1 stall
;Handshake by sending a 0-byte data packet
     call Send0ByteDataPacket
;Clear the stall condition
     mov A,0
     mov [endpoint_stall], A
;Set Data 0/1 to Data0
     iord USB_EP1_TX_Config
     and A, ~DataToggle
     iowr USB_EP1_TX_Config
;Send NAK in response to Endpoint 0 OUT packets.
     iord USB_Status_Control
     and A,0EFh
     iowr USB_Status_Control
     ret

;Stall Endpoint 1
SetEndpointStall:
;wValue = 0.
      mov A, [wValue]
      cmp A, endpoint_stalled
;If endpoint_stalled = 1, the endpoint is already stalled,
;so return a Stall for this request.
      jnz SendStall
```

```
;Handshake by sending a 0-byte data packet.
        call Send0ByteDataPacket
;Stall the endpoint.
        mov A,1
        mov [endpoint_stall], A
        mov A, 30h
        iowr USB_EP1_TX_Config
        ret

GetDeviceStatus:
;Device Status is a 2-byte value.
;Bit 0 must be 0 (bus-powered).
;Bit 1 is remote wakeup: 0=disabled, 1=enabled.
;All other bits are unused.
;Return to status bytes to the host.
        mov A, 2
        mov [data_count], A
;control_read_table holds the two possible values for device status.
;Get the address of the first value.
        mov A, (get_dev_status_table - control_read_table)
;Add an index value to select the correct value.
        add A, [remote_wakeup_status]
;Send the value.
        jmp SendDescriptor

GetDescriptor:
;The high byte of wValue contains the descriptor type.
;The low byte of wValue contains the descriptor index.

        mov A, [wValueHi]               ; load descriptor type

;Test for standard descriptor types first.
;Supported descriptor types are device, configuration, string.
;Unsupported descriptor types are interface, endpoint.

; Get Descriptor (device)               wValueHi = 1
    cmp A, device
    jz GetDeviceDescriptor
; Get Descriptor (configuration)        wValueHi = 2
    cmp A, configuration
    jz GetConfigurationDescriptor
; Get Descriptor (string)               wValueHi = 3
    cmp A, string
    jz GetStringDescriptor
```

```
; Test for HID-class descriptor types.
; Get Descriptor (HID)              wValueHi = 21h
       cmp A, HID
       jz GetHIDDescriptor
; Get Descriptor (report)           wValueHi = 22h
    cmp A, report
    jz GetReportDescriptor
; Get Descriptor (physical)          wValueHi = 23h  *** not supported ***
;Stall unsupported requests.
    jmp SendStall

GetConfiguration:
;Send the current device configuration.
;0 = unconfigured, 1 = configured.

;Send 1 byte
       mov A, 1
       mov [data_count], A
;Get the address of the data to send.
       mov A, (get_configuration_status_table - control_read_table)
;Add an index to point to the correct configuration.
       add A, [configuration_status]
;Send the data.
       jmp SendDescriptor

GetInterfaceStatus:
;Interface status is 2 bytes, which are always 0.
;Send 2 bytes.
       mov A, 2
       mov [data_count], A
;Get the address of the data to send.
       mov A, (get_interface_status_table - control_read_table)
;Send the data.
       jmp SendDescriptor

GetEndpointStatus:
;Endpoint status is 2 bytes.
;Bit 0 = 0 when the endpoint is not stalled.
;Bit 0 = 1 when the endpoint is stalled.
;All other bits are unused.
;Send 2 bytes.
       mov A, 2
       mov [data_count], A
```

```
;Get the stall status.
        mov A, [endpoint_stall]
;Shift left to get an index (0 or 2) to point to data
;in the endpoint status table
        asl A
;Get the address of the data to send.
        add A, (get_endpoint_status_table - control_read_table)
;Send the data.
        jmp SendDescriptor

SetReport:
;The CY7C63000 doesn't support interrupt-mode OUT transfers.
;So the host uses Control transfers with Set_Report requests
;to get data from the device.

;Get the report data.

;For debugging: set Port 0, bit 0 =1 to show that we're here.
;       iord Port0_Data
;       or a, 1
;       iowr Port0_Data

;Find out how many bytes to read. This value is in WLength.
;Save the length in data_count.
    mov A, [wLength]
    mov [data_count], A

;Enable receiving data at Endpoint 0 by setting the EnableOuts bit
;The bit clears following any Setup or OUT transaction.
    iord USB_Status_Control
    or A, 10h
;Clear the StatusOuts bit to disable automatic sending of ACK after
;receiving a valid status packet in a Control read (IN) transfer.
;Otherwise, the USB engine will respond to a data OUT packet with a Stall.
    and A, F7h
    iowr USB_Status_Control
;Now we're ready to receive the report data.
;An Endpoint 0 OUT interrupt indicates the arrival of the report data.
ret

SetIdle:
        jmp SendStall   ; *** not supported ***

SetProtocol:
```

```
;Switches between a boot protocol (wValue=0) and report protocol
(wValue=1).
;This firmware doesn't distinguish between protocols.
        mov A, [wValue]
        mov [protocol_status], A
        call Send0ByteDataPacket
        ret

GetReport:
;Sends a report to the host.
;The high byte of wValue contains the report type.
;The low byte of wValue contains the report ID.
;Not supported (Use interrupt transfers to send data.)
    jmp SendStall

GetReportDescriptor:
;Save the descriptor length
    mov A, (end_hid_report_desc_table - hid_report_desc_table)
    mov [data_count], A
;Get the descriptor's starting address.
    mov A, (hid_report_desc_table - control_read_table)
    call SendDescriptor

     ret

GetIdle:
;Not supported
        jmp SendStall

GetProtocol:
;Send the current protocol status.
;Send 1 byte.
        mov A, 1
        mov [data_count], A
;Get the address of the data to send.
        mov A, (get_protocol_status_table - control_read_table)
;Add an index that points to the correct data.
        add A, [protocol_status]
;Send the data.
        jmp SendDescriptor

; Standard Get Descriptor routines
;
;Send the device descriptor.
```

```
GetDeviceDescriptor:
;Get the length of the descriptor
;(stored in the first byte in the device descriptor table).
       mov A, 0
      index device_desc_table
       mov [data_count], A
;Get the starting address of the descriptor.
       mov A, (device_desc_table - control_read_table)
;Send the descriptor.
       jmp SendDescriptor

GetConfigurationDescriptor:
;Send the configuration descriptor.
;Get the length of the descriptor.
    mov A, (end_config_desc_table - config_desc_table)
     mov [data_count], A
;Get the starting address of the descriptor.
    mov A, (config_desc_table - control_read_table)
;Send the descriptor.
    jmp SendDescriptor

GetStringDescriptor:
;Use the string index to find out which string it is.
    mov A, [wValue]
    cmp A, 0h
    jz LanguageString
    cmp A, 01h
    jz ManufacturerString
    cmp A, 02h
    jz ProductString
;    cmp A, 03h
;    jz SerialNumString
;    cmp A, 04h
;    jz ConfigurationString
;    cmp A, 05h
;    jz InterfaceString
; No other strings supported
     jmp SendStall

SendDescriptor:
;The starting address of the descriptor is in the accumulator. Save it.
       mov [data_start], A
;Get the descriptor length.
       call get_descriptor_length
```

```
;Send the descriptor.
       call control_read
       ret

;Send the requested string.
;For each, store the descriptor length in data_count, then send the descriptor.
LanguageString:
    mov A, (USBStringDescription1 - USBStringLanguageDescription)
     mov [data_count], A
    mov A, (USBStringLanguageDescription - control_read_table)
     jmp SendDescriptor
ManufacturerString:
    mov A, ( USBStringDescription2 - USBStringDescription1)
     mov [data_count], A
    mov A, (USBStringDescription1 - control_read_table)
     jmp SendDescriptor
ProductString:
    mov A, ( USBStringDescription3 - USBStringDescription2)
     mov [data_count], A
    mov A, (USBStringDescription2 - control_read_table)
     jmp SendDescriptor
;SerialNumString:
;    mov A, ( USBStringDescription4 - USBStringDescription3)
;     mov [data_count], A
;    mov A, (USBStringDescription3 - control_read_table)
;     jmp SendDescriptor
;ConfigurationString:
;    mov A, ( USBStringDescription5 - USBStringDescription4)
;     mov [data_count], A
;    mov A, (USBStringDescription4 - control_read_table)
;     jmp SendDescriptor
;InterfaceString:
;    mov A, ( USBStringEnd - USBStringDescription5)
;     mov [data_count], A
;    mov A, (USBStringDescription5 - control_read_table)
;     jmp SendDescriptor

; HID class Get Descriptor routines
;
GetHIDDescriptor:
;Send the HID descriptor.
;Get the length of the descriptor.
    mov A, (Endpoint_Descriptor - Class_Descriptor)
     mov [data_count], A
```

```
;Get the descriptor's starting address.
    mov A, ( Class_Descriptor - control_read_table)
;Send the descriptor.
     call SendDescriptor
     ret

;=====================================================
================
;USB support routines
;=====================================================
================

get_descriptor_length:
;The host sometimes lies about the number of bytes it
;wants from a descriptor.
;A request to get a descriptor should return the smaller of the number
;of bytes requested or the actual length of the descriptor.
;Get the requested number of bytes to send
    mov A, [wLengthHi]
;If the requested high byte is >0,
;ignore the high byte and use the firmware's value.
;(255 bytes is the maximum allowed.)
    cmp A, 0
    jnz use_actual_length
;If the low byte =0, use the firmware's value.
    mov A, [wLength]
    cmp A, 0
    jz use_actual_length
;If the requested number of bytes => the firmware's value,
;use the firmware's value.
    cmp A, [data_count]
    jnc use_actual_length
;If the requested number of bytes < the firmware's value,
;use the requested number of bytes.
    mov [data_count], A
use_actual_length:
      ret

Send0ByteDataPacket:
;Send a data packet with 0 bytes.
;Use this handshake after receiving an OUT data packet.
;Enable responding to IN packets and set Data 0/1 to Data 1.
    mov A, C0h
    iowr USB_EP0_TX_Config
```

```
;Enable interrupts.
    mov A, [interrupt_mask]
    iowr Global_Interrupt
WaitForDataToTransfer:
;Wait for the data to transfer.
;Clear the watchdog timer
    iowr Watchdog
;Bit 7 of USB_EP0_TX_Config is cleared when the host acknowledges
;receiving the data.
    iord USB_EP0_TX_Config
    and A, 80h
    jnz WaitForDataToTransfer
    ret

control_read:
;Do a Control Read transfer.
;The device receives a Setup packet in the Setup stage,
;sends 1 or more data packets (IN) in the Data stage,
;and receives a 0-length data packet (OUT) in the Status stage.
;Before calling this routine, the firmware must set 2 values:
;data_start is the starting address of the descriptor to send,
;expressed as an offset from the control read table.
;data_count is the number of bytes in the descriptor.
    push X
;Set the Data 0/1 bit to 0.
    mov A, 00h
    mov [endp0_data_toggle], A

control_read_data_stage:
;Initialize count variables.
    mov X, 00h
    mov A, 00h
    mov [loop_counter], A

;Clear the Setup bit.
    iowr USB_EP0_RX_Status
;Check the Setup bit.
    iord USB_EP0_RX_Status
    and A, 01h
;If not cleared, another setup packet has arrived,
;so exit the routine.
    jnz control_read_status_stage

;Set the StatusOuts bit to 1 to cause the device to automatically return
```

```
;ACK in response to a received OUT packet in the Status stage.
    mov A, 08h
    iowr USB_Status_Control
;If there is no data to send, prepare a 0-length data packet.
;(The host might require a final 0-length packet if the descriptor is
;a multiple of 8 bytes.)
    mov A, [data_count]
    cmp A, 00h
    jz dma_load_done

dma_load_loop:
;Copy up to 8 bytes for transmitting into Endpoint 0's buffer
;and increment/decrement the various counting variables.

;Place the byte to send in the accumulator:
;(control_read_table) + (data_start).
    mov A, [data_start]
    index control_read_table
;Place the byte in Endpoint 0's buffer.
    mov [X + Endpoint_0], A
;Increment the offset of the data being sent.
    inc [data_start]
;Increment the offset of Endpoint 0's buffer.
    inc X
;Increment the number of bytes stored in the buffer.
    inc [loop_counter]
;Decrement the number of bytes left to send.
    dec [data_count]
;If the count = 0, there's no more data to load.
    jz dma_load_done
;If 8 bytes haven't been loaded into the buffer, get another byte.
;If 8 bytes have been loaded, it's the maximum for the transaction,
;so send the data.
    mov A, [loop_counter]
    cmp A, 08h
    jnz dma_load_loop

dma_load_done:
;Send the data.

;Check the Setup bit.
;If it's not 0, another Setup packet has arrived,
;so exit the routine.
    iord USB_EP0_RX_Status
```

```
    and A, 01h
    jnz control_read_status_stage

;Set the bits in the USB_EP0_TX_Config register.
;Toggle the Data 0/1 bit.
    mov A, [endp0_data_toggle]
    xor A, 40h
    mov [endp0_data_toggle], A
;Enable responding to IN token packets.
    or A, 80h
;The low 4 bits hold the number of bytes to send.
    or A, [loop_counter]
    iowr USB_EP0_TX_Config
;Enable interrupts
    mov A, [interrupt_mask]
    iowr Global_Interrupt

wait_control_read:

;Clear the watchdog timer
    iowr Watchdog

;Wait for the data to transfer and the host to acknowledge,
;indicated by Bit 7 = 0.
    iord USB_EP0_TX_Config
    and A, 80h
;When all of the transaction's data has transferred,
;find out if there is more data to send in the transfer.
    jz control_read_data_stage
;Find out if the host has sent an OUT packet to acknowledge
;and end the transfer.
    iord USB_EP0_RX_Status
    and A, 02h
    jz wait_control_read

control_read_status_stage:
;The transfer is complete.
    pop X
    mov A, [interrupt_mask]
    iowr Global_Interrupt
    ret

;=================================================
===============
```

```
;Lookup Tables
;Contain the descriptors and the codes for status indicators.
;The firmware accesses the information by referencing a specific
;table's address as an offset from the control_read_table.
;=================================================================
================

control_read_table:

device_desc_table:
    db 12h        ; Descriptor length (18 bytes)
    db 01h        ; Descriptor type (Device)
    db 10h,01h     ; Complies with USB Spec. Release (0110h = release 1.10)
    db 00h        ; Class code (0)
    db 00h        ; Subclass code (0)
    db 00h        ; Protocol (No specific protocol)
    db 08h        ; Maximum packet size for Endpoint 0 (8 bytes)
    db 25h,09h     ; Vendor ID (Lakeview Research, 0925h)
    db 34h,12h     ; Product ID (1234)
    db 01h,00h     ; Device release number (0001)
    db 01h        ; Manufacturer string descriptor index
    db 02h        ; Product string descriptor index
    db 00h        ; Serial Number string descriptor index (None)
    db 01h        ; Number of possible configurations (1)
end_device_desc_table:

config_desc_table:
    db 09h        ; Descriptor length (9 bytes)
    db 02h        ; Descriptor type (Configuration)
    db 22h,00h     ; Total data length (34 bytes)
    db 01h        ; Interface supported (1)
    db 01h        ; Configuration value (1)
    db 00h        ; Index of string descriptor (None)
    db 80h        ; Configuration (Bus powered)
    db 32h        ; Maximum power consumption (100mA)

Interface_Descriptor:
    db 09h        ; Descriptor length (9 bytes)
    db 04h        ; Descriptor type (Interface)
    db 00h        ; Number of interface (0)
    db 00h        ; Alternate setting (0)
    db 01h        ; Number of interface endpoint (1)
    db 03h        ; Class code ()
    db 00h        ; Subclass code ()
```

```
    db 00h          ; Protocol code ()
    db 00h          ; Index of string()

Class_Descriptor:
    db 09h          ; Descriptor length (9 bytes)
    db 21h          ; Descriptor type (HID)
    db 00h,01h      ; HID class release number (1.00)
    db 00h          ; Localized country code (None)
    db 01h          ; # of HID class dscrptr to follow (1)
    db 22h          ; Report descriptor type (HID)
                ; Total length of report descriptor
    db (end_hid_report_desc_table - hid_report_desc_table),00h

Endpoint_Descriptor:
    db 07h          ; Descriptor length (7 bytes)
    db 05h          ; Descriptor type (Endpoint)
    db 81h          ; Encoded address (Respond to IN, 1 endpoint)
    db 03h          ; Endpoint attribute (Interrupt transfer)
    db 06h,00h      ; Maximum packet size (6 bytes)
    db 0Ah          ; Polling interval (10 ms)

end_config_desc_table:

;---------------------------------------------------------------------
;The HID-report descriptor table
;---------------------------------------------------------------------

hid_report_desc_table:
    db 06h, A0h, FFh      ;       Usage Page (vendor defined) FFA0
    db 09h, 01h     ;           Usage (vendor defined)
    db A1h, 01h     ;           Collection (Application)
    db 09h, 02h     ;           Usage (vendor defined)
    db A1h, 00h     ;           Collection (Physical)
    db 06h, A1h, FFh ;          Usage Page (vendor defined)

;The input report
    db 09h, 03h     ;           usage - vendor defined
    db 09h, 04h     ;           usage - vendor defined
    db 15h, 80h     ;           Logical Minimum (-128)
    db 25h, 7Fh     ;           Logical Maximum (127)
    db 35h, 00h     ;           Physical Minimum (0)
    db 45h, FFh;                Physical Maximum (255)
;   db 66h, 00h, 00h;           Unit (None (2 bytes))
    db 75h, 08h     ;           Report Size (8)  (bits)
```

```
    db 95h, 02h     ;           Report Count (2)  (fields)
    db 81h, 02h     ;           Input (Data, Variable, Absolute)

;The output report
    db 09h, 05h     ;           usage - vendor defined
    db 09h, 06h     ;           usage - vendor defined
    db 15h, 80h     ;           Logical Minimum (-128)
    db 25h, 7Fh     ;           Logical Maximum (127)
    db 35h, 00h     ;           Physical Minimum (0)
    db 45h, FFh     ;           Physical Maximum (255)
;   db 66h, 00h, 00h;             Unit (None (2 bytes))
    db 75h, 08h     ;           Report Size (8)  (bits)
    db 95h, 02h     ;           Report Count (2)  (fields)
    db 91h, 02h     ;           Output (Data, Variable, Absolute)

    db C0h          ;           End Collection

    db C0h          ;           End Collection

end_hid_report_desc_table:

;---------------------------------------------------------------------
;String Descriptors
;---------------------------------------------------------------------

;Define the strings

; string 0
USBStringLanguageDescription:
   db 04h          ; Length
   db 03h          ; Type (3=string)
   db 09h          ; Language:  English
   db 04h          ; Sub-language: US
; string 1

;The Length value for each string =
;((number of characters) * 2) + 2

USBStringDescription1:     ; IManufacturerName
   db 1Ah          ; Length
   db 03h          ; Type (3=string)
   dsu "USB Complete" ;

; string 2
```

```
USBStringDescription2:     ; IProduct
   db 16h          ; Length
   db 03h          ; Type (3=string)
   dsu "HID Sample"  ;

;string 3
;If the firmware contains a serial number, it must be unique
;for each device or the devices may not enumerate properly.
USBStringDescription3:     ; serial number

; string 4
;USBStringDescription4:     ; configuration string descriptor
;    db 16h          ; Length
;    db 03h          ; Type (3=string)
;    dsu "Sample HID"  ;

;string 5
;USBStringDescription5:     ; configuration string descriptor
;    db 32h          ; Length
;    db 03h          ; Type (3=string)
;    dsu "EndPoint1 Interrupt Pipe"  ;

USBStringEnd:

;-----------------------------------------------------------------------
;Status information.
;The status can apply to the device or an interface or endpoint.
;An index selects the correct value.
;-----------------------------------------------------------------------
get_dev_status_table:
       db      00h, 00h        ; remote wakeup disabled, bus powered
       db      02h, 00h        ; remote wakeup enabled, bus powered
get_interface_status_table:
       db      00h, 00h        ; always return both bytes zero
get_endpoint_status_table:
       db      00h, 00h        ; not stalled
       db      01h, 00h        ; stalled
get_configuration_status_table:
       db      00h             ; not configured
       db      01h             ; configured
get_protocol_status_table:
       db      00h             ; boot protocol
       db      01h             ; report protocol
```

B. 3 MODULO DE LA COMUNICACIÓN USB

En este modulo encontramos las diferentes funciones para la comunicación USB.

```
Attribute VB_Name = "VB_USB"

' Global data variables
Public hDevice As Long      'Handle to the device
Public lpDeviceName As String 'Copy of the device name

' Registry and File Constants
Const HKEY_LOCAL_MACHINE = &H80000002
Const GENERIC_READ = &H80000000
Const GENERIC_WRITE = &H40000000
Const FILE_SHARE_WRITE = &H2
Const FILE_SHARE_READ = &H1
Const OPEN_EXISTING = &H3
Const CTL_CODE_SEND_PACKET = &H222028
Public Type PacketStructure
   Recipient   As Byte
   DeviceModel As Byte
   MajorCmd    As Byte
   MinorCmd    As Byte
   DataLSB     As Byte
   DataMSB     As Byte
   Length      As Integer
End Type
Public Type RetPacketStructure
   B0   As Byte
   B1   As Byte
   B2   As Byte
   B3   As Byte
   B4   As Byte
   B5   As Byte
   B6   As Byte
   B7   As Byte
End Type

' declare references to external procedures in a dynamic-link library (DLL).
Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA"
_
```

```
   (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As
Long, _
   ByVal samDesired As Long, phkResult As Long) As Long

Declare Function RegQueryValueEx Lib "advapi32" Alias
"RegQueryValueExA" _
   (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved
As Long, _
   ByRef lpType As Long, ByVal szData As String, ByRef lpcbData As
Long) As Long

Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As
Long

Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" _
   (ByVal lpFileName As String, ByVal dwDesiredAccess As Long, _
    ByVal dwShareMode As Long, ByVal lpSecurityAttributes As Long, _
    ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As
Long, _
    ByVal hTemplateFile As Long) As Long

Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As
Boolean
Declare Function DeviceIoControl Lib "kernel32" _
   (ByVal hDevice As Long, ByVal dwIocontrolCode As Long, _
    ByRef lpBuffer As PacketStructure, ByVal nInBufferSize As Long, _
    ByRef lpOutBuffer As RetPacketStructure, ByVal nOutBufferSize As
Long, _
    ByRef lpBytesReturned As Long, ByVal lpOverLapped As Long) As
Boolean

'OpenDevice - This function reads the device name from the registry and then
'opens the device and stores a handle to the device in hDevice. Returns zero
'on error.  This function also stores the full device name is DeviceName.
Function OpenDevice() As Boolean
On Error GoTo ERROR_HANDLER
'Get Device Name from the registry
lpDeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _

"System\CurrentControlSet\Services\Delcom\USBIODS\Parameters\", _
                  "DeviceName", "")
If lpDeviceName = "" Then   ' exit on error
   MsgBox "Unable to open device, check connection and power."
   lpDeviceName = "Device Not Found!"
```

```
   OpenDevice = False
   Exit Function
End If
' Try and open the device. This will fail if device not present
hDevice = CreateFile(lpDeviceName, GENERIC_READ Or
GENERIC_WRITE, _
                FILE_SHARE_WRITE Or FILE_SHARE_READ, 0, _
                OPEN_EXISTING, 0, 0)
If hDevice <= 0 Then    ' check for error
   MsgBox "Unable to open device, check connection and power"
   lpDeviceName = "Device Not Found!"
   OpenDevice = False
Else
   OpenDevice = True
End If

Exit Function
ERROR_HANDLER:
   MsgBox "OpenDevice() ERROR #" & Str$(Err) & " : " & Error
End Function

'CloseDevice - Closes the device, always close device after use.
'If you don't close the device after use, you will not be able
'to open it up again without cycle pluging the USB cable.
Function CloseDevice() As Boolean
On Error GoTo ERROR_HANDLER
CloseDevice = CloseHandle(hDevice)  ' Close the device
hDevice = 0                         ' Null the handle
If CloseDevice = False Then         ' Check for errors
   MsgBox "Error closing file"     ' Display errors
End If
Exit Function
ERROR_HANDLER:
   MsgBox "CloseDevice() ERROR #" & Str$(Err) & " : " & Error
End Function
'Sends the USB packet to the device
Function SendPacket(ByRef TxPacket As PacketStructure) As
RetPacketStructure
Dim lpResult As Long
Dim RxPacket As RetPacketStructure

On Error GoTo ERROR_HANDLER
If hDevice <= 0 Then                ' check for valid handle
   MsgBox "SendPacket() Handle invalid!"
```

```
   Exit Function
End If
TxPacket.Recipient = 8      ' always 8
TxPacket.DeviceModel = 18   ' always 18

    ' Call the read length function
If 0 = DeviceIoControl(hDevice, CTL_CODE_SEND_PACKET, TxPacket,
8 + TxPacket.Length, _
                  RxPacket, 8, lpResult, 0) Then
   MainForm.Timer1.Enabled = False ' turn off timer when error
   MsgBox "SendPacket() DeviceIoControl Failed. Timer Disabled"
   Exit Function
End If

SendPacket = RxPacket

Exit Function
ERROR_HANDLER:
   MainForm.Timer1.Enabled = False ' turn off timer when error
   MsgBox "SendPacket() ERROR #" & Str$(Err) & " : " & Error & " Timer
Disabled"
End Function

' GetRegValue - Gets the Key value in the registry given a registry key.
Function GetRegValue(hKey As Long, lpszSubKey As String, szKey As
String, _
                szDefault As String) As String
On Error GoTo ERROR_HANDLER

Dim phkResult As Long, lResult As Long
Dim szBuffer As String, lBuffSize As Long

'Create Buffer
szBuffer = Space(255)          ' Allocate buffer space
lBuffSize = Len(szBuffer)      ' Set the length

                        'Open the Key
RegOpenKeyEx hKey, lpszSubKey, 0, 1, phkResult

                        'Query the value
lResult = RegQueryValueEx(phkResult, szKey, 0, 0, szBuffer, lBuffSize)

RegCloseKey phkResult          'Close the Key
```

```
'Return obtained value
If lResult = ERROR_SUCCESS Then
   GetRegValue = szBuffer
Else
   GetRegValue = szDefault
End If
Exit Function

ERROR_HANDLER:
   MsgBox "GetRegValue() ERROR #" & Str$(Err) & " : " & Error &
Chr(13) _
   & "Please exit and try again."

   GetRegValue = szDefault
End Function
```

B.4 PROGRAMACIÓN DEL SOFTWARE SERVIDOR

FORM TEMPERATURA:

```
Private Sub atras1_Click()
Unload Me
'MainForm.Show
End Sub

Private Sub Form_Load()
   With Trend3
      .AutoRedraw = False
      .XSpan = 1 / 24 / 60
      .XMax = Now
      .XMin = .XMax - .XSpan
      .SetXDisplay .XMin, .XMax
   End With
naranja.Visible = False
atras1.Visible = False
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
'boton pausa
```

```
If X > 6480 And X < 8040 And Y > 4800 And Y < 5280 Then
  naranja.Visible = True
End If
If X < 6480 Or X > 8040 Or Y < 4800 Or Y > 5280 Then
  naranja.Visible = False
End If
'boton atras
If X > 0 And X < 1455 And Y > 4800 And Y < 5610 Then
  atras1.Visible = True
End If
If X < 0 Or X > 1455 Or Y < 4800 Or Y > 5610 Then
  atras1.Visible = False
End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Set frmTime = Nothing
End Sub
Private Sub naranja_Click()
 Timer3.Enabled = Not Timer3.Enabled
   If Timer3.Enabled = True Then
      P.Caption = "Pausa"
   Else
      P.Caption = "Iniciar"
   End If
End Sub
Private Sub P_Click()
Timer3.Enabled = Not Timer3.Enabled
   If Timer3.Enabled = True Then
      P.Caption = "Pausa"
   Else
      P.Caption = "Iniciar"
   End If
End Sub

Private Sub Timer3_Timer()
   Dim Value As Single
   Static i As Long
      With Trend3
      Value = MainForm.valor_temph.Caption / 1
      .AddXY 0, Now, Value
      .Refresh
      i = i + 1
      If i > 500 Then i = 0
```

```
      DisplayStatistic
   End With
End Sub

Private Sub Trend3_CursorChange(X As Double)
   lblX.Caption = Format(Trend3.CursorX, "hh:mm:ss")
   lblY.Caption = Format(Trend3.CursorValue(0), "##0.00")
End Sub

Private Sub Trend3_Pan()
   DisplayStatistic
End Sub

Private Sub DisplayStatistic()
   With Trend3
      lblMax.Caption = Format$(.VarMax, "##0.00")
      lblMax.Refresh
      lblMin.Caption = Format$(.VarMin, "##0.00")
      lblMin.Refresh
      lblVisibleMax.Caption = Format$(.VarVisibleMax, "##0.00")
      lblVisibleMax.Refresh
      lblVisibleMin.Caption = Format$(.VarVisibleMin, "##0.00")
      lblVisibleMin.Refresh
   End With
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   MainForm.Show
End Sub

Private Sub Form_Resize()
If Me.WindowState <> 1 Then Me.Height = 6360: Me.Width = 8190
End Sub
```

MAINFORM:

```
Dim mivalor As String
Dim giro
Dim girando
Dim pulsos
Dim db As New ADODB.Connection
Dim tp As New ADODB.Recordset
Dim ta, th As Double
Function Send(ByVal xDato As Variant)
  If Winsock1.State = 7 Then
```

```
    Winsock1.SendData xDato
  End If
    Exit Function

End Function
Private Sub Boton_giro_Click()
   If Boton_giro.Value = 1 Then
     girando = 1
   Else
     girando = 0
   End If
End Sub

Private Sub Command1_Click()
sp_th.Text = sp_th.Text + 1
If Trim(sp_th.Text) > 39 Then
   sp_th.Text = 39
End If
End Sub

Private Sub Command2_Click()
If giro < 10 Then
  giro = giro + 1
End If
End Sub

Private Sub Command3_Click()
sp_th.Text = sp_th.Text - 1
If Trim(sp_th.Text) < 37 Then
   sp_th.Text = 37
End If
End Sub
Private Sub Command4_Click()
If giro > 0 Then
  giro = giro - 1
End If

End Sub
Private Sub flecha0_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
flecha0.Visible = False
flecha1.Visible = True
End Sub
```

```
Private Sub flecha1_Click()
Call Send("4CHAT:" & Text1.Text)
List1.AddItem Text1.Text
End Sub

Sub EsperarConexion()
   Label1.Caption = Winsock1.LocalIP
   Label2.Caption = Winsock1.LocalHostName
   If Winsock1.State = 0 Then
      Winsock1.Listen
   End If
End Sub
Private Sub Form_Activate()
EsperarConexion
End Sub
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
 'tp.Close
   'db.Close

   j = CloseDevice
   Winsock1.Close

   Unload FrmPausa
   Unload Humedad
   Unload Temperatura

End Sub
Private Sub Humed_Click()
Humedad.Show
End Sub
Private Sub Humedad_n_Click()
Humedad.Show
End Sub
Private Sub Temp_Click()
Temperatura.Show
End Sub
Private Sub temp_n_Click()
Temperatura.Show
End Sub
Private Sub Winsock1_ConnectionRequest(ByVal requestID As Long)
Winsock1.Close
Winsock1.Accept requestID

End Sub
```

```
Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
Dim datos As String
Dim Ident As String
Dim nPos As Integer
Dim valor As String
If Winsock1.State = 7 Then
Winsock1.GetData datos
nPos = Val(Mid(datos, 1, 1))
If nPos > 0 Then
   Ident = Trim(UCase(Mid(datos, 2, nPos)))
   valor = Trim(Mid(datos, Len(Ident) + 3, 250))
   s = valor_temph.Caption & "&" & valor_humedad & "&" &
velocidad.Caption & "&" & Trim(sp_th.Text)
   If Ident = "DATA" Then Call Send(s)
   If Ident = "CHAT" Then List1.AddItem datos
End If
End If
Exit Sub
End Sub
Private Sub Form_Load()
Dim xRecive As String
Dim Status As Long
Dim Packet As PacketStructure
Dim Ret As RetPacketStructure
Status = OpenDevice()

If Status = 0 Then
   Status = CloseDevice()
End
Else
   Packet.MajorCmd = 11
   Packet.MinorCmd = 10
   Packet.DataLSB = 0
   Packet.DataMSB = 0
   Packet.Length = 0
   Ret = SendPacket(Packet)
End If

'db.CursorLocation = adUseClient
'db.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Documents
and Settings\Corv\Escritorio\temperaturas.mdb;Persist Security Info=False"
'tp.Open "select*from T", db, adOpenKeyset, adLockPessimistic
Timer1.Interval = 100
Timer1.Enabled = True
```

```
flecha1.Visible = False
temp_n.Visible = False
Humedad_n.Visible = False
giro = 10
pulsos = 0
sp_th.Text = 37
End Sub
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
'boton de temperatura
If X > 0 And X < 1560 And Y > 0 And Y < 480 Then
  temp_n.Visible = True
End If
If X < 0 Or X > 1560 Or Y < 0 Or Y > 480 Then
  temp_n.Visible = False
End If
'boton de humedad
If X > 1560 And X < 3120 And Y > 0 And Y < 480 Then
  Humedad_n.Visible = True
End If
If X < 1560 Or X > 3120 Or Y < 0 Or Y > 480 Then
  Humedad_n.Visible = False
End If
'flecha para enviar
If X > 2880 And X < 3420 And Y > 2880 And Y < 3420 Then
  flecha1.Visible = True
End If
If X < 2880 Or X > 3420 Or Y < 2880 Or Y > 3420 Then
  flecha1.Visible = False
End If
flecha0.Visible = True
flecha1.Visible = False

End Sub
Private Sub datos()
Dim Packet As PacketStructure
Dim Ret As RetPacketStructure
Dim Humedad, ta, th
   ' enviando codigo 001 para recibir temperatura huevos
   Packet.MajorCmd = 10
   Packet.MinorCmd = 12
   Packet.Length = 0
   Packet.DataLSB = &H1
   Packet.DataMSB = &H0
```

```
    Ret = SendPacket(Packet)
    Call retardo
    'pedir datos
    Packet.MajorCmd = 11
    Packet.MinorCmd = 0
    Packet.Length = 0
    Ret = SendPacket(Packet)
    mivalor = Ret.B0
    valor_temph.Caption = mivalor
    ' encerar los codigos
    Packet.MajorCmd = 10
    Packet.MinorCmd = 12
    Packet.Length = 0
    Packet.DataLSB = &H0
    Packet.DataMSB = &H1
    Ret = SendPacket(Packet)
      ' enviando codigo 010 para recibir temperatura agua
    Packet.DataLSB = &H2
    Packet.DataMSB = &H0
    Ret = SendPacket(Packet)
    Call retardo
    ' pedir datos
    Packet.MajorCmd = 11
    Packet.MinorCmd = 0
    Packet.Length = 0
    Ret = SendPacket(Packet)
    mivalor = Ret.B0
    valor_humedad.Caption = mivalor
    ' encerar los codigos
    Packet.MajorCmd = 10
    Packet.MinorCmd = 12
    Packet.Length = 0
    Packet.DataLSB = &H0
    Packet.DataMSB = &H2
    Ret = SendPacket(Packet)
End Sub

Public Function VerEstado(Estado As Byte) As String
    Select Case Estado
        Case 0
            VerEstado = "Sin Conexiones"
           MainForm.EsperarConexion
            With MainForm
             End With
```

```
        Case 1
           VerEstado = "Abierto"
        Case 2
           VerEstado = "Esperando Conexion"
           With MainForm

           End With
        Case 3
           VerEstado = "Conexion Pendiente"
        Case 4
           VerEstado = "Resolviendo Host"

        Case 5
           VerEstado = "Host Resuelto"
        Case 6
           VerEstado = "Conectando"
        Case 7
           VerEstado = "Conectado"
           With MainForm

              .LblIpLocal.Caption = "IP = " & .Winsock1.RemoteHostIP
           End With
        Case 8
           VerEstado = "Cerrando Conexion"
           Winsock1.Close
        Case 9
           VerEstado = "Error"
        End Select
End Function

Private Sub Timer1_Timer()
Dim Packet As PacketStructure
Dim Ret As RetPacketStructure
Dim c2, c1 As String
Dim g, columna
On Error GoTo ERROR_HANDLER
 BarraEstado.Panels.Item(1) = "Estado = " & VerEstado(Winsock1.State)

 Call datos
'----------------------------------------
    velocidad.Caption = giro
     'alarmas para las temperaturas de los huevos
    If Trim(valor_temph.Caption) < 85 Then
       If temph_verde.Visible = True Then
```

```
            temph_verde.Visible = False
        Else
            temph_verde.Visible = True
        End If
    Else: temph_verde.Visible = True
    End If
    If Trim(valor_temph.Caption) > 85 And Trim(valor_temph.Caption) < 170
Then
        If temph_amarillo.Visible = True Then
            temph_amarillo.Visible = False
        Else
            temph_amarillo.Visible = True
        End If
    Else: temph_amarillo.Visible = True
    End If
    If Trim(valor_temph.Caption) > 170 Then
        If temph_rojo.Visible = True Then
            temph_rojo.Visible = False
        Else
            temph_rojo.Visible = True
        End If
    Else: temph_rojo.Visible = True
    End If
    'alarmas para las temperaturas de la humedad
    If Trim(valor_humedad.Caption) < 85 Then
        If humed_verde.Visible = True Then
            humed_verde.Visible = False
        Else
            humed_verde.Visible = True
        End If
    Else: humed_verde.Visible = True
    End If
    If Trim(valor_humedad.Caption) > 85 And Trim(valor_humedad.Caption)
< 170 Then
        If humed_amarillo.Visible = True Then
            humed_amarillo.Visible = False
        Else
            humed_amarillo.Visible = True
        End If
    Else: humed_amarillo.Visible = True
    End If
    If Trim(valor_humedad.Caption) > 170 Then
        If humed_rojo.Visible = True Then
            humed_rojo.Visible = False
```

```
    Else
        humed_rojo.Visible = True
    End If
  Else: humed_rojo.Visible = True
  End If
  Packet.MajorCmd = 10
  Packet.MinorCmd = 12
  Packet.Length = 0
If girando = 1 Then
  'giro del motor de paso
  Packet.DataLSB = &HC0          'envio codigo 1100
  Packet.DataMSB = &H0
  Ret = SendPacket(Packet)
  Call retardote
  ' encerar los codigos
  Packet.DataLSB = &H0
  Packet.DataMSB = &HC0
  Ret = SendPacket(Packet)
  Call retardote
  Packet.DataLSB = &H60          'envio codigo 0110
  Packet.DataMSB = &H0
  Ret = SendPacket(Packet)
  Call retardote
  ' encerar los codigos
  Packet.DataLSB = &H0
  Packet.DataMSB = &H60
  Ret = SendPacket(Packet)
  Call retardote
  Packet.DataLSB = &H30          'envio codigo 0011
  Packet.DataMSB = &H0
  Ret = SendPacket(Packet)
  Call retardote
  ' encerar los codigos
  Packet.DataLSB = &H0
  Packet.DataMSB = &H30
  Ret = SendPacket(Packet)
  Call retardote
  Packet.DataLSB = &H90          'envio codigo 1001
  Packet.DataMSB = &H0
  Ret = SendPacket(Packet)
  Call retardote
  ' encerar los codigos
  Packet.DataLSB = &H0
  Packet.DataMSB = &H90
```

```
    Ret = SendPacket(Packet)
    Call retardote
    pulsos = pulsos + 1
    grados.Caption = pulsos * 7.2
    If pulsos = 25 Then
       Boton_giro.Value = 0
       girando = 0
       pulsos = 0
       grados.Caption = ""
    End If
 End If

Exit Sub
ERROR_HANDLER:
    MsgBox "Timer1 ERROR #" & Str$(Err) & " : " & Error & Chr(13)
End Sub
Private Sub retardo()
For j = 0 To 3225
   If retarda.Visible = True Then
      retarda.Visible = False
   Else
      retarda.Visible = True

   End If
Next j
End Sub
Private Sub retardote()
 For j = 0 To giro
     Call retardo
 Next j
End Sub

Private Sub Winsock1_Error(ByVal Number As Integer, Description As
String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
 Winsock1.Close
End Sub
```

FORM HUMEDAD:

```
Private Sub atras1_Click()
Unload Me
'MainForm.Show
End Sub

Private Sub Form_Load()
    With Trend3
        .AutoRedraw = False
        .XSpan = 1 / 24 / 60
        .XMax = Now
        .XMin = .XMax - .XSpan
        .SetXDisplay .XMin, .XMax
    End With
naranja.Visible = False
atras1.Visible = False
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
'boton pausa
If X > 6480 And X < 8040 And Y > 4800 And Y < 5280 Then
   naranja.Visible = True
End If
If X < 6480 Or X > 8040 Or Y < 4800 Or Y > 5280 Then
   naranja.Visible = False
End If
'boton atras
If X > 0 And X < 1455 And Y > 4800 And Y < 5610 Then
   atras1.Visible = True
End If
If X < 0 Or X > 1455 Or Y < 4800 Or Y > 5610 Then
   atras1.Visible = False
End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set frmTime = Nothing
End Sub

Private Sub Label1_Click()
 Timer3.Enabled = Not Timer3.Enabled
```

```
    If Timer3.Enabled = True Then
        P.Caption = "Pausa"
    Else
        P.Caption = "Iniciar"
    End If
End Sub

Private Sub naranja_Click()
 Timer3.Enabled = Not Timer3.Enabled
    If Timer3.Enabled = True Then
        P.Caption = "Pausa"
    Else
        P.Caption = "Iniciar"
    End If
End Sub
Private Sub P_Click()
Timer3.Enabled = Not Timer3.Enabled
    If Timer3.Enabled = True Then
        P.Caption = "Pausa"
    Else
        P.Caption = "Iniciar"
    End If
End Sub

Private Sub Timer3_Timer()
    Dim Value As Single
    Static i As Long
    If Trim(MainForm.valor_humedad.Caption) <> "" Then
        With Trend3
        Value = Trim(MainForm.valor_humedad.Caption) / 1
        .AddXY 0, Now, Value
        .Refresh
        i = i + 1
        If i > 500 Then i = 0
        DisplayStatistic
        End With
    End If
End Sub

Private Sub Trend3_CursorChange(X As Double)
    lblX.Caption = Format(Trend3.CursorX, "hh:mm:ss")
    lblY.Caption = Format(Trend3.CursorValue(0), "##0.00")
End Sub
```

```
Private Sub Trend3_Pan()
   DisplayStatistic
End Sub

Private Sub DisplayStatistic()
   With Trend3
      lblMax.Caption = Format$(.VarMax, "##0.00")
      lblMax.Refresh
      lblMin.Caption = Format$(.VarMin, "##0.00")
      lblMin.Refresh
      lblVisibleMax.Caption = Format$(.VarVisibleMax, "##0.00")
      lblVisibleMax.Refresh
      lblVisibleMin.Caption = Format$(.VarVisibleMin, "##0.00")
      lblVisibleMin.Refresh
   End With
End Sub


Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   MainForm.Show
End Sub

Private Sub Form_Resize()
If Me.WindowState <> 1 Then Me.Height = 6360: Me.Width = 8190
End Sub
```

B.5 PROGRAMACIÓN DEL SOFTWARE CLIENTE

```
Dim bth, bta, bhu As Boolean

Private Sub TxtIpServidor_Change()
   If TxtIpServidor.Text <> "" Then
      If Len(TxtIpServidor) >= 7 Then
         CmdConectar.Enabled = 1
      Else
         CmdConectar.Enabled = 0
      End If
   Else
      If Len(TxtIpServidor) < 7 Then
         CmdConectar.Enabled = 0
      End If
   End If
End Sub

Private Sub TxtIpServidor_KeyPress(KeyAscii As Integer)
   If KeyAscii = 13 Then
      CmdConectar_Click
   End If
End Sub
Function Send(ByVal xDato As Variant)
   WSocket.SendData xDato

End Function


Private Sub CmdConectar_Click()
   WSocket.RemoteHost = TxtIpServidor.Text
   WSocket.Connect
End Sub

Private Sub CmdDesconectar_Click()
  WSocket.Close
  TxtIpServidor = Empty

End Sub

Private Sub Form_Activate()
Label2.Caption = WSocket.LocalIP
Label3.Caption = WSocket.LocalHostName
```

```
End Sub

Private Sub aceptar2_Click()
HUMEDAD.Show
End Sub

Private Sub flecha1_Click()
Call Send("4Chat:" & Text1.Text)
End Sub

Private Sub Form_Load()
flecha1.Visible = False
'naranja.Visible = False
Temp_n.Visible = False
Humedad_n.Visible = False
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
'boton de temperatura
If X > 0 And X < 1560 And Y > 0 And Y < 480 Then
  Temp_n.Visible = True
End If
If X < 0 Or X > 1560 Or Y < 0 Or Y > 480 Then
  Temp_n.Visible = False
End If
'boton de humedad
If X > 1560 And X < 3120 And Y > 0 And Y < 480 Then
  Humedad_n.Visible = True
End If
If X < 1560 Or X > 3120 Or Y < 0 Or Y > 480 Then
  Humedad_n.Visible = False
End If
'flecha para enviar
If X > 2880 And X < 3420 And Y > 2880 And Y < 3420 Then
  flecha1.Visible = True
End If
If X < 2880 Or X > 3420 Or Y < 2880 Or Y > 3420 Then
  flecha1.Visible = False
End If
End Sub
Private Sub Humed_Click()
HUMEDAD.Show
'Unload Me
```

```
End Sub

Private Sub Humedad_n_Click()
HUMEDAD.Show
'Unload Me
End Sub



Public Function VerEstado(Estado As Byte) As String
   Select Case Estado
      Case 0
         VerEstado = "Sin Conexiones"
         With SERVIDOR
            .TxtIpServidor.Enabled = True
            '.TxtEnviar.Enabled = False
            '.CmdEnviar.Enabled = False
         End With
      Case 1
         VerEstado = "Abierto"
      Case 2
         VerEstado = "Esperando Conexion"
      Case 3
         VerEstado = "Conexion Pendiente"
      Case 4
         VerEstado = "Resolviendo Host"
      Case 5
         VerEstado = "Host Resuelto"
      Case 6
         VerEstado = "Conectando"
      Case 7
         VerEstado = "Conectado"
         With SERVIDOR
            .CmdConectar.Enabled = False
            .TxtIpServidor.Enabled = False
           ' .TxtEnviar.Enabled = True
            '.CmdEnviar.Enabled = True
         End With
      Case 8
         VerEstado = "Cerrando Conexion"
         SERVIDOR.WSocket.Close
      Case 9
         VerEstado = "Error"
      End Select
```

```
End Function
Private Sub WSocket_ConnectionRequest(ByVal requestID As Long)
 WSocket.Close
 WSocket.Accept requestID
End Sub


Private Sub Temp_n_Click()
TEMPERATURA.Show
'Unload Me
End Sub

Private Sub Temp_Click()
TEMPERATURA.Show
'Unload Me
End Sub

Private Sub Timer1_Timer()
BarraEstado.Panels.Item(1) = "Estado = " & VerEstado(WSocket.State)
If WSocket.State = 7 Then
 Call Send("4DATA")
End If
'alarmas para las temperaturas de los huevos
If Trim(valor_temph.Caption) <> "" Then 'And Trim(valor_humed.Caption)
<> "" Then
   If Trim(valor_temph.Caption) < 85 Then
     If temph_verde.Visible = True Then
         temph_verde.Visible = False
     Else
         temph_verde.Visible = True
     End If
   Else: temph_verde.Visible = True
   End If
   If Trim(valor_temph.Caption) > 85 And Trim(valor_temph.Caption) < 170
Then
     If temph_amarillo.Visible = True Then
         temph_amarillo.Visible = False
     Else
         temph_amarillo.Visible = True
     End If
   Else: temph_amarillo.Visible = True
   End If
   If Trim(valor_temph.Caption) > 170 Then
     If temph_rojo.Visible = True Then
```

```
            temph_rojo.Visible = False
        Else
            temph_rojo.Visible = True
        End If
    Else: temph_rojo.Visible = True
    End If
    'alarmas para las temperaturas de la humedad
    If Trim(valor_humed.Caption) < 85 Then
        If humed_verde.Visible = True Then
            humed_verde.Visible = False
        Else
            humed_verde.Visible = True
        End If
    Else: humed_verde.Visible = True
    End If
    If Trim(valor_humed.Caption) > 85 And Trim(valor_humed.Caption) <
170 Then
        If humed_amarillo.Visible = True Then
            humed_amarillo.Visible = False
        Else
            humed_amarillo.Visible = True
        End If
    Else: humed_amarillo.Visible = True
    End If
    If Trim(valor_humed.Caption) > 170 Then
        If humed_rojo.Visible = True Then
            humed_rojo.Visible = False
        Else
            humed_rojo.Visible = True
        End If
    Else: humed_rojo.Visible = True
    End If
End If
End Sub


Private Sub retardo()
For j = 0 To 3225
  If retarda.Visible = True Then
    retarda.Visible = False
  Else
    retarda.Visible = True
  End If
Next j
```

```
End Sub
Private Sub retardote()
For j = 0 To 50
Call retardo
Next j
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   Me.WSocket.Close
   Unload HUMEDAD
   Unload TEMPERATURA
End Sub
Private Sub WSocket_DataArrival(ByVal bytesTotal As Long)
Dim sdatos As String
Dim npos As Integer
Dim ident As String
Dim valor As String
WSocket.GetData sdatos
npos = Val(Mid(sdatos, 1, 1))
If npos <> 4 Then
      s = sdatos
      'MsgBox s
      columna = InStr(1, s, "&", 1)
      cad1 = Left(s, columna - 1)
      valor_temph = cad1
      'MsgBox cad1
      s = Trim(Right(s, Len(s) - columna))
      columna = InStr(1, s, "&", 1)
      cad2 = Left(s, columna - 1)
      'MsgBox cad2
      valor_humed.Caption = cad2
      s = Trim(Right(s, Len(s) - columna))
      columna = InStr(1, s, "&", 1)
      cad3 = Left(s, columna - 1)
      'MsgBox cad3
      velocidad.Caption = cad3
      cad4 = Trim(Right(s, Len(s) - columna))
      Text2.Text = cad4
End If
If npos = 4 Then
      ident = Trim(UCase(Mid(sdatos, 2, npos)))
      valor = Trim(Mid(sdatos, 6, 250))
      MsgBox ident
       MsgBox valor
```

```
    If ident = "CHAT" Then List1.AddItem sdatos
End If

End Sub
```

ANEXO C

DIRECCIONES ELECTRONICAS EN DONDE CONSTA INFORMACIÓN TÉCNICA

C.1 SENSOR DE HUMEDAD C7600A

http://content.honeywell.com/sensing/prodinfo/humiditymoisture/

C.2 SENSOR DE TEMPERATURA LM35

http://www.national.com/pf/LM/LM35.html

C.3 OPTOCOPLADOR MOC3042

http://www.fairchildsemi.com/pf/MO/MOC3042-M.html

C.4 COMANDOS DE ESCRITURA Y LECTURA PARA PROGRAMACION DEL USB.

www.**delcom**-eng.com/downloads/USBPRGMNL.pdf