DISEÑO E IMPLEMENTACIÓN DE UN CONTROL AUTOMÁTICO PARA UNA MÁQUINA CORTADORA DE PIEZAS DE MADERA

NICOLÁS GÓMEZ VILLEGAS
VANESSA RODRÍGUEZ LORA

Proyecto de Grado para optar por el título de Ingeniero de Sistemas

Asesores
Germán Guzmán Rivera
José Luís Montoya Pareja

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2005

Nota de aceptación

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del Jurado

_____
Firma del Jurado

Medellín 4 de noviembre- de 2005

DEDICADA A...

Nicolás:
Dedico este proyecto a mis padres y hermanos por su constante ayuda y apoyo incondicional.

Vanessa:
A Dios por traer a mi vida aquellas personas y acontecimientos que me han ayudado a salir adelante, por permitirme alcanzar mis metas y mis objetivos. Aquí no termina mi camino, apenas comienza…

A mi madre por su apoyo y dedicación. Gracias por estar a mi lado y por sacrificarte por mí. No tengo palabras ni hechos que me ayuden a recompensar todo lo que has hecho por mí, Dios te pague por todo… .

A Guillermo, porque el respeto y el amor que se siente por un padre se gana con cariño y dedicación. Padre es aquel que lo da todo por ti y está a tu lado, eso eres para mi; mi padre.

A hermanita, mis hermanos, sobrinos y amigos por ayudarme y apoyarme.

A Jairo y Jorge Correa por todo el apoyo que me dieron durante mi formación académica, por ayudar a mi familia en los momentos difíciles. Siempre estarán en mis recuerdos y en mis oraciones.

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

Edwin Giraldo por su gran ayuda, su colaboración, paciencia y conocimiento. Su trabajo y orientación fue vital para la realización del proyecto. Sin él nada de esto sería posible.

A nuestros asesores Germán Guzmán por su apoyo, y a José Luís por sus acertados consejos.

Juan Francisco Cardona por sus recomendaciones, por su interés y por su deseo de hacer de este proyecto algo mejor.

Elizabeth Rendón por su iniciativa, por su soporte e indicaciones.

A nuestras familias, amigos y todas aquellas personas que estuvieron involucrados con este proyecto. A todos gracias por su ayuda, paciencia y colaboración.

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág.

GLOSARIO

AUTOMÁTICA: dicho de un mecanismo: Que funciona en todo o en parte por sí solo. Ciencia que trata de sustituir en un proceso el operador humano por dispositivos mecánicos o electrónicos.

CAD: *Computer Aided Design*. Diseño asistido por computador. Este ermite concretar las ideas que se inicialmente se plantean a partir de los croquis o bocetos, para luego obtener las planimetría y la realización de piezas o elementos.

CIRCUITO: conjunto de elementos que consumen o generan corriente eléctrica unidos por conductores por los que circula dicha corriente.

CNC: Control numérico computarizado. Es un sistema versátil que permite controlar el movimiento de las herramientas y las partes por los programas de computador que utilizan datos numéricos.

CONTROL: regulación, manual o automática, sobre un sistema. Tablero o panel donde se encuentran los mandos. Dispositivo que regula a distancia el funcionamiento de un aparato, mecanismo o sistema.

DIP: encapsulado en el cual el circuito se encuentra bajo una superficie que lo recubre.

DOO: Diseño Orientado a Objetos. Es un método de diseño que abarca el proceso de descomposición orientado a objetos y una notación para representar los modelos lógico y físico tal como los modelos estáticos y dinámicos del sistema bajo diseño.

DRIVER: Controlador. Este es un programa que interactúa con un dispositivo de hardware en particular. Contiene todos los datos necesarios del dispositivo para que el resto de programas y el sistema operativo sepan como han de utilizarlo.

EDA: (Electronic Design Automation). Diseño electrónico automatizado. Se dice de todas aquellas herramientas de software que tienen como finalidad del diseño de circuitos.

ENCODER: sensor electro-opto-mecánico que unido a un eje proporciona información de la posición angular. Su fin es actuar como un dispositivo de realimentación en sistemas de control integrado.

EOP: End of packet. Señal que indica que se ha llegado al final de un paquete de transmisión.

ESTATOR: parte fija de una máquina rotatoria por oposición a la parte móvil o rotor.

HANDSHAKE: confirmación. Comunicación básica entre dispositivos electrónicos para asegurar la transmisión de datos. Normalmente se necesitan dos líneas de control, adicionales a las de datos

HOST: anfitrión, puede ser el PC o un dispositivo

MHCN: Máquinas de herramientas de control numérico. Son aquellas máquinas que son controladas numéricamente y que poseen herramientas específicas para realizar el trabajo para el cual fueron diseñadas.

MICROCONTROLADOR: computadora completa situada en un único chip, que contiene todos los elementos del microprocesador básico además de otras funciones especializadas.

NRZI: Non Return to Zero Invert. código de no retorno a cero que invierte los ceros, introduciendo capacidad de sincronización en ausencia de transmisión.

PID/VID: Product ID/ Vendor ID. En USB son unos los descriptores que se emplean para el reconocimiento del dispositivo.

PIEZOELÉCTRICO: electricidad creada por una presión mecánica.

PROTOCOLO: conjunto de reglas predeterminadas que hacen posible el intercambio coordinado de mensajes entre usuarios, procesos, máquinas, esto incluye mecanismos de control de las relaciones entre las entidades comunicantes, la localización de los recursos y el flujo ordenado de la comunicación

PWM: modulador por ancho de pulsos. Un modulador por ancho de pulso (PWM) es un dispositivo que puede usarse para variar la velocidad de un motor, su sentido de giro y frenarlo

SISTEMA EMBEBIDO: sistemas que están diseñados alrededor de un microcontrolador que integra en un chip memoria de programa, memoria de datos y varias funciones periféricas.

USB: Universal Serial Bus. Interfaz de hardware que permite conectar periféricos de baja velocidad, como el teclado, el ratón, la impresora o cámaras digitales, a una computadora.

RESUMEN

Este documento es el compendio del trabajo realizado por algo más de un año con respecto al objetivo del proyecto al diseñar e implementar de un control automático para una máquina cortadora de piezas de madera. Este trabajo describe no solo los factores técnicos, sino que también hace referencia a las consideraciones teóricas que debieron tenerse en cuenta para el desarrollo del proyecto.

El capítulo uno contiene algunas bases teóricas sobre lo que son las máquinas de control numérico (CNC), de los componentes mecánicos más importantes en la máquina como lo son los motores y sobre los componentes electrónicos que se emplearon para el diseño de la tarjeta. También contiene los estándares para el diseño de circuitos electrónicos y hace una breve descripción de las herramientas que pueden emplearse con este propósito y justifica las razones que ayudaron a decidir cual de ellas emplear. También se incluye parte del trabajo realizado con el Protocolo USB y los elementos más importantes de este trabajo y que tuvieron mayor relevancia dentro del desarrollo de este proyecto. Los demás aspectos que no fueron desarrollados con profundidad en este trabajo se encuentran en la sección de anexos como ayuda al lector para que sirva de apoyo a futuros trabajos.

El capítulo dos hace una descripción general de la máquina Corbal 3000, su funcionamiento mecánico, así como también del viejo control. También muestra el trabajo realizado para el diseño del nuevo control, las consideraciones que se tomaron en cuenta para la implementación de este y la función que cumplirán algunos componentes electrónicos dentro del mismo.

El capítulo tres describe el trabajo que se realizó con el software de prueba tanto para la parte de comunicación PC- Control, sino que también describe las rutinas de movimiento que se desarrollaron para las figuras básicas como lo son líneas, diagonales y círculos. Se muestra el diseño de los programas y los algoritmos en los cuales se basó el desarrollo de las rutinas.

En el capítulo cuatro se presentan las conclusiones aportadas al trabajo desde los objetivos propuestos y desde el trabajo realizado durante el desarrollo del proyecto. En el capítulo cinco se muestran los que se consideran los futuros trabajos que podrían realizarse con la máquina Corbal y que se dejan como

sugerencias para futuros proyectos tanto para las áreas de Ingeniería de sistemas como mecánica.

Los resultados obtenidos están contenidos en el capítulo seis, allí se muestran los resultados más importantes del trabajo, tales como el control, el desarrollo de la comunicación a través del uso del protocolo USB, el trabajo con la nueva gama de microcontroladores y el uso de rutinas gráficas modificadas para trabajar con sistemas XY.

INTRODUCCIÓN

Las máquinas de control numérico se han venido empleando en un sin fin de aplicaciones, pero de un modo muy especial al área industrial. El contar con este tipo de máquinas ayuda a la industria a incrementar su producción y a aumentar sus factores de calidad.

Estas máquinas aunque posean una fundamentación matemática para poder realizar su trabajo requiere además de un diseño electrónico que la soporte y de sistemas de control óptimos que permitan potencializar sus beneficios.

El ámbito de la electrónica y del control digital se encuentran en constante cambio, se encuentran en busca de aplicaciones ilimitadas a todas las áreas en las cuales se pueda simplificar y facilitar el trabajo del hombre, haciendo que la tecnología empleada sea cada vez más ágil, confiable y de menor volumen al contar con el uso de microcontroladores.

Este proyecto se basa en estos principios para poner a disposición de la Ingeniería mecánica los fundamentos electrónicos que le permitan mejorar las máquinas diseñadas y para que estas a su vez ayuden al ser humano a optimizar su entorno.

Este proyecto tuvo como objeto diseñar e implementar un control que permitiera controlar una máquina CNC empleando un servicio de cortes específico para esta. Con esto se pretendía hacer más sencillo el control de la máquina y optimizar su uso. Adicionalmente se pretendía iniciar un nuevo campo de investigación sobre el control de máquinas y dispositivos empleando para ello comunicación USB con el PC.

Uno de los principales aportes que el proyecto hace es el control de máquinas y dispositivos a través del puerto de comunicación USB, lo que permite es estas se integren más fácilmente a las nuevas arquitecturas de computo.

DEFINICIÓN DEL PROBLEMA

Este proyecto nace como iniciativa de Elizabeth Rendón profesora de Ingeniería de Diseño y del Profesor Germán Guzmán para optimizar el control de una máquina cortadora de piezas de madera.

Esta máquina es producto del proyecto de grado de Elizabeth Rendón para optar por el titulo de Ingeniería Mecánica. Se desea que desde el punto de vista de Ingeniería de sistemas y muy especialmente desde el énfasis en control digital se diseñe y elabore un control que optimice el uso y funcionamiento de la máquina.

Actualmente el control de la máquina es poco eficiente, se manejan tarjetas de control para cada uno de los motores, así como también varias fuentes de voltaje. El uso de varios componentes y el volumen de todos estos hacen que el control de la máquina sea difícil, implicando además la poca estabilidad en las conexiones entre los mismos.

Fuera de diseñar un control en una tarjeta única evitando fallas entre las conexiones y empleando una sola fuente de alimentación se desea optimizar la comunicación de la máquina con un PC empleando el puerto USB y no el puerto paralelo que usa actualmente.

OBJETIVOS

Objetivo General

Diseñar un control que optimice el uso y el funcionamiento de la máquina cortadora de modelos en madera diseñada por Elizabeth Rendón en su proyecto de grado.

Objetivos Específicos

- Mejorar la comunicación de la máquina con el computador por medio del puerto USB, diseñando un protocolo de comunicación genérico.

- Investigar el protocolo de envío de datos del PC por medio del puerto USB para integrarlo en el control de la máquina.

- Buscar en el mercado los componentes óptimos que formarán parte del control de la máquina.

- Buscar un procesador o microcontrolador que permita manejar de manera eficiente los recursos de la máquina.

- Evaluar en el diseño del control la existencia de uno o múltiples microcontroladores.

- Diseñar un sistema embebido que entienda la forma de hacer los cortes.

- Diseñar un servicio de cortes que se comunique con el sistema embebido a través del protocolo USB.

IMPORTANCIA

Esta máquina fue inicialmente diseñada para facilitar el corte de piezas de madera (balso) para los aeromodelistas. El diseño actual del control de la máquina ha dificultado su uso y ha impedido que se aproveche su verdadero potencial.

El control permitirá que el uso de la máquina sea más sencillo al estar en una sola tarjeta de control evitando que se presenten las actuales fallas de conexión entre las tarjetas, empleando una sola fuente de alimentación eléctrica y mejorando considerablemente la comunicación con el PC por medio del puerto USB.

La máquina una vez puesta en funcionamiento óptimo puede ampliarse a diferentes áreas productivas como lo puede ser el corte de moldes en cuero para la talabartería y el dibujo de planos sencillos para el diseño.

1. MARCO TEORICO

1.1 MÁQUINAS DE CONTROL NUMÉRICO[1]

Existen cinco formas de automatizar en la industria moderna, de modo que se deberá analizar cada situación a fin de decidir correctamente el esquema más adecuado.

Los tipos de automatización son:

- Control Automático de Procesos: se refiere usualmente al manejo de procesos caracterizados de diversos tipos de cambios (generalmente químicos y físicos); un ejemplo de esto lo podría ser el proceso de refinación de petróleo.

- El Procesamiento Electrónico de Datos: frecuentemente es relacionado con los sistemas de información, centros de cómputo. Sin embargo en la actualidad también se considera dentro de esto la obtención, análisis y registros de datos a través de interfaces y computadores.

- La Automatización Fija: es aquella asociada al empleo de sistemas lógicos tales como los sistemas de relevadores y compuertas lógicas; sin embargo estos sistemas se han ido flexibilizando al introducir algunos elementos de programación como en el caso de los (PLC'S) o Controladores Lógicos Programables y compuertas lógicas programables.

- El Control Numérico Computarizado: un mayor nivel de flexibilidad lo poseen las máquinas de control numérico computarizado. Este tipo de control se ha aplicado con éxito a Máquinas de Herramientas de Control Numérico (MHCN). Entre las MHCN podemos mencionar: frezadoras CNC, tornos CNC, máquinas de Electroerosionado, máquinas de Corte por Hilo, etc.

- La Automatización Flexible: el mayor grado de flexibilidad en cuanto a automatización se refiere es el de los Robots industriales, que en forma más genérica, se les denomina como "Celdas de Manufactura Flexible".

[1] Tomado y adaptado de www.monografias.com

El CNC se utiliza para controlar los movimientos de una máquina por medio de números. Las máquinas y herramientas con control numérico se clasifican de acuerdo al tipo de operación de corte.

C.N.C. se refiere al control numérico de máquinas, generalmente estas de herramientas. Normalmente este tipo de control se ejerce a través de un computador y la máquina está diseñada a fin de obedecer las instrucciones de un programa dado. Esto se ejerce a través del siguiente proceso:

- Programación.
- Interfaz.
- Máquinas Herramientas C.N.C.

1.1.1 Ventajas de la MHCN

- Mayor precisión y mejor calidad de productos.
- Mayor uniformidad en los productos producidos.
- Un operario puede operar varias máquinas a la vez.
- Fácil procesamiento de productos de apariencia complicada.
- Flexibilidad para el cambio en el diseño y en modelos en un tiempo corto.
- Fácil control de calidad.
- Reducción en costos de inventario, traslado y de fabricación en los modelos y abrazaderas.

- Para la manufactura no se requieren operadores con experiencia.

- Se reduce la fatiga del operador.

- Mayor seguridad en las labores.

- Aumento del tiempo de trabajo en corte por maquinaria.

- Fácil control de acuerdo con el programa de producción lo cual facilita la competencia en el mercado.

- Fácil administración de la producción e inventario lo cual permite la determinación de objetivos o políticas de la empresa.

- Permite simular el proceso de corte a fin de verificar que éste sea correcto.

1.1.2 Desventajas

- Alto costo de la maquinaria.

- Falta de opciones o alternativas en caso de fallas.

- Es necesario programar en forma correcta la selección de las herramientas de corte y la secuencia de operación para un eficiente funcionamiento.

- Los costos de mantenimiento aumentan, ya que el sistema de control es más complicado y surge la necesidad de entrenar al personal de servicio y operación.

- Es necesario mantener un gran volumen de producción a fin de lograr una mayor eficiencia de la capacidad instalada.

A continuación se presentan los componentes mecánicos y electrónicos de la máquina. Esto se hace con el objeto que comprendiendo su comportamiento y usabilidad se pueda hacer un mejor trabajo en el diseño e implementación del nuevo control para la máquina.

1.2 MOTORES PASO A PASO[2]

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

La característica principal de estos motores es el hecho que se pueden mover 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 fijados en una posición o bien totalmente libres. Si una o más de sus bobinas están energizadas, el motor estará fijado en la posición correspondiente y por el contrario quedará completamente libre si no circula corriente por ninguna de las bobinas.

1.2.1 Principio de funcionamiento Básicamente estos motores están constituidos por un rotor sobre el que van aplicados distintos imanes permanentes y por un cierto número de bobinas excitadoras en su estator.

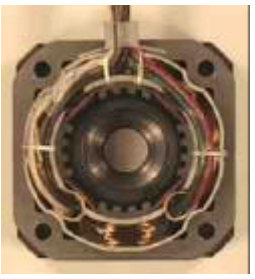
Las bobinas son parte del estator y el rotor es un imán permanente. Toda la conmutación (o excitación de las bobinas) deber ser externamente manejada por un controlador.

[2] Tomado de www.todorobot.com.ar

Figura 1. Rotor

Fuente www.todorobot.com.ar

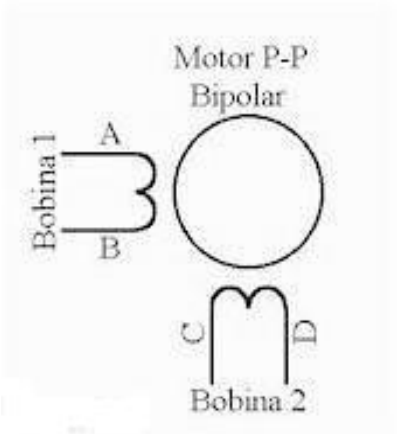
Figura 2. Estator de 4 bobinas



Fuente www.todorobot.com.ar

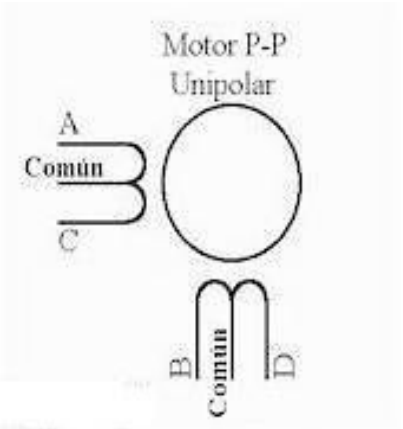
Existen dos tipos de motores paso a paso de imán permanente: los unipolares y bipolares. Ya que la máquina emplea motores paso a paso unipolares se hará una descripción más detallada de estos y su funcionalidad.

Figura 3. Motor Bipolar



Fuente www.todorobot.com.ar

Figura 4. Motor Unipolar

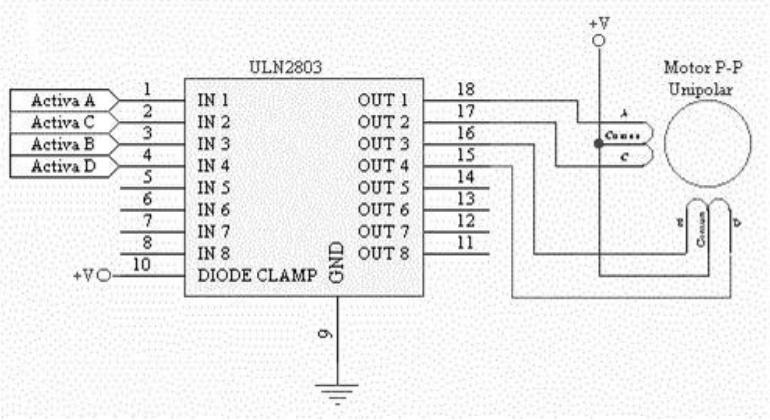


Fuente www.todorobot.com.ar

1.2.2 Unipolar Se caracterizan porque La polaridad en sus bobinas no cambia durante su funcionamiento. Estos motores suelen tener 6 ó 5 cables de salida, dependiendo de su conexión interna (ver figura 4). Este tipo se caracteriza por ser más simple de controlar. En la figura 5 se puede ver un ejemplo de conexión para controlar un motor paso a paso unipolar mediante el uso de un controlador de motores paso a paso, el cual es un arreglo de 8 transistores tipo Darlington capaces de manejar cargas del orden de los 500mA. Las entradas de activación (Activa A, B, C y D) pueden ser directamente gobernadas por un microcontrolador.

Este controlador no fue usado para el diseño, se emplearon en su lugar transistores tipo Darlington ya que estos son más económicos y más fáciles de adquirir en el mercado.

Figura 5. Conexión para control de motores Unipolares



Fuente www.todorobot.com.ar

1.2.3. 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. Para revertir el sentido de giro, simplemente se deben ejecutar las secuencias en modo inverso.

- Secuencia Normal: Esta es la secuencia más conocida. 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.

Tabla 1. Secuencia Normal Motores Unipolares

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**

Fuente www.todorobot.com.ar

Secuencia del tipo onda drive: En esta secuencia se activa solo una bobina a la vez lo que permite en algunos motores un funcionamiento más suave. La

contrapartida es que al estar solamente una bobina activada, el torque de paso y retención es menor.

Tabla 2. Secuencia Normal Motores Unipolares

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

Fuente www.todorobot.com.ar

Secuencia del tipo medio paso: En esta secuencia se activan las bobinas de tal forma que se pueda brindar un movimiento igual a la mitad del paso real. Para ello se activan primero 2 bobinas y luego solo 1 y así sucesivamente. Como vemos en la tabla la secuencia completa consta de 8 movimientos en lugar de 4.

Tabla 3. Secuencia de medio paso

PASO	Bobina A	Bobina B	Bobina C	Bobina D
1	**ON**	OFF	OFF	OFF
2	**ON**	**ON**	OFF	OFF
3	OFF	**ON**	OFF	OFF
4	OFF	**ON**	**ON**	OFF
5	OFF	OFF	**ON**	OFF
6	OFF	OFF	**ON**	**ON**
7	OFF	OFF	OFF	**ON**
8	**ON**	OFF	OFF	**ON**

Fuente www.todorobot.com.ar

Debido a que los motores paso a paso son dispositivos mecánicos y como tal deben vencer cierta inercia, 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. 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. Para la programación de las rutinas se empleó sin embargo, una frecuencia uniforme, ya que el poco peso de la herramienta es irrelevante para realizarlo haciendo el aumento de ésta.

1.3 INTERRUPTOR DE FIN DE CARRERA[3]

Su elemento básico son los microinterruptores que son dispositivos utilizados para detectar determinados posicionamientos en multitud de maquinaria. Su uso muchas veces es más económico que utilizar sensores, si es que la aplicación no lo requiere expresamente. La tecnología de fabricación ha evolucionado muy positivamente, reduciendo tamaños, aumentado cargas tanto mecánicas como eléctricas.

Estos son de muy diversas formas pero todos se basan en la operación por medio de un actuador mecánico. Este actuador mecánico mueve a su vez una lengüeta metálica en donde están colocados los contactos eléctricos, y los abre o cierra de acuerdo con la disposición física de estos contactos.

[3] Tomado de http://menbers.fortunecity.es

Desde el punto de vista eléctrico son extremadamente simples, ya que consisten en uno o varios juegos de contactos con cierta capacidad de conducción a cierto voltaje. Estos contactos pueden ser de apertura instantánea o lenta, y de contactos de operación traslapada o de abre y cierra.

Figura 6. Microinterruptores

Fuente www.marbelonline.com

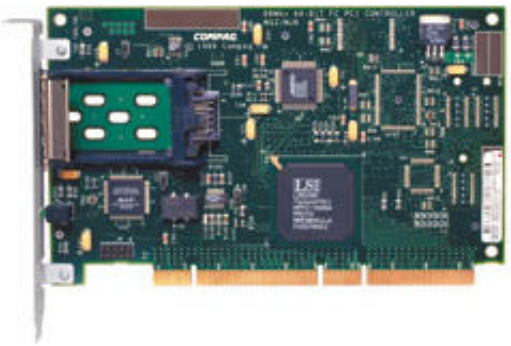
1.4 TARJETAS CONTROLADORAS[4]

Todos los dispositivos periféricos, tanto internos como externos necesitan valerse de algún medio para comunicarse entre ellos y las computadoras. Algunas veces les llaman controladores, interfaces, puertos o adaptadores.
Básicamente un controlador es un traductor entre la CPU y el dispositivo periférico como discos duros, disquete, teclado o monitor. Estos controladores ejecutan las siguientes funciones:

- Aíslan el equipo de los programas.
- Adecuan las velocidades entre los dispositivos que operan a diferentes velocidades.
- Convierten datos de un formato a otro.

[4] Tomado de http://www.electronica2000.250x.com

Figura 7. Tarjeta Controladora



Fuente www.aldirsa.com

1.5 DESCRIPCIÓN DE COMPONENTES ELECTRÓNICOS

1.5.1. Microcontroladores[5] Un microcontrolador es un circuito integrado de alta escala de integración que incorpora la mayor parte de los elementos que configuran un computador. Un microcontrolador dispone normalmente de los siguientes componentes:

- Procesador o UCP (Unidad Central de Proceso).
- Memoria RAM para Contener los datos.
- Memoria para el programa tipo ROM/PROM/EPROM.
- Líneas de E/S para comunicarse con el exterior.
- Diversos módulos para el control de periféricos (temporizadores, Puertas Serie y Paralelo, CAD: Conversores Analógico/Digital, CDA: Conversores Digital/Analógico, etc.).
- Generador de impulsos de reloj que sincronizan el funcionamiento de todo el sistema.

[5] Tomado de www.monografias.com

Los productos que para su regulación incorporan un microcontrolador disponen de las siguientes ventajas:

- Aumento de prestaciones: un mayor control sobre un determinado elemento representa una mejora considerable en el mismo.

- Aumento de la fiabilidad: al reemplazar el microcontrolador por un elevado número de elementos disminuye el riesgo de averías y se precisan menos ajustes.

- Reducción del tamaño en el producto acabado: La integración del microcontrolador en un chip disminuye el volumen.

- Mayor flexibilidad: las características de control están programadas por lo que su modificación sólo necesita cambios en el programa de instrucciones.

El microcontrolador es en definitiva un circuito integrado que incluye todos los componentes de un computador. Debido a su reducido tamaño es posible montar el controlador en el propio dispositivo al que gobierna. En este caso el controlador recibe el nombre de controlador embebido.

Los microcontroladores están siendo empleados en multitud de sistemas presentes en la vida diaria, como pueden ser juguetes, electrodomésticos, computadoras, impresoras, módems, vehículos, instrumentación electrónica, control de sistemas. Una aplicación típica es emplear varios microcontroladores para controlar pequeñas partes de un sistema. Estos pequeños controladores pueden comunicarse entre ellos y con un procesador central (sistema maestro - esclavo), probablemente más potente, para compartir la información y coordinar sus acciones.

A la hora de escoger el microcontrolador a emplear en un diseño concreto hay que tener en cuenta multitud de factores, como la documentación y herramientas de desarrollo disponibles, su precio, la cantidad de fabricantes que lo producen y por supuesto las características del microcontrolador (tipo de memoria de programa, número de temporizadores, interrupciones, etc.). Los factores más importantes son:

- Costo: Como es lógico, los fabricantes de microcontroladores compiten duramente para vender sus productos.

- Aplicación. Antes de seleccionar un microcontrolador es imprescindible analizar los requisitos de la aplicación:

  - Procesamiento de datos: puede ser necesario que el microcontrolador realice cálculos críticos en un tiempo limitado. En ese caso se debe asegurar seleccionar un dispositivo suficientemente rápido para ello. Por otro lado, habrá que tener en cuenta la precisión de los datos a manejar: si no es suficiente con un microcontrolador de 8 bits, puede ser necesario acudir a microcontroladores de 16 ó 32 bits, o incluso a hardware de punto flotante. Una alternativa más barata y quizá suficiente es usar librerías para manejar los datos de alta precisión que se cargan en la memoria de éste.

  - Entrada/Salida: para determinar las necesidades de Entrada/Salida del sistema es conveniente dibujar un diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar. Una vez realizado este análisis puede ser necesario añadir periféricos o hardware externo o cambiar a otro microcontrolador más adecuado a ese sistema.

  - Consumo: algunos productos que incorporan microcontroladores están alimentados con baterías y su funcionamiento puede ser tan vital como activar una alarma antirrobo. Lo más conveniente en un caso como éste puede ser que el microcontrolador esté en estado de bajo consumo pero que despierte ante la activación de una señal (una interrupción) y ejecute el programa adecuado para procesarla.

  - Memoria: para detectar las necesidades de memoria de nuestra aplicación debemos separarla en memoria volátil (RAM), memoria no volátil (ROM, EPROM, etc.), la memoria Flash y la memoria no volátil modificable (EEPROM). Este último tipo de memoria puede ser útil para incluir información específica de la aplicación como un número de serie o parámetros de calibración. El tipo de memoria a emplear vendrá determinado por el volumen de ventas previsto del producto: de menor a mayor volumen será conveniente emplear EPROM, OTP y ROM. En cuanto a la cantidad de memoria necesaria puede ser imprescindible realizar una versión preliminar (aunque sea en

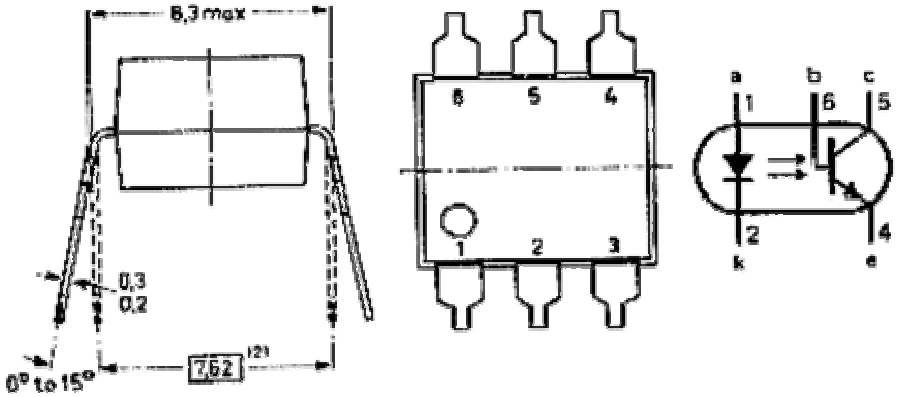
pseudo-código) de la aplicación y a partir de ella hacer una estimación de cuánta memoria volátil y no volátil es necesaria y si es conveniente disponer de memoria no volátil modificable.

- Ancho de palabra: el criterio de diseño debe ser seleccionar el microcontrolador de menor ancho de palabra que satisfaga los requerimientos de la aplicación. Usar un microcontrolador de 4 bits supondrá una reducción en los costes importante, mientras que uno de 8 bits puede ser el más adecuado si el ancho de los datos es de un byte. Los microcontroladores de 16 y 32 bits, debido a su elevado costo, deben reservarse para aplicaciones que requieran sus altas prestaciones (Entrada/Salida potente o espacio de direccionamiento muy elevado).

- Diseño de la tarjeta: la selección de un microcontrolador específico condicionará el diseño de la tarjeta de circuito. Debe tenerse en cuenta que quizá usar un microcontrolador barato puede encarecer el resto de componentes del diseño.

1.5.2. Optoacopladores[6] Un optoacoplador es un componente formado por la unión de al menos un emisor (diodo LED) y un fotodetector (fototransistor u otro) acoplados a través de un medio conductor de luz, puede ser encapsulado o de tipo discreto. Todos estos elementos se encuentran dentro de un encapsulado que por lo general es del tipo DIP.

Figura 8. Estructura externa e interna de un optoacoplador

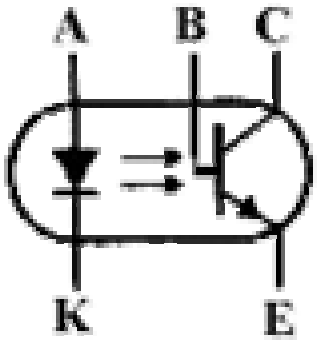


Fuente www.redeya.com

[6] Tomado de www.redeya.com

La señal de entrada es aplicada al fotoemisor y la salida es tomada del fotorreceptor. Los optoacopladores son capaces de convertir una señal eléctrica en una señal luminosa modulada y volver a convertirla en una señal eléctrica. La gran ventaja de un optoacoplador reside en el aislamiento eléctrico que puede establecerse entre los circuitos de entrada y salida.

Figura 9. Esquema del Optoacoplador



Fuente www.redeya.com

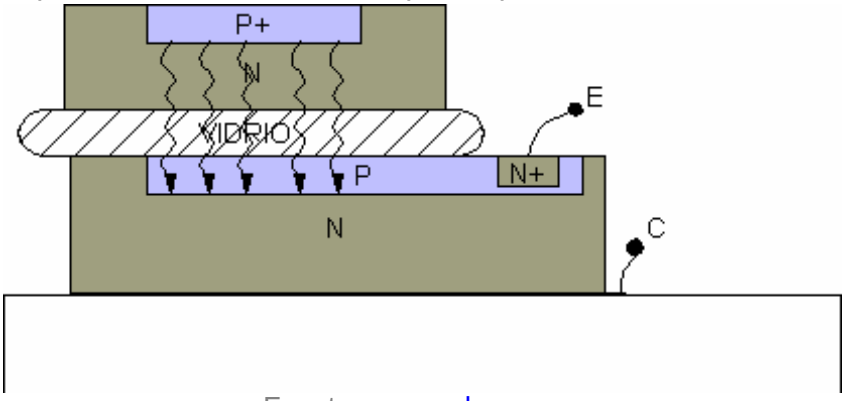
Los fotoemisores que se emplean en los optoacopladores de potencia son diodos que emiten rayos infrarrojos (IRED) y los fotorreceptores pueden ser tiristores o transistores.

Cuando aparece una tensión sobre los terminales del diodo IRED, este emite un haz de rayos infrarrojo que transmite a través de una pequeña guía-ondas de plástico o cristal hacia el fotorreceptor. La energía luminosa que incide sobre el fotorreceptor hace que este genere una tensión eléctrica a su salida. Este responde a las señales de entrada, que podrían ser pulsos de tensión.

Cuanta mayor intensidad atraviesa el fotodiodo, mayor será la cantidad de fotones emitidos y, por tanto, mayor será la corriente que recorra el fototransistor. Se trata de una manera de transmitir una señal de un circuito eléctrico a otro.

Las implementaciones de un optoacoplador son variadas y dependen de la casa que los fabrique. Una de las más populares se ve en la Figura 10. Se puede observar como el LED, en la parte superior, emite fotones que, tras atravesar el vidrio, inciden sobre el fototransistor

Figura 10 Esquema constructivo de un optoacoplador



Fuente www.redeya.com

- Diferentes tipos de Optoacopladores

o Fototransistor: se compone de un optoacoplador con una etapa de salida formada por un transistor BJT. Se trata de un transistor bipolar sensible a la luz.

El funcionamiento de un fototransistor viene caracterizado por los siguientes puntos:

- Un fototransistor opera, generalmente sin terminal de base (Ib=0) aunque en algunos casos hay fototransistores tienen disponible un terminal de base para trabajar como un transistor normal.

- La sensibilidad de un fototransistor es superior a la de un fotodiodo, ya que la pequeña corriente fotogenerada es multiplicada por la ganancia del transistor.

o Fototriac: se compone de un optoacoplador con una etapa de salida formada por un triac

o Fototriac de paso por cero: Optóacoplador en cuya etapa de salida se encuentra un triac de cruce por cero. El circuito interno de cruce por cero conmuta al triac sólo en los cruce por cero de la corriente alterna.

1.5.3 Transistores[7] Este componente electrónico es utilizado como amplificador u oscilador en sistemas de comunicación, control y computación.

Figura 11. Transistor

Fuente www.electrokit.com

Este es un dispositivo semiconductor activo que tiene tres o más electrodos, los tres electrodos principales son emisor, colector y base. La conducción entre estos electrodos se realiza por medio de electrones y huecos.

El germanio y el silicio son los materiales más frecuentemente utilizados para la fabricación de los elementos semiconductores. Los transistores pueden efectuar prácticamente todas las funciones de los antiguos tubos electrónicos, incluyendo la amplificación y la rectificación, con muchísimas ventajas. Sus elementos son:

- Emisor: que emite los portadores de corriente,(huecos o electrones). Su labor es la equivalente al cátodo en los tubos de vacío.

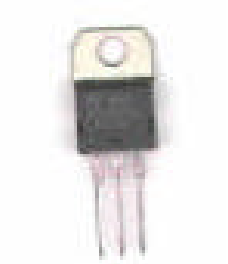
- Base: que controla el flujo de los portadores de corriente. Su labor es la equivalente a la rejilla en los tubos de vacío.

- Colector: que capta los portadores de corriente emitidos por el emisor. Su labor es la equivalente a la placa en los tubos de vacío.

- Transistores Darlington- Tip122: Estos son transistores que se unen para multiplicar su ganancia de corriente y poder manejar valores más amplios de corriente a partir de corrientes muy pequeña, son utilizados para suministrare potencia a un circuito principalmente.

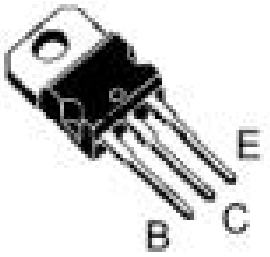
[7] Tomado de www.solomantenimiento.com

Figura 12. TIP 122



Fuente ECG

Figura 13. Pines del TIP 122



Fuente ECG

1.5.4. Resistencias[8] Componente fabricado específicamente para ofrecer un valor determinado de resistencia al paso de la corriente eléctrica.
Las resistencias de carbón son el tipo más utilizado. Son de pequeño tamaño y baja disipación de potencia. Según el proceso de fabricación y su constitución interna, se puede distinguir:

- Resistencias Aglomeradas: También se conocen con el nombre de "composición", debido a su constitución: una mezcla de carbón, materia aislante, y resina aglomerante. Variando el porcentaje de estos componentes se obtienen los distintos valores de resistencias.

- Resistencias de Capa de Carbón: En este tipo de resistencias, la fabricación está basada en el depósito de la composición resistiva sobre un cuerpo tubular formado por materiales vítreos cerámicos.

Figura 14. Resistencia de Carbón

Fuente www.ugr.es

Las resistencias siguen además un código de colores para su lectura, con estos se puede precisar su valor en Ohmnios, su símbolo es ?. En la sección de anexos se puede encontrar la tabla con este código de colores y se indica como es su lectura.

[8] Tomado de www.solomantenimiento.com

1.5.5. Condensadores[9] Este dispositivo se encarga de almacenar energía eléctrica. Los condensadores son componentes pasivos diseñados con el fin de almacenar energía electrostática o presentar una capacidad eléctrica determinada, es decir, son componentes pasivos de dos terminales en los que la intensidad que los atraviesa (aparentemente) es proporcional a la variación de tensión existente entre sus terminales respecto al tiempo. Su unidad de medida en el S.I (sistema internacional) es el Faradio aunque por las limitaciones características de los mismos se usan distintos submúltiplos (micro, µ / nano, n / pico, p).

Desde el punto de vista constructivo, un condensador está constituido por dos placas conductoras separadas por un material dieléctrico. En su interior se establece un campo eléctrico, sin pérdida de energía, como consecuencia de la polarización dieléctrica (no confundir material aislante y dieléctrico, todos los dieléctricos son aislantes, pero no todos los aislantes son dieléctricos; los dieléctricos son materiales no conductores en los que resulta posible su polarización). La capacidad de un condensador va a depender del tamaño de sus placas, de la distancia que las separa y del material del que está formado el dieléctrico.

Existen condensadores de tipo fijo o variable. Estos son:

- Condensadores fijos: su valor capacitivo no se puede alterar.

- Condensadores variables: se puede modificar su capacidad dentro de unos márgenes determinados.

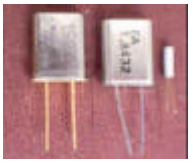
Figura 15. Condensadores

Fuente www.anatronic.com

[9] Tomado de www.solomantenimiento.com

1.5.6 Cristales de cuarzo[10] El cristal de cuarzo es utilizado como componente de control de la frecuencia de circuitos osciladores convirtiendo las vibraciones mecánicas en voltajes eléctricos a una frecuencia específica.

Figura 16. Cristales de Cuarzo



Fuente www.planetaelectronico.com

Esto ocurre debido al efecto "piezoeléctrico". La piezo-electricidad es electricidad creada por una presión mecánica. En un material piezoeléctrico, al aplicar una presión mecánica sobre un eje, dará como consecuencia la creación de una carga eléctrica a lo largo de un eje ubicado en un ángulo recto respecto al de la aplicación de la presión mecánica.

En algunos materiales, se encuentra que aplicando un campo eléctrico según un eje, produce una deformación mecánica según otro eje ubicado a un ángulo recto respecto al primero.

Por las propiedades mecánicas, eléctricas, y químicas, el cuarzo es el material más apropiado para fabricar dispositivos con frecuencia estable o constante.

Estos cristales crean frecuencias que permiten el funcionamiento del circuito. Estos actúan creando una oscilación que se convierte en corriente alterna que es la que sincroniza al microcontrolador.

El circuito eléctrico equivalente que se muestra a continuación es un esquema del cristal de cuarzo trabajando a una determinada frecuencia de resonancia. El capacitor Co o capacidad en paralelo, representa en total la capacidad entre los electrodos del cristal más la capacidad de la carcaza y sus terminales. R1, C1 y L1 conforman la rama principal del cristal y se conocen como componentes o parámetros donde:

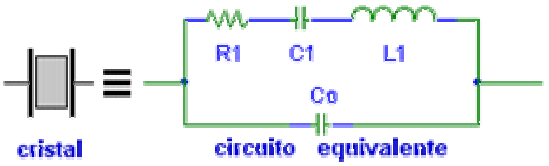
[10] Tomado de http://huarpe.com

- L1 representa la masa vibrante del cristal

- C1 representa la elasticidad del cuarzo y

- R1 representa las pérdidas que ocurren dentro del cristal.

Figura 17. Circuito eléctrico equivalente



Fuente http://huarpe.com

1.5.7 Diodos[11] Es un componente electrónico que solo permite el paso de corriente eléctrica en un solo sentido.

La principal aplicación del diodo es la obtención de una tensión continua a partir de una fuente de corriente alterna lo cual ocurre porque deja circular corriente a través suyo cuando se conecta el polo positivo de la fuente al ánodo, y el negativo al cátodo, y se opone al paso de la misma si se realiza la conexión opuesta de forma que realiza así la conversión de corriente alterna en continua al permitir solo el paso de la alternancia positiva. A este proceso se le llama "rectificación".

Figura 18. Diodos

Fuente www.cnice.mecd.es

[11] Tomado de www.solomantenimiento.com

El diodo semiconductor está formado básicamente por una unión P-N, con un terminal conectado a cada uno de los contactos metálicos de sus extremos y con una cápsula capaz de alojar al conjunto, de la que salen los terminales correspondientes al ánodo "P" positivo y al cátodo "N" negativo.

Existen diferentes tipos de diodos, los más conocidos son:

- De frecuencia de línea: El voltaje de conducción para el que se fabrica es aquel tan pequeño como sea posible con un trr (tiempo de retorno inverso) muy alto, para aplicaciones a la frecuencia de 60 Hz. Con especificaciones de voltaje de bloqueo de varios kilovoltios y corriente de varios kiloamperios. Se conectan en serie o en paralelo.

- De recuperación rápida: Utilizados en circuitos de alta frecuencia combinados con transistores. Soporta niveles de potencia de algunos cientos de voltios y de amperios.

- Diodos LED (Emisores de Luz): Diodo que cuando se le aplica tensión, polarizado directamente, emite luz. Se fabrica con un compuesto formado por Galio, Arsénico y Fósforo. Empleado en aparatos electrónicos como indicador luminoso, prácticamente esta presente en televisores, equipos de música etc., el color dependerá del material en que se ha fabricado.

- Fotodiodo: Diodo que se vuelve conductor si está polarizado directamente al recibir luz, se utiliza como sensor de mandos a distancia emisores de rayos infrarojos, sería lo contrario a un LED.

1.6 DISEÑO CIRCUITOS

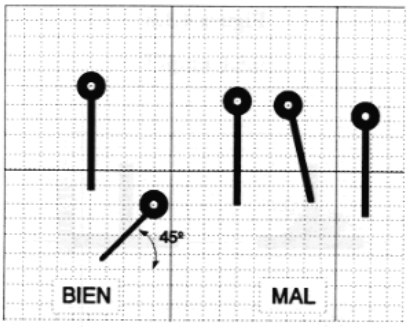
Siguiendo el objetivo general de este proyecto se diseñó un control que optimiza el uso y el funcionamiento de la máquina cortadora de piezas de madera, teniendo un control conformado por una tarjeta única que evita que se presenten fallas de conexión en el sistema y haciendo que el uso de la máquina se simplifique, siguiendo los siguientes estándares.

1.6.1 Estándares para los circuitos impresos[12] El diseño de circuitos impresos debe seguir una serie de estándares que pretenden mantener la integridad de las señales, evitando la producción de interferencias.

Estos estándares son unas reglas simples definidas por la IEEE, que básicamente hablan de la disposición de los componentes, de la distancia entre estos y el ancho de las pistas conductoras.
Estas reglas son:

- Se diseñará sobre una hoja cuadriculada en décimas de pulgada, de modo que se hagan coincidir las pistas con las líneas de la cuadrícula o formando un ángulo de 45º con éstas, y los puntos de soldadura con las intersecciones de las líneas

Figura 19. Ubicación de los componentes



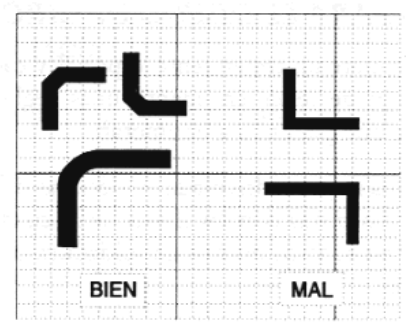
Fuente www.lu1dma.com.ar

- El diseño debe ser lo más sencillo posible, entre más cortas sean las pistas y más simple la distribución de componentes, mejor será el diseño.

- No deben haber pistas con ángulos de 90º; cuando sea preciso efectuar un giro en una pista, se hará con dos ángulos de 135º (Fig 20); si es necesario ejecutar una bifurcación en una pista, se hará suavizando los ángulos con sendos triángulos a cada lado (Fig. 21).

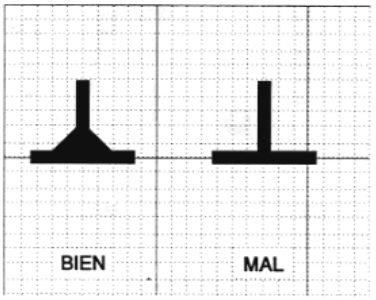
[12] Tomado de www.lu1dma.com.ar

Figura 20. Ángulos en las pistas



Fuente www.lu1dma.com.ar

Figura 21. Bifurcación



Fuente www.lu1dma.com.ar

- Los puntos de soldadura deben ser círculos cuyo diámetro será, al menos, el doble del ancho de la pista que en él termina.

- El ancho de las pistas dependerá de la intensidad que vaya a circular por ellas. Para esto se tiene los siguientes datos:

Tabla 4. Ancho de Pistas

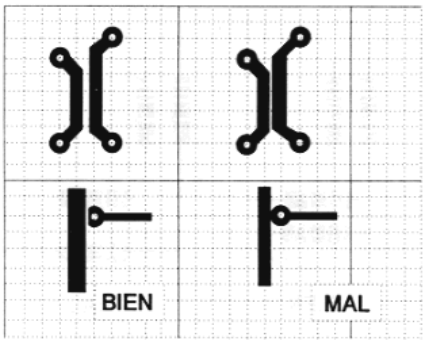
Ancho (milímetros)	Corriente (Amperios)
0.8 mm	2 A
2 mm	5 A
4.5 mm	10 A

Fuente www.lu1dma.com.ar

En general, se realizan pistas de unos 2 mm aproximadamente.

- Entre pistas próximas y entre pistas y puntos de soldadura, se observará una distancia que dependerá de la tensión eléctrica que se prevea existirá entre ellas; como norma general, se dejará una distancia mínima de unos 0,8 mm.; en casos de diseños complejos, se podrá disminuir los 0,8 mm hasta 0,4 mm. En algunas ocasiones será preciso cortar una porción de ciertos puntos de soldadura para que se cumpla esta norma.

Figura 22. Distancia entre pistas

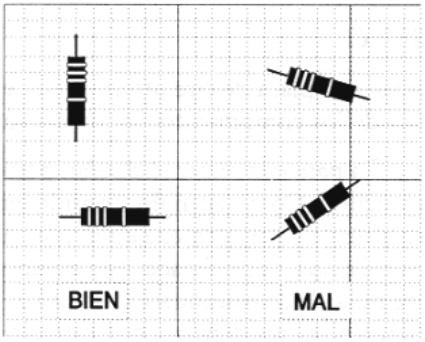


Fuente www.lu1dma.com.ar

- La distancia mínima entre pistas y los bordes de la placa será de dos décimas de pulgada, aproximadamente unos 5 mm.

- Todos los componentes se colocarán paralelos a los bordes de la placa.

Figura 23. Posición de los componentes



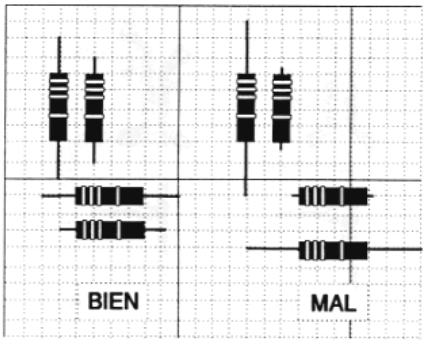
Fuente wwwlu1dma.com.ar

- No se podrán colocar pistas entre los bordes de la placa y los puntos de soldadura de terminales de entrada, salida o alimentación, exceptuando la pista de masa.

- No se pasarán pistas entre dos terminales de componentes activos (transistores, tiristores, etc.).

- Como norma general, se debe dejar, una o dos décimas de pulgada de patilla entre el cuerpo de los componentes y el punto de soldadura correspondiente.

Figura 24. Soldadura de los componentes



Fuente www.u1dma.com.ar

Es importante indicar que para diseñar un circuito impreso, es necesario conocer el tamaño y la forma física de los componentes, o, mejor aún, disponer de ellos.

1.7. HERRAMIENTAS PARA EL DISEÑO DE CIRCUITOS ELECTRÓNICOS[13]

Los grandes cambios que ha sufrido la industria electrónica en los últimos años, ha sido en parte posible al uso de herramientas de diseño asistido por computadora (CAD) y en forma especifica a herramientas de diseño automático tipo EDA.

El uso de éste tipo de herramientas fuera de simplificar el trabajo de diseño ha acelerado los procesos de este mismo. Esto se tradujo en un cambio de metodología para el diseño y evolución de los circuitos electrónicos.

Existen gran número de herramientas que apoyan el diseño de circuitos electrónicos, las cuales poseen entre si características que las diferencian las unas de las otras. En ésta sección se hace una breve descripción de dos herramientas, las cuales se encuentran disponibles en la Universidad, estas son *Circuit Maker* y *HiWire II.*

[13] Apartes tomados de www.forosdeelectronica.com

1.7.1 HiWire Esta herramienta gráfica para el diseño de circuitos electrónicos posee características de una herramienta CAD. Esta herramienta de diseño permite crear rejillas en diferentes sistemas métricos, lo que permite adaptar los diseños de acuerdo a las necesidades. Es una herramienta poco usada actualmente por los diseñadores de circuitos ya que corre bajo DOS.

1.7.2 Circuit Maker Actualmente es el programa para el diseño de circuitos que más se está empleando en la universidad ya que su uso es más sencillo y posee características que optimizan el trabajo, por eso se adoptará a esta herramienta para el diseño de la tarjeta controladora del proyecto. Adicionalmente las empresas dedicadas a la impresión de estas tarjetas trabajan con este software lo que facilita su elaboración. A continuación se describe la herramienta con mayor detalle.

- Diseño de esquemas: Permite crear esquemas con una gran simplicidad. Después de haber insertado los componentes en el espacio de trabajo, la conexión entre patillas del circuito se simplifica gracias al software. Éste permite conectar las patillas de un componente a las patillas del circuito al cual se quiere conectar encargándose del trazado de la conexión. También se pueden trazar pistas manualmente y cuando se desplaza un circuito, los trazados de pistas se vuelven a editar automáticamente.

- Soporta la simulación lógica, analógica o mixta del circuito: Gracias al motor de simulación, Circuitmaker permite simular realmente los esquemas teniendo en cuenta las características de los componentes (retrasos de propagación, tiempo de espera, etc.). De esta manera se puede simular cualquier combinación de componentes lógicos y analógicos sin insertar convertidores D/A o A/D. Posee también instrumentos virtuales para la comprobación y análisis del circuito.

- Biblioteca de componentes: Gracias al editor de símbolos, Circuit maker permite crear nuevos símbolos e incluirlos en las librerías. No existen limitaciones de patillas, de esta forma se podrán crear símbolos para microprocesadores y otros tipos de componentes mas complejos. También tiene la posibilidad de editar los parámetros de los componentes y de modificarlos en todo momento. Permite además la importación y exportación de otras librerías

1.8. PROTOCOLO DE COMUNICACIÓN USB (UNIVERSAL SERIAL BUS)[14]

Empezar de cero con USB puede ser algo complicado debido a al volumen de la especificación 2.0 que contiene 650 paginas, y está es sólo el principio de una larga lista de estándares asociados a USB, también están los estándares de la clases de USB como la clase HID y si se está diseñando un *host* hay hasta tres estándares de interfaz de control para escoger, ninguno de ellos detallado en la especificación 2.0.

Es por esto que se tratará de hacer una breve introducción al USB; es necesario aclarar que en este texto se referirá al estándar 1.1 de USB porque es el usado por la mayoría de los microcontroladores.

El USB versión 1.1 soporta dos velocidades, una alta a 12Mbits/segundo y una baja a 1.5Mbits/segundo. La baja es menos susceptible a las interferencias, reduciendo así el costo de componentes especializados para disminuir las mismas.

El *Universal Serial Bus* (USB) es controlado por un *Host* y solo puede haber un *Host* por bus, la especificación no define ningún diseño con múltiples maestros, pero en la especificación 2.0 se introduce un protocolo de negociación de *Host*, el cual permite a dos dispositivos negociar el papel de *Host* y está enfocado a conexiones punto a punto.

El USB usa una topología de estrella parecida a la de la red *ethernet 10baseT* esto impone el uso de un *hub*, esta topología permite monitorear el poder de cada dispositivo y puede ser hasta apagado si ocurre un sobre voltaje, esto sin interrumpir el desempeño de los demás dispositivos, puede soportar dispositivos de velocidades diferentes, el *hub* filtra las transacciones de máxima velocidad para que los dispositivos lentos no las reciban.

USB como lo sugiere su nombre es un bus serial. Usa 4 cables blindados de los cuales dos son poder (+5V y GND) los dos restantes son un par trenzado de señales diferenciales de datos. USB usa un sistema de codificación NRZI (*Non Return To Zero Invert*) para enviar los datos con un campo de sincronización para sincronizar los relojes del *host* y del dispositivo.

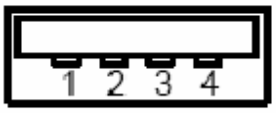
[14] Traducido de www.beyondlogic.org

USB soporta plug and play con *drivers* que se cargan y descargan automáticamente. El usuario simplemente conecta el dispositivo al bus, el *host* detecta esta adición, interroga al nuevo dispositivo y carga el *driver* apropiado, el usuario no tiene que preocuparse por ningún dato de configuración, simplemente usa su dispositivo y lo desconecta cuando termine, una vez el host detecta la ausencia descarga el *driver* automáticamente.

La carga del *driver* apropiado se hace usando la combinación PID/VID (Product ID / Vendor ID) donde los VID son proporcionados por el foro de desarrolladores de USB por un precio y esto es considerado como un punto critico para USB.

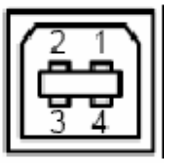
1.8.1 Conectores Todos los dispositivos tienen una conexión hacia el Host y el Host una conexión hacia los dispositivos. Los conectores en el Host y en el dispositivo no son iguales, hay dos tipos de conectores llamados A y B.

Figura 25. Pines del conector USB Tipo A



Fuente www.beyondlogic.com

Figura 26. Pines del conector USB Tipo B



Fuente www.beyondlogic.com

Conectores del tipo A se encuentran en los *Host* y en los *hub*, los conectores tipo B se encuentran en los dispositivos.
Los cables de USB usan alambres internos de color.

Tabla 5. Estándar Cables internos del USB

Número de Pin	Color del Cable	Función
1	**Rojo**	VBUS (5 voltios)
2	**Blanco**	D-
3	**Verde**	D+
4	**Negro**	Tierra

Fuente www.beyondlogic.com

1.8.2 Parte Eléctrica USB usa un par trenzado de transmisión diferencial para los datos codificado usando NRZI y está empacado para garantizar la adecuada transición en la corriente de datos. Para transmitir un uno diferencial D+ 2.8V con una resistencia de 15K? conectada a tierra y D- 0.3Vcon una resistencia de 1.5K? conectada a 3.6V y un cero diferencial es el contrario en los pines con las mismas resistencias conectadas a positivo y negativo.

El receptor define un uno (1) diferencial como D+ 200 mV más alto que D- y un cero diferencial D+ 200 mV menor que D-. La polaridad de la señal se invierte dependiendo de la velocidad del bus, por ende se usan los términos 'J' y 'K' para representar los niveles lógicos, en baja velocidad 'J' es un cero diferencial y en máxima velocidad 'J' es un uno diferencial.

1.8.3 Descriptores de USB Todos los dispositivos de USB tienen una jerarquía de descriptores que describe al Host información como que es el dispositivo, quien lo hace, que versión de USB usa, de cuantas maneras puede ser configurado, cuantos endpoint tiene y que tipo son.

Los descriptores de USB más comunes son

- Descriptor de dispositivo.
- Descriptor de configuración.
- Descriptor de interfaz.
- Descriptor de *endpoint*.
- Descriptor de cadena.

Los dispositivos USB solo pueden tener un descriptor de dispositivo. El descriptor de dispositivo incluye, información como cual es la versión de USB la identificación de producto y vendedor es usada, para cargar los *driver* apropiados y el número posible de configuraciones que un dispositivo puede tener. El número de configuraciones indica cuantas ramas de descriptores de configuración siguen.

El descriptor de configuración especifica valores tales como la cantidad de voltaje que la configuración usa, si el dispositivo usa poder propio o es energizado por el bus y el número de interfaces que tiene. Cuando el dispositivo es enumerado, el *Host* lee el descriptor del dispositivo y que configuración puede usar, solo puede usar una configuración al tiempo.

El descriptor de interfaz puede ser visto como una cabecera o un grupo funcional de *endpoint* desempeñando una sola característica del dispositivo. A diferencia de los descriptores de configuración, no hay limitación para tener un solo descriptor de interfaz activo al tiempo. Un dispositivo puede tener uno o varios descriptores de interfaz activos al tiempo.

Los descriptores de interfaz tienen un campo **bInterfaceNumber** especificando el número de interfaz y otro **bAlternateSetting** que permite a la interfaz cambiar la configuración al vuelo. Por ejemplo se puede tener un dispositivo como dos interfaces, la interfaz uno y la interfaz dos. La interfaz uno tiene el **bInterfaceNumber** = 0 indicando que es el primer descriptor de interfaz y el **bAlternateSetting** = 0.

La interfaz dos tendrá un **bInterfaceNumber** = 1 indicando que es la segunda interfaz y un **bAlternateSetting** = 0. Entonces se puede hacer otro descriptor, también con un **bInterfaceNumber** = 1 indicando que es la segunda interfaz, pero con **bAlternateSetting** = 1, indicando que este descriptor de interfaz puede ser alternativo a al otro descriptor de interfaz dos.

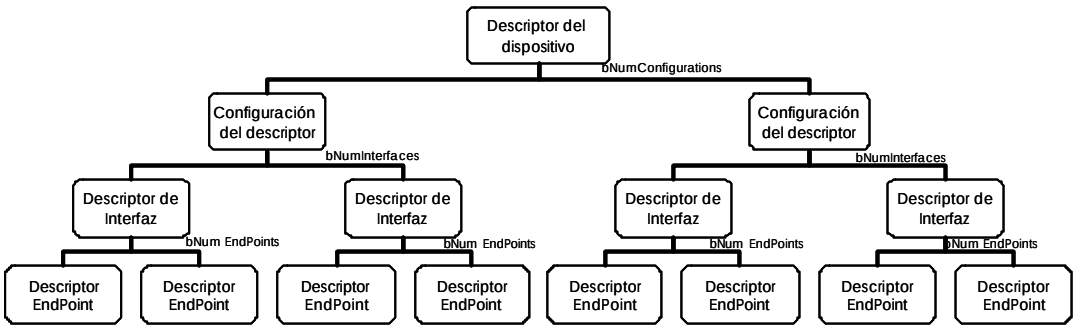
Cuando se activa esta configuración los primeros dos descriptores de interfaz tendrán un **bAlternateSetting** = 0. No obstante durante la operación el *Host* puede enviar una petición *SetInterface* dirigida a la interfaz uno con una configuración alternativa de uno para activar el otro descriptor de interfaz.

Esto da ventaja a tener dos configuraciones, en las cuales se puede estar transmitiendo datos por la interfaz cero mientras que se cambia la configuración del *endpoint* asociado con la interfaz uno sin afectar a la interfaz cero.

Cada descriptor de *endpoint* se usa para especificar el tipo de transferencia, dirección, intervalo de actualización y tamaño máximo de paquete para cada

endpoint. El *endpoint* cero, es el *endpoint* de control por defecto por lo tanto nunca tendrá un descriptor.

Figura 27. Descriptores USB



Fuente www.beyonlogic.com

- Composición de los Descriptores de USB: Todos los descriptores están hechos con un formato en común. El primer byte especifica el tamaño del descriptor, mientras que el segundo indica el tipo de descriptor. Si el tamaño del descriptor es menor al especificado, el *Host* deberá ignorarlo, pero si es mayor al tamaño especificado el *Host* ignorará los bytes extra y buscará el siguiente descriptor al final del tamaño actual devuelto.

Tabla 6. Composición de los descriptores USB

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño de la descripción en bytes
1	bDescriptorType	1	Constante	Tipo de descripción
2	bcdUSB	2	BCD	Inicio de parámetros para el descriptores USB

Fuente www.beyondlogic.com

El descriptor de dispositivo para USB solo puede tener una descripción. Especifica una información básica e importante tal como la versión de USB, el tamaño máximo de paquete, la identificación del producto y del vendedor y el número de configuraciones posibles para el dispositivo, el formato del descriptor del dispositivo se muestra a continuación.

Tabla 7. Formato de los descriptores USB

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en Bytes (18 bytes)
1	bDescriptorType	1	Constante	Descriptor del dispositivo (0x01)
2	bcdUSB	2	BCD	Número de especificación del dispositivo que cumple.
4	bDeviceClass	1	Clase	Código de clase Si es igual a cero cada interfaz específica su propio código de clase. Si es igual a 0xFF, el código de clase lo especifica el vendedor. De otro modo el campo es un código de clase válido.
5	bDeviceSubClass	1	SubClase	Código de subclase (Asignado por USB Org)
6	bDeviceProtocol	1	Protocolo	Código de Protocolo (Asignado por USB Org)
7	bMaxPacketSize	1	Número	Tamaño máximo del paquete para *Zero Endpoint*. Tamaños válidos 8, 16, 32, 64
8	idVendor	2	ID Vendedor	ID (Asignado por USB Org)
10	idProduct	2	ID Producto	ID (Asignado por el fabricante)
12	bcdDevice	2	BCD	Número de liberación del dispositivo.
14	iManufacturer	1	Index	Indice de manufactura. Cadena de descriptor.
15	iProduct	1	Index	Indice de Producto. Cadena de Descriptor.
16	iSerialNumber	1	Index	Índice del número serial. Cadena de descriptor.
17	bNumConfigurations	1	Entero	Número de posible configuración.

Fuente www.beyondlogic.com

- **bcdUSB** este campo reporta la versión más alta de USB que soporta el dispositivo. El valor es un decimal codificado en binario con el siguiente formato 0xJJMN donde JJ es el mayor número de versión, M es el menor número de versión y N es el sub-menor número de versión. Así USB 2.0 se reporta 0x0200, USB 1.1 0x0110 y USB 1.0 0x0100.

- **bDeviceClass, bDeviceSubClass** y **bDeviceProtocol** son usados por el sistema operativo para encontrar la clase de *driver* para el dispositivo. Normalmente solo **bDeviceClass** está en el nivel de dispositivo. La mayoría de las especificaciones de clase escogen identificarse al nivel de interfaz y como resultado le dan a **bDeviceClass** el valor 0x00. Lo que permite a un dispositivo soportar varias clases.

- **bMaxPacketSize** reporta el tamaño máximo de paquete para el *endpoint* 0. Todos los dispositivos deben soportar *endpoint* 0.

- **idVendor** y **idProduct** para que el sistema operativo encuentre un *driver* para el dispositivo la identificación de vendedor es asignada por USB-IF.

- **bcdDevice** tiene el mismo formato que **bcdUSB** y es usado para proveer un número de versión para el dispositivo. Este valor es asignado por el desarrollador.

- Tres descriptores de cadena existen para dar detalles sobre el manufacturador, producto y número serial. No es necesario tener los descriptores de cadena. Si no hay descriptor presente, un índice de cero debe ser usado.

- **bNumConfigurations** define el número de configuraciones que el dispositivo soporta con su velocidad actual.

- Descriptor de Configuración: Un dispositivo USB puede tener muchas configuraciones diferentes pero la mayoría de los dispositivos son simples y tienen solo una. El descriptor de configuración especifica como es energizado el dispositivo, cual es el consumo máximo de energía, el número de interfaces que tiene. Es posible tener dos configuraciones, una cuando el dispositivo es energizado por el bus y otra cuando tiene fuente propia. Como esta es una cabecera para los descriptores de interfaz, también es posible tener una

configuración usando un modo de transferencia diferente al de otra configuración.

Una vez que todas las configuraciones han sido examinadas por el *Host*, el *Host* enviará un comando *SetConfiguration* con un valor diferente de cero que iguale al valor de **bConfiguration** de una de las configuraciones. Esto se hace para seleccionar la configuración deseada.

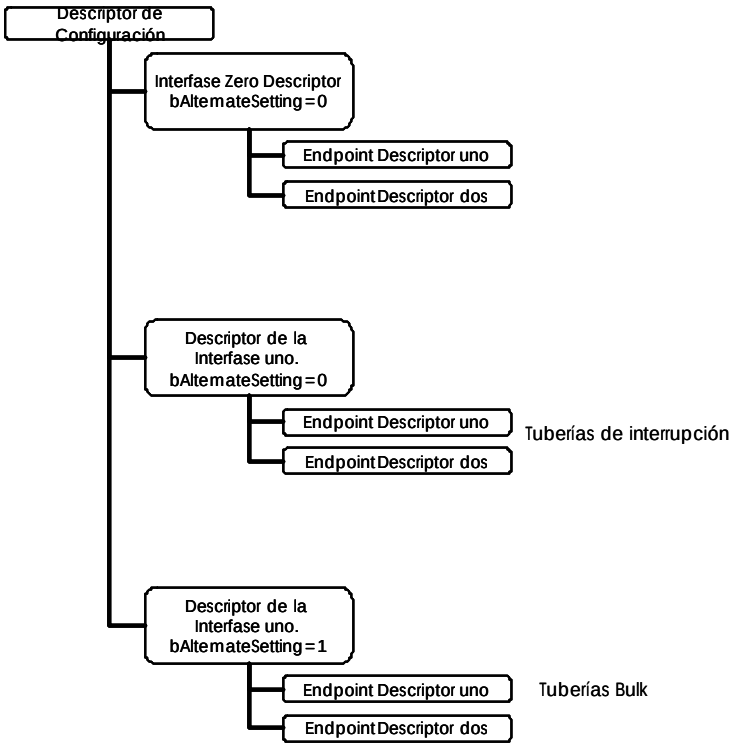
Tabla 8. Descriptor de configuración

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes.
1	bDescriptorType	1	Constante	Descriptor de configuración (0x02)
2	wTotalLenght	2	Número	Longitud total del dato de retorno.
4	bNumInterfaces	1	Número	Número de interfazs.
5	bConfigurationValue	1	Número	Valor para usar como un argumento para seleccionar esta configuración.
6	iConfiguration	1	Index	Índice de la cadena del descriptor. Para describir la configuración.
7	bmAttributes	1	Bitmap	D7 Poder del Bus D6 Poder propio. D5 Alarma remota D4..0 Reservado. (0)
8	bMaxPower	1	mA	Máximo consumo de potencia.

Fuente www.beyondlogic.com

Cuando se ha leído el descriptor de configuración, devuelve toda la jerarquía de configuración que incluye todos los descriptores relacionados, de interfaz y de *endpoint*. El campo **wTotalLength** refleja el número de bytes en la jerarquía.

Figura 28. Jerarquía de Descriptores



Fuente www.beyondlogic.com

- **bNumInterfaces** especifica el número de interfaces presentes para esta configuración.

- **bConfigurationValue** es usado por la petición *SetConfiguration* para seleccionar esta configuración.

- **iConfiguration** es el índice de un descriptor de cadena describiendo la configuración en lenguaje humanamente entendible.

- **bmAttributes** especifica los parámetros de poder para la configuración. Si un dispositivo tiene fuente propia, pone D6. El bit D7 en USB 1.0 es usado para indicar un dispositivo energizado por el bus, pero esto es hecho ahora por **bMaxPower**. Si un dispositivo utiliza energía del bus, tenga o no fuente propia, debe reportar el consumo de energía en **bMaxPower**. Los dispositivos pueden

también soportar inicio remoto lo que permite al dispositivo despertar al *Host* cuando el *Host* está suspendido.

- **bMaxPower** define el poder o energía máxima tomada por el dispositivo del bus. Esta se define en unidades de 2mA, con un máximo de aproximadamente 500mA. La especificación no permite a un dispositivo de alto poder energizado por el bus tomar más de 500mA del *Vbus*. Si un dispositivo pierde poder externo, no debe tomar más de lo indicado en **bMaxPower**. Debe acabar todas las operaciones que no puede hacer sin el poder externo.

- Descriptor de Interfaz El descriptor de interfaz puede ser visto como una cabecera o un grupo funcional de *endpoint* desempeñando una sola característica del dispositivo. El descriptor de interfaz contiene el siguiente formato.

Tabla 9. Descriptor de interfaz

Offset	Campo	Tamaño	Valar	Descripción
0	bLength	1	Número	Tamaño del descriptor en bytes.
1	bDescriptorType	1	Constante	Descriptor de interfaz (0x04)
2	bInterfaceNumber	1	Número	Número de interfaz.
3	bAlternateSetting	1	Número	Valor usado para seleccionar configuraciones alternativas.
4	bNumEndpoints	1	Número	Número de EndPoints usados para esta interfaz.
5	bInterfaceClass	1	Clase	Código de clase. (Asignado por USB Org)
6	bInterfaceSubClass	1	SubClase	Código de subclase. (Asignado por USB Org)
7	bInterfaceProtocol	1	Protocolo	Código de Protocolo
8	iInterface	1	Index	Índice de la cadena del descriptor que describe esta interfaz.

Fuente www.beyondlogic.com

- **bInterfaceNumber** indica el índice del descriptor de interfaz. Debe iniciar en cero y ser incrementado una vez por cada nuevo descriptor de interfaz.

- **bAlternativeSetting** es usado para especificar interfaces alternativas. Estas interfaces son seleccionadas con la petición *SetInterface*.

- **bNumEndpoints** indica el número de *endpoint* usados por la interfaz. Este valor debe excluir el *endpoint* 0 y es usado para indicar el número de descriptores de *endpoint* a seguir.

- **bInterfaceClass, bInterfaceSubClass** y **bInterfaceProtocol** es usado para indicar las clases soportadas (HID, comunicaciones, almacenamiento masivo) esto permite que muchos dispositivos usen la misma clase de *driver* impidiendo la necesidad de escribir *driver* específicos.

- **Interfaz** permite una descripción de cadena de la interfaz.

- Descriptor de Endpoint: El descriptor de *endpoint* es usado para describir *endpoint* diferentes al *endpoint* 0. El *endpoint* 0 siempre está definido para ser un *endpoint* de control y es configurado antes de que los descriptores sean pedidos. El bus va a usar la información devuelta por estos descriptores para determinar los requerimientos de ancho de banda en el bus.

Tabla 10. Descriptor End Points

FOCET	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en Bytes (7 bytes)
1	bDescriptionType	1	Constante	Descriptor Endpoint (0x05)
2	bEndpointAddress	1	Endpoint	Dirección Endpoint decodificada como sigue: 0..3b: Número Endpoint 4..6b:Reservado Set to Zero 7b: Dirección (Ignorada por el control de Endpoints) 0= Salida Endpoint

				1 = Entrada Endpoint
3	bmAttributes	1	Bitmap	Bits 0..1 Tipo de transferencia 00 = Control 01 = Asincrónica 10 = Bulk 11 = Interrupción Bits 2..7 están reservados si el endpoint es asincrónico. Bits 3..2 = Tipo sincrónico (Modo Iso) 00 = No Sincronización 01 = Asincrónico 10 = Adaptativo 11 = Sincrónico Bits 5..4 = tipo de uso (Modelos Iso) 00 = Endpoint de dato 01 = Endpoint de Feedback 10 = Endpoint explícito de Feedback de datos 11 = Reservado
4	wMaxPacketSize	2	Número	Tamaño máximo del paquete. Este endpoint es capaz de enviar o recibir.
6	bInterval	1	Número	Intervalos para conteo de transferencias de datos del Endpoint. Valor en cuadro. Conteos ignorados para los endpoint de bulk y control. Iso debe ser igual a 1 y el campo debe tener un rango de 1 a 255 para interrupción de los endpoint.

Fuente www.beyondlogic.com

- **bEndpointAddress** indica cual *endpoint* está describiendo el descriptor.

- **bmAttributes** indica el tipo de transferencia. Puede ser de control, interrupción, sincrónica o *Bulk*. Si se especifica un *endpoint* sincrónico atributos

adicionales pueden ser seleccionados, tales como el tipo de sincronización y usos.

- **wMaxPacketSize** indica el tamaño de datos máximo en el *endpoint*.

- **bInterval** se usa para especificar el intervalo de activación para ciertas transferencias. Las unidades están expresadas en marcos, lo cual es igual a 1ms para velocidad baja o alta y 125us para máxima velocidad.

- Descriptor de cadena: Los descriptores de cadena proveen información que puede ser leída por un humano y son opcionales. Si no se usan todos los índices de descriptores de cadena deben ser puestos en 0 indicando que no hay ningún descriptor de cadena.

Las cadenas están codificadas en formato unicode y puede haber productos que soporten múltiples lenguajes. El índice de cadena 0 debe devolver una lista de los lenguajes soportados.

Tabla 11. Descriptor de cadena

Offset	Campo	Tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en Bytes
1	bDescriptorType	1	Constante	Cadena del descriptor(0x03)
2	wLANGIS[0]	2	Número	Lenguaje soportado código Zero (Por ejemplo 0x0409 Inglés-Estados Unidos)
3	wLANGIS[1]	2	Número	Lenguaje soportado código Zero (Por ejemplo 0x0c09 Inglés-Australia)

4	wLANGIS[2]	2	Número	Lenguaje soportado código Zero (Por ejemplo 0x0407 Alemán – estándar)

Fuente www.beyondlogic.com

El descriptor de cadena definido arriba muestra el formato del descriptor de cadena 0. El *Host* tiene que leer este descriptor para determinar que lenguajes hay disponibles. Si un lenguaje es soportado, puede ser referenciado enviando la identificación del lenguaje en el campo **wIndex** de la petición *Get Descriptor* (cadena).

Todas las cadenas subsecuentes toman el siguiente formato.

Tabla 12. Formato de Cadenas subsecuentes

FOCET	Campo	tamaño	Valor	Descripción
0	bLength	1	Número	Tamaño del descriptor en Bytes
1	bDescriptorType	1	Constante	Cadena del descriptor (0x03)
2	bString	N	Unicode	Cadena decodificada Unicode

Fuente www.beyondlogic.com

En la sección de anexos se complementa esta información con otros conceptos del protocolo USB y su usabilidad.

2. DESCRIPCIÓN TÉCNICA DEL PROYECTO

2.1 DESCRIPCIÓN DE LA MÁQUINA Y ANTIGUO CONTROL

La máquina cortadora de piezas de madera “Corbal 3000” es una máquina de control numérico que posee tres ejes coordenados (XYZ). Cada uno de estos ejes es movido por motores paso a paso y como herramienta de corte en el eje Z se tiene un moto-tool, la cual es una herramienta rotativa multipropósito y que es la más usada para este tipo de proyectos.

La máquina corta figuras geométricas usando los datos enviados desde el computador e invocando las rutinas de movimiento en el microcontrolador, que las traduce en señales que mueven los motores.

El control de la máquina originalmente manejaba una tarjeta de control para cada uno de los motores paso a paso con su respectiva fuente de voltaje. En la Figura 30 se puede apreciar el tamaño del control (fondo) con relación a la máquina. Fue precisamente su gran volumen y poca confiabilidad, especialmente con respecto a las conexiones la principal causa que originó este proyecto que pretende diseñar un control más óptimo que simplifique el funcionamiento de la máquina, siendo más confiable y seguro en su funcionamiento ya que evita tener gran número de conexiones entre diferentes tarjetas, evitando que se pierdan datos o pasos durante el proceso de corte de las piezas.

Figura 29. Máquina Cortadora

Imagen proporcionada por Elizabeth Rendón

Figura 30. Máquina cortadora y control Original

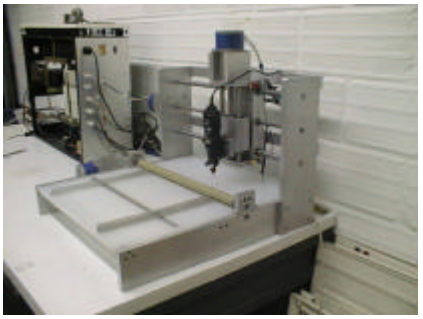


Imagen proporcionada por Elizabeth Rendón

A continuación se muestran las características mecánicas de la máquina Corbal.

Tabla 13. Especificaciones de la máquina Corbal 3000

Área de trabajo Ancho: 485 mm (19") Largo: continuo Altura:128 mm (5") **Velocidad máxima de avance en Vacio** Eje X: 16.362 mm/seg. (0.644 in/seg) Eje Y: 17.638 mm/seg. (0.694 in/seg) Eje Z: 17.638 mm/seg. (0.694 in/seg) **Motores** Tres motores de paso de 110 Oz.in. NEMA 34. **Controlador** Tarjetas de motores secuenciadas con microcontrolador 16F84. **Precisión** Eje X: 0.0334 mm (0.00131") Eje Y y Z: 0.03531 mm (0.00139") **Material a Cortar** Balsa de espesores de 1 a 3mm **Peso** Peso Neto: 64lbs. (32 Kg.)	**Dimensiones de la Máquina** Ancho: 31" (787.4mm) Largo: 25" (635mm) Altura 23" (584.2mm) **Herramienta** Moto- Tool Dremel. Hobby Kit 5950 **Velocidad de la Herramienta** 5000 a 30000 R.P.M **Especificaciones Eléctricas** 110 V A.C 60 Hz **Comunicación** A través del Puerto paralelo del computador. **Cable** Un Cable para puerto paralelo de 2 mts de longitud **Programa** AeroCam Ver. 1.0 **Requerimientos del Sistema** Procesador 586 o mayor con 64 MB RAM

Fuente Elizabeth Rendón

A continuación se describen algunos componentes de la máquina, ya que el conocimiento de su funcionamiento es esencial para el desarrollo del control y de los programas que manejarán la máquina.

2.1.1 Ejes

Figura 31. Posición de los ejes

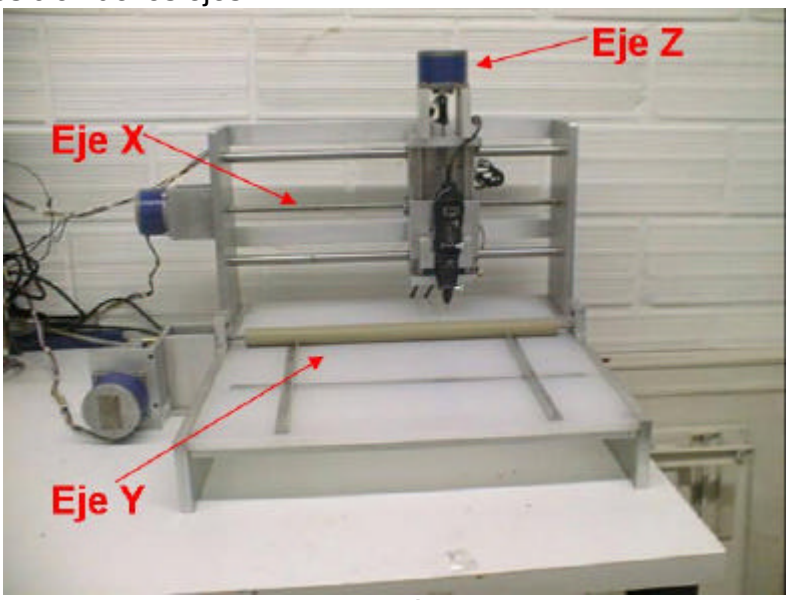


Imagen proporcionada por Elizabeth Rendón

La máquina está diseñada para realizar cortes en 2D gracias a su sistema de ejes coordenados XYZ. Los ejes permiten que el movimiento de los motores paso a paso que originen el corte de figuras básicas. Cada uno de los ejes tiene tareas específicas dentro del funcionamiento general de la máquina.

- Eje X: Esta representado por un largo tornillo el cual con el movimiento del motor paso a paso permite que la herramienta de corte se desplace hacia la izquierda o hacia la derecha para realizar los cortes (Fig 32).

- Eje Y: Se encarga de mover el rodillo de alimentación de material, permitiendo avanzar o retroceder según sea necesario (Fig 33)

- Eje Z: Este eje es el que maneja la herramienta de corte Motor Tool. El movimiento del motor permite que esta se mueva hacia arriba o hacia abajo evitando el desperdicio de material (Fig 34).

Figura 32. Motor de movimiento del eje X

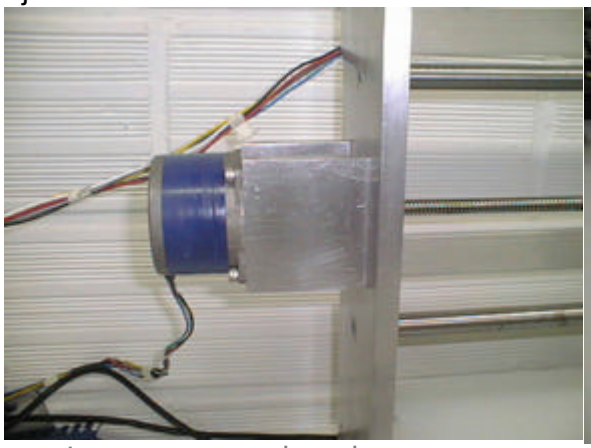


Imagen proporcionada por Elizabeth Rendón

Figura 33. Motor de movimiento del eje Y

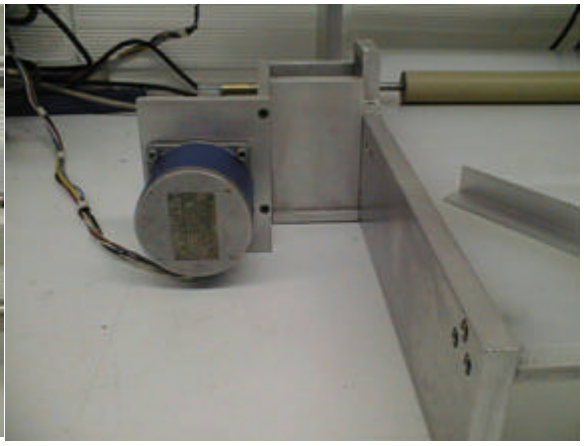


Imagen proporcionada por Elizabeth Rendón

Figura 34. Motor de movimiento del eje Z

Imagen proporcionada por Elizabeth Rendón

2.1.2 Motores usados en la Máquina La máquina posee tres motores paso a paso, cada uno de estos encargados de mover los ejes XYZ. Los motores paso a paso son de fácil control, pueden utilizarse sin encoders que le indiquen su posición por lo tanto son ideales para máquinas que requieren de gran precisión. Su control se limita a un determinado número de pasos que este debe dar energizando sus bobinas por medio de una rutina simple diseñada en el microcontrolador (Fig 35).

Figura 35. Motor paso a paso

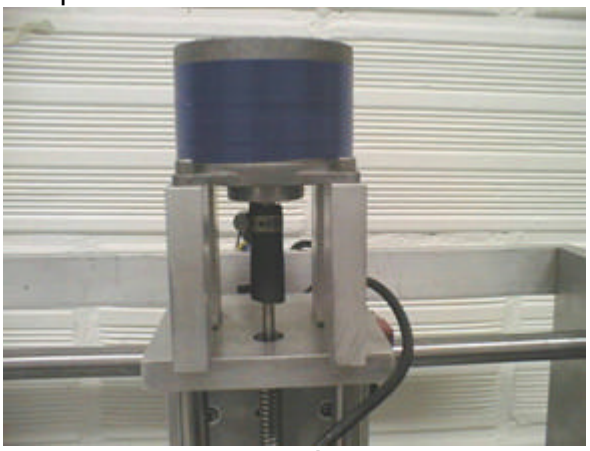


Imagen proporcionada por Elizabeth Rendón

Cada motor tiene seis cables, dos comunes y cuatro más que llevan corriente a las bobinas, los cables están agrupados así

Tabla 14. Distribución cables Motor paso a paso

Comunes	Bobinas
Blanco	Azul(1a)
	Amarillo(2a)
Negro	Gris (1b)
	Rojo(2b)

Fuente Vanessa Rodríguez Lora

A continuación se muestran las rutinas de movimiento empleadas para el movimiento de los motores a la derecha y a la izquierda.

Tabla 15. Secuencia de movimiento a la derecha

Orden	1ª	1b	2a	2b
1	0	0	0	1
2	1	0	0	0
3	0	1	0	0

4	0	0	**1**	0

Fuente Vanessa Rodríguez Lora

Tabla 16. Secuencia de movimiento a la izquierda

Orden	1ª	1b	2a	2b
1	0	0	0	**1**
2	0	1	**1**	0
3	0	**1**	0	0
4	**1**	0	0	0

Fuente Vanessa Rodríguez Lora

Para la implementación de una de las rutinas del sistema de cortes (Capítulo tres) fue necesario implementar la secuencia de medio paso para que está permitiera tener figuras mejor definidas.

Tabla 17. Secuencia de movimiento medio paso

PASO	1a	1b	2a	2b
1	**1**	0	0	0
2	**1**	**1**	0	0
3	0	**1**	0	0
4	0	**1**	**1**	0
5	0	0	**1**	0
6	0	0	**1**	**1**
7	0	0	0	**1**
8	**1**	0	0	**1**

Fuente Vanessa Rodríguez Lora

2.1.3 Microinterruptores En la máquina estos le indican a los microcontroladores si el eje se encuentra al inicio o al final de su recorrido, evitando que la máquina se dañe tratando de avanzar o retroceder más allá de sus limites, para esto se deben tener dos microinterruptores por cada uno de los ejes a controlar (Fig 36).

Figura 36. Microinterruptores máquina

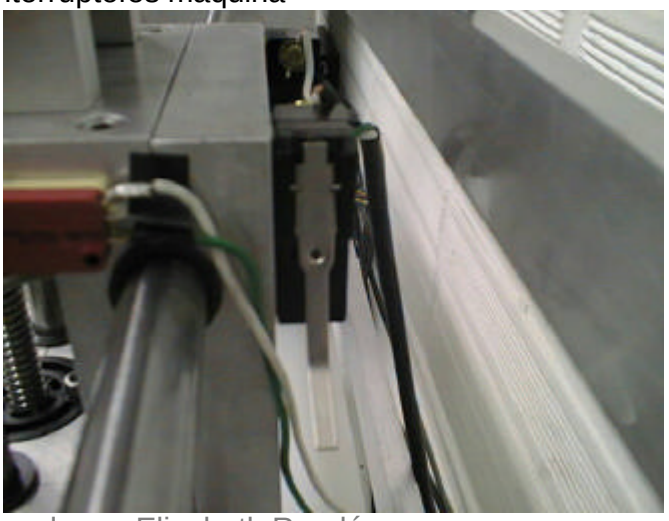


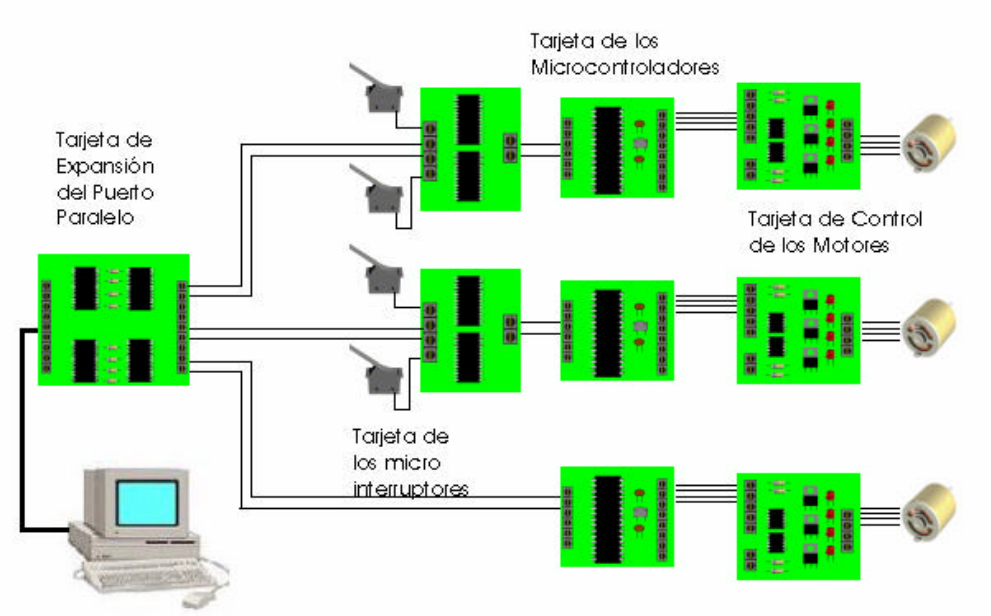
Imagen proporcionada por Elizabeth Rendón

2.1.4 Tarjetas de control La máquina originalmente empleaba diferentes tipos de tarjetas para su control. El hecho de tener muchas tarjetas implicaba que existieran problemas de conexión entre estas, ocasionando fallas en el funcionamiento de la máquina. Las tarjetas que se emplearon fueron:

- Tarjeta de control de motores paso a paso.
- Tarjeta para micro controlador PIC 16F84
- Tarjeta de control de microinterruptor
- Tarjeta reguladora de voltaje.

A continuación se muestra el diagrama completo del control original, donde se muestran todas las tarjetas empleadas. Este diagrama evidencia más claramente todas las posibles fallas de conexión y comunicación entre las tarjetas ocasionando el mal funcionamiento de la máquina y su difícil manejo (Fig 37). Adicionalmente en la sección de anexos se presentan en detalle cada una de las tarjetas empleadas en el montaje y su función dentro de este.

Figura 37. Flujo de señales en las tarjetas de control



Elaborada por Alejandro Ruiz

2.2. FUNCIÓN DE LOS COMPONENTES EN EL NUEVO CONTROL

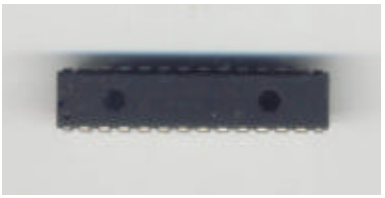
A continuación se describe cual es la función de algunos componentes electrónicos dentro del nuevo control como punto clave del trabajo desarrollado en este proyecto para el diseño de un nuevo control para la máquina, debido a que el anterior tenia un volumen considerable, eran un número importante de tarjetas conectadas entre si, y que el trabajo importante de control lo realizaba el PC por medio de Software, por lo que se quería con este proyecto darle cierta autonomía. En ningún momento se le realizaron cambios mecánicos a la máquina, eso se deja como recomendación en el capitulo correspondiente a “Futuros trabajos.”

2.2.1 Microcontrolador En un inicio se pensó en la posibilidad de emplear dos microcontroladores a modo maestro esclavo. Para ello se emplearían los Pic16f873 para el movimiento de los motores y el Pic16c765 para la comunicación con el computador, este a su vez sería el encargado de interpretar los datos e indicarle al microcontrolador de movimiento que rutina debería ejecutar.

Este diseño original se desechó al encontrarlo poco eficiente ya que la sincronización de los microcontroladores implica un problema de comunicación entre ellos al encontrarse oscilando a diferentes frecuencias. La solución a este problema implicaba emplear un intermediario con un protocolo de comunicación específico, lo que hacía este desarrollo más complejo.

- Microcontrolador 18F2550: Este microcontrolador tiene incorporadas las instrucciones para el manejo del puerto USB, adicionalmente, por el número de pines y puertos que posee es un microcontrolador óptimo para el manejo del control y a su vez implementar la comunicación vía puerto USB.

Figura 38. PIC18F2550



Fuente Vanessa Rodríguez L

Figura 39. Pines del PIC 18f2550

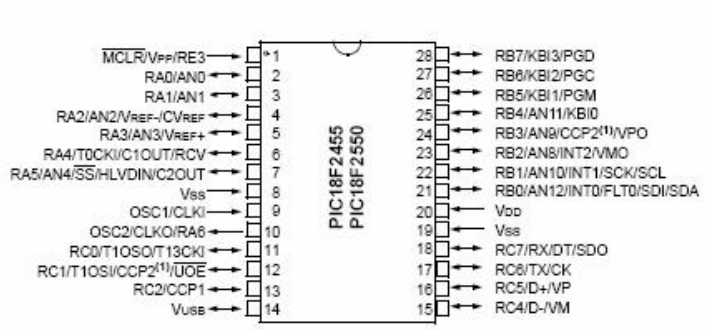


Imagen Microchip

Entre sus funciones se encuentran el movimiento de los tres motores. Las órdenes de movimiento serán enviadas al PIC. Este microcontrolador se programó con funciones básicas de corte las cuales iniciará según la orden que este reciba, también al tener pines de comunicación con el puerto USB (versión 1.1) permite la comunicación con el PC recibiendo los datos enviados por este.

2.2.2 Transistores Una vez la señal sale del microcontrolador ésta no tiene la potencia necesaria para hacer mover los motores, por esto se usan transistores tipo Darlington que pueden gobernar la corriente necesaria para mover los motores.

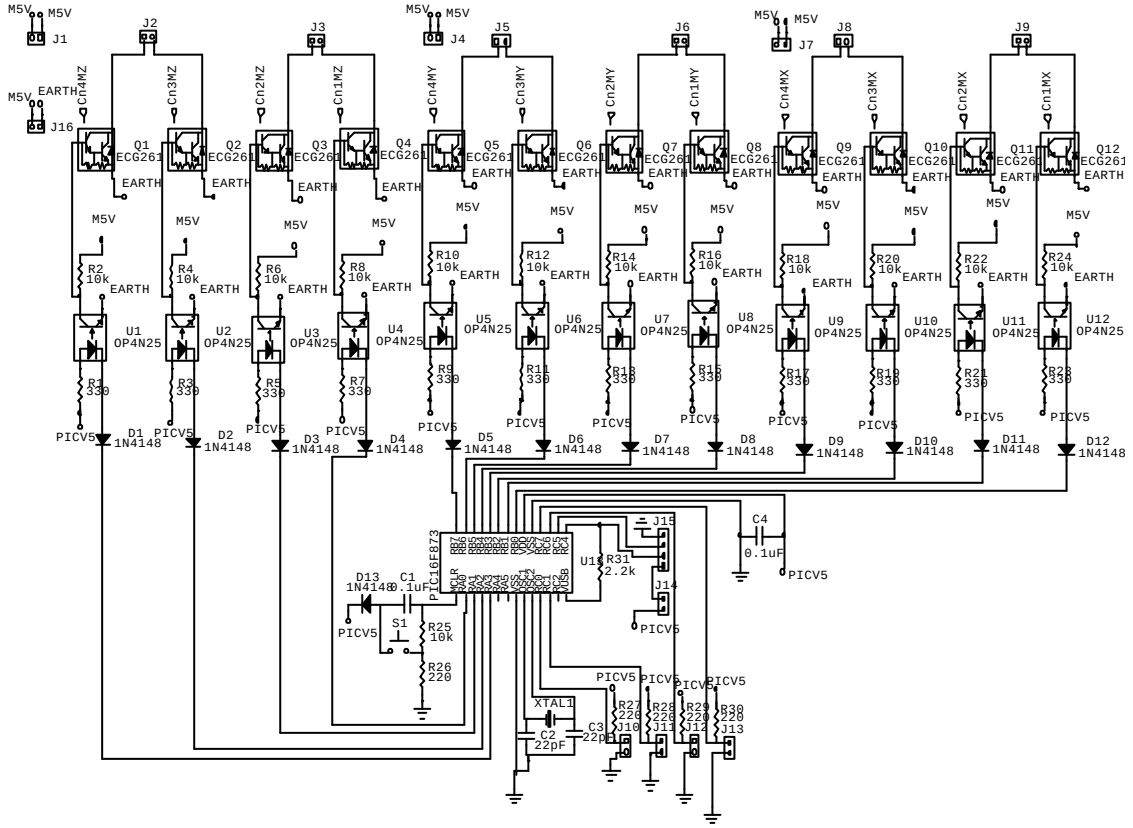
2.2.3 Resistencias En el control estas se encargan de limitar el paso de corriente a través del circuito, evitando que los componentes se quemen por el paso de esta, especialmente el microcontrolador. Entre más alto sea el valor nominal de esta, mayor será la resistencia al paso de corriente.

2.2.4 Condensadores Su función principal es eliminar el ruido dentro del circuito, permitiendo un buen funcionamiento del control.

2.2.5 Diodos Estos están ubicados de tal manera dentro del circuito que evitan que la corriente por la inductancia de los motores queme los circuitos de salida pines del microcontrolador que envían la señal a los motores.

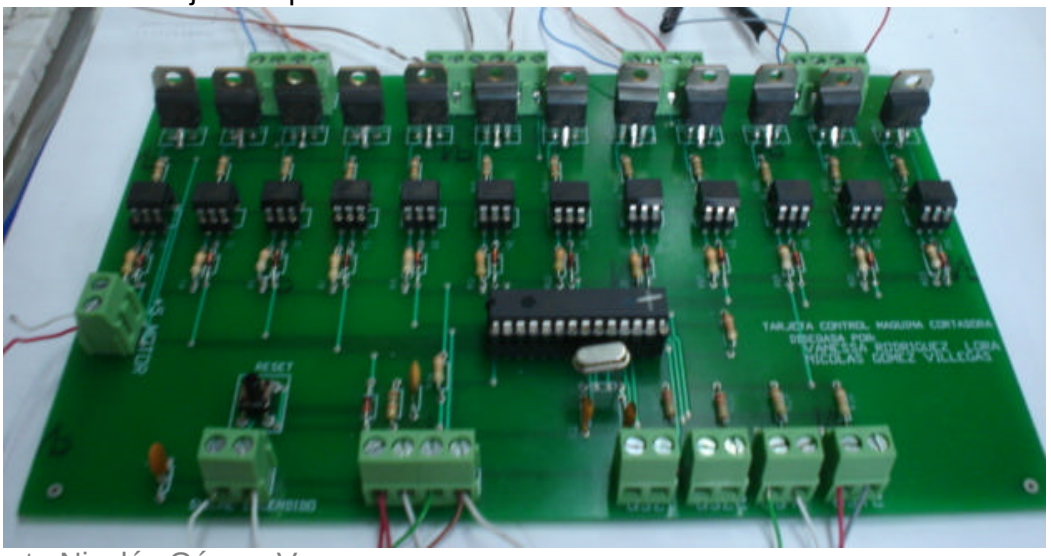
A continuación se muestran las figuras correspondientes al esquema de la tarjeta y la imagen de la tarjeta ya terminada.

Figura 40. Esquema Nuevo Control



Fuente Edwin Giraldo. Laboratorio de Control digital

Figura 41. Montaje Completo del Nuevo Control



Fuente Nicolás Gómez V

En la sección de anexos se encuentran los apartados correspondientes a la distribución de pines del PIC18f2550, los manuales de usuario y sistema.

3. DISEÑO DE SOFTWARE

3.1 RUTINAS

En éste capitulo se describen las rutinas diseñadas e implementadas para el probar el diseño del control.

La rutinas se han separado en dos grupos. Una es la parte de control que está implementada en Visual Basic para el PC, con ayuda de controles HID para el uso del puerto USB y en PICC para el microcontrolador. El segundo grupo son las de movimiento, allí están comprendidas las rutinas encargadas de cortar las figuras básicas diseñadas, así como también las tablas de control de los motores.

3.1.1 Rutinas de control Estas se dividen a su vez en dos partes las rutinas de control para el microcontrolador y las rutinas de control para el computador.

Para el computador se implemento un software de prueba sin diseño específico solo para comprobar que la comunicación entre el computador y el microcontrolador estaba funcionando correctamente. También se implementó un protocolo sencillo que está montado sobre el protocolo USB ya que los paquetes de información que recibe el dispositivo son máximo de 8 bytes el protocolo debía encargarse de dividir los datos en paquetes de este tamaño.

El trabajo con USB es una tarea interesante ya que requiere del estudio de los estándares de USB que ahora está en la versión 2.0 y siguen avanzando con nuevas definiciones constantemente. En este proyecto se trabajó con el estándar 1.1 ya que está consolidado y porque el microcontrolador trabaja con este estándar además, de que existen algunos microcontroladores que aún no usan la versión de USB 2.0.

Lo primero que hay que tener en cuenta al trabajar con USB es el archivo descriptor en el cual hay que definir el dispositivo que se está diseñando, en este descriptor dice todo lo que el dispositivo puede hacer y como lo debe reconocer el PC, si hay algún problema en el descriptor, el PC no lo reconocerá y no funcionara de ningún modo, así que hay que ser muy cuidadoso con este archivo.

La mayoría de los compiladores para microcontroladores tienen funciones preestablecidas para el manejo de USB, lo que facilita las cosas para el programador. Pero no hay ningún apoyo para el diseño del descriptor, lo mejor es tomar un descriptor ya hecho, estudiarlo y modificarlo lentamente. A la hora de escoger que tipo de dispositivo se va a implementar hay que tener presente que funcionalidad tendrá el mismo.

El sistema operativo Windows reconoce ciertos dispositivos automáticamente, lo cual facilita el desarrollo y evita al programador tener que hacer un controlador propio para el dispositivo, entonces es una buena opción manejar dispositivos HID cuando se está comenzando en el área de USB, claro que un programador experto puede hacer dispositivos más complicados, pero no es recomendable para iniciar. La parte del PC que se uso para este proyecto fue Visual Basic con un control Activex que permite reconocer los dispositivos que están conectados y escoger cual es el que va a funcionar con la aplicación.

El control ya tiene definidas las funciones de envío y recepción de datos, lo que facilita las pruebas y limita los factores de error. Una vez el dispositivo ya es reconocido por el PC solo hay que usar las funciones de envío y recepción acorde con las necesidades del programa.

Hay que tener especial cuidado con el voltaje del microcontrolador, ya que es muy susceptible a cambios y se quema fácilmente. El puerto USB en el microcontrolador también es sensible y puede quemarse independientemente del PIC, lo que hace que el PIC funcione pero los pines asignados para el trabajo con USB no.

El lenguaje Visual Basic es un lenguaje que funciona por medio de eventos, lo que permite desencadenar acciones mediante modificaciones de campos o cuando se oprime un botón por ejemplo. Lo que hace este programa es recibir por medio del teclado la información de los diagramas que se van a realizar en la máquina y oprimiendo el botón enviar se inicia la acción de la máquina, es justo en este momento donde se inicia el protocolo de comunicación entre el computador y la máquina.

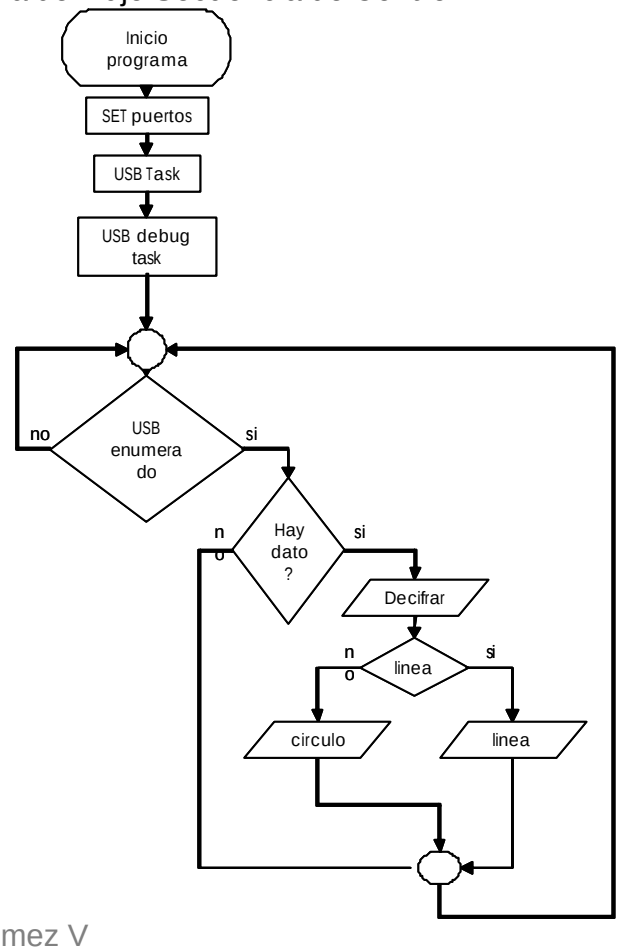
Los datos que se envían a la máquina son los correspondientes a la longitud que debe estar dada en milímetros y a la pendiente de las diagonales que debe ser proporcionada en radianes.

Primero se deben dividir los datos para poder ser enviados por USB, ya que este maneja paquetes de datos de 8 bytes. La división de los datos para la longitud se hace partiéndolos en unidades, decenas y centenas, para la pendiente se hace para las unidades y las centésimas.

Este software de prueba se encarga de manejar la conexión a este dispositivo y únicamente reconoce a este debido al descriptor.

También se implementó una rutina de control para el microcontrolador que consiste en la recepción de los paquetes de datos. El siguiente diagrama muestra el diseño de esta rutina.

Figura 42: Diagrama de Flujo Secuencia de Control



Fuente Nicolás Gómez V

3.1.2 Rutinas de movimiento Antes de describir los programas que se encargarán de mover la máquina es necesario explicar la lógica utilizada para describir las diferentes figuras, en este caso específico, líneas verticales, líneas horizontales, diagonales y círculos. Es necesario tener en cuenta el factor de conversión para los ejes al momento de programar las rutinas. Esta precisión es diferente para los ejes X y Y. Así que el movimiento para cado uno de estos ejes debe ser diseñado e implementado teniendo en cuenta estos factores.

Según las especificaciones entregadas inicialmente de la máquina se tenía que la precisión para el eje X era de 0.02708 mm por cada paso del motor y para Y era de 0.02747 mm por paso. Después de haber realizado varias pruebas de movimiento en las cuales se trazaba la misma figura pero con dimensiones diferentes, se encontró que estas medidas no correspondían a las medidas reales, por lo que después de realizar varios cálculos se encontró que las medidas para los pasos de los motores eran de 0.0285 para el motor en X y de 0.046 para el motor en Y

A continuación se hace una breve descripción de las rutinas implementadas para que sirvan de apoyo al lector para realizar trabajos futuros.

- **INT DERECHAR (FLOAT mto)** esta rutina es la encargada de hacer que la herramienta se mueva hacia la derecha, haciendo uso de una tabla con los pulsos de los pines en el puerto al cual está conectado el motor.

- **INT IZQUIERDAR (FLOAT mto)** esta rutina es la encargada de hacer que la herramienta se mueva hacia la izquierda, haciendo uso de una tabla con los pulsos de los pines en el puerto al cual está conectado el motor.

- **INT ARRIBAR (FLOAT mto)** esta rutina es la encargada de hacer que el material a ser cortado se mueva hacia arriba, haciendo uso de una tabla con los pulsos de los pines en el puerto al cual está conectado el motor.

- **INT ABAJOR (FLOAT mto)** esta rutina es la encargada de hacer que el material a ser cortado se mueva hacia abajo, haciendo uso de una tabla con los pulsos de los pines en el puerto al cual está conectado el motor.

- **INT ARRIBATOOL ()** esta rutina es la encargada de elevar la herramienta para evitar el desperdicio de material.

- **INT ABAJOTOOL ()** esta rutina es la encargada de bajar la herramienta para hacer los cortes en el material.

- **FLOAT SGN (FLOAT X)** esta es una función de apoyo para el trabajo de la diagonal, la cual devuelve un valor de 1 ó – 1 dependiendo del signo del valor que recibe.

- **VERIFICA (INT DIR)** esta es una función de apoyo para el movimiento en el eje X para el final de carrera en el mismo.

- Líneas Verticales, Horizontales y Diagonales: En lugar de implementar un programa que dibujara líneas verticales, horizontales y diagonales en forma separada, se decidió tener un programa único que realizara cortes en cualquier sentido, independiente del ángulo de la línea.

Los programas que se tenían originalmente dibujaban líneas rectas en X o en Y dejando uno de los motores estáticos mientras que el otro se movía realizando el corte. El algoritmo del programa implementado originalmente para trazar líneas horizontales era el siguiente:

```
void LineH(int x, int y)
{
    int i;
    for(i=x; i<=x; i++)
        PutPixel(i, y);
}
```

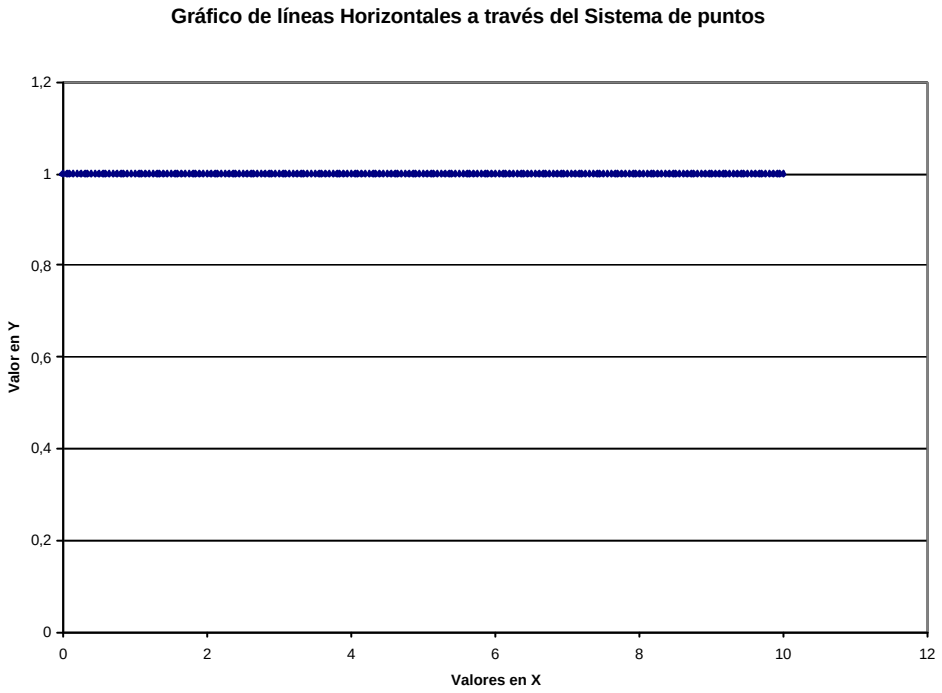
Es preciso aclarar que la función PutPixel es una función gráfica[15] encargada de poner los puntos que irán haciendo la línea. Es necesario recordar que antes de dibujar la línea se tiene que hacer la relación de distancia con pasos del motor que son los que realmente van a formar la línea, es decir, lo que en este sistema gráfico se está llamando puntos para la máquina cortadora son pasos de motor.

[15] Funciones obtenidas en www.geocities/programacion_grafica/primitivas/primitivas.htm#lineas

$$\text{Pasos en X} = \frac{Longitud\,(mm)}{0.02708\ mm}$$

Gráficamente esto es:

Figura 43. Gráfico de líneas horizontales a través del sistema de puntos



Fuente Vanessa Rodríguez L

Análogamente ocurre con las líneas verticales, pero hay que tener muy presente que el factor de conversión es diferente. Es decir, la fórmula que se utilizará en este caso es:

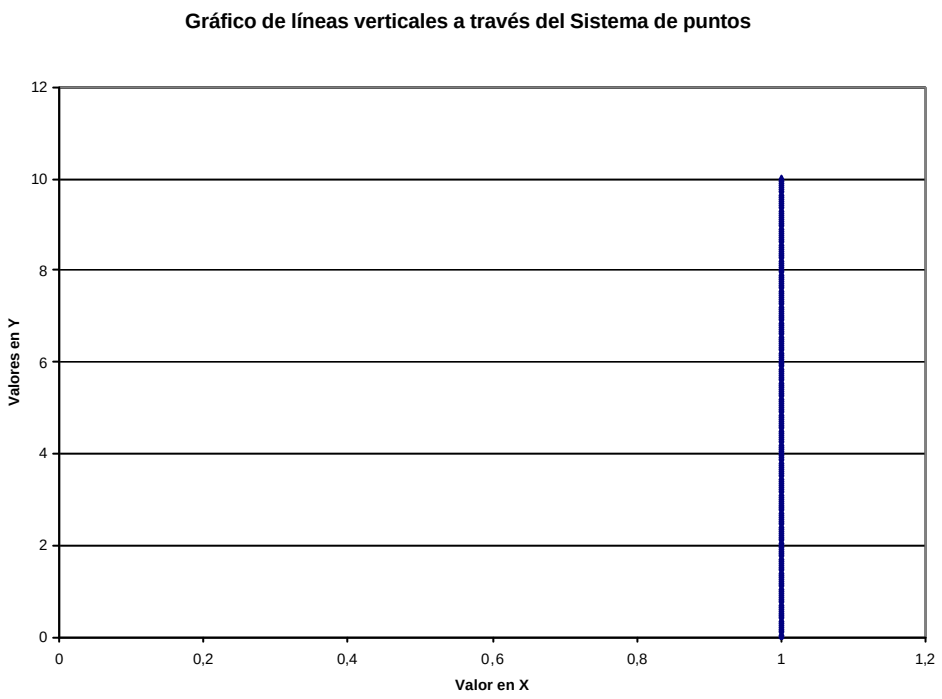
$$\text{Pasos en Y} = \frac{Longitud\,(mm)}{0.02747\ mm}$$

Este es el algoritmo que se implementó originalmente para líneas verticales:

```
void LineV(int x, int y)
{
    int i;
    for(i=y; i<=y; i++)
        PutPixel(x, i);
}
```

De forma gráfica se tiene:

Figura 44. Gráfico de líneas verticales a través del sistema de puntos



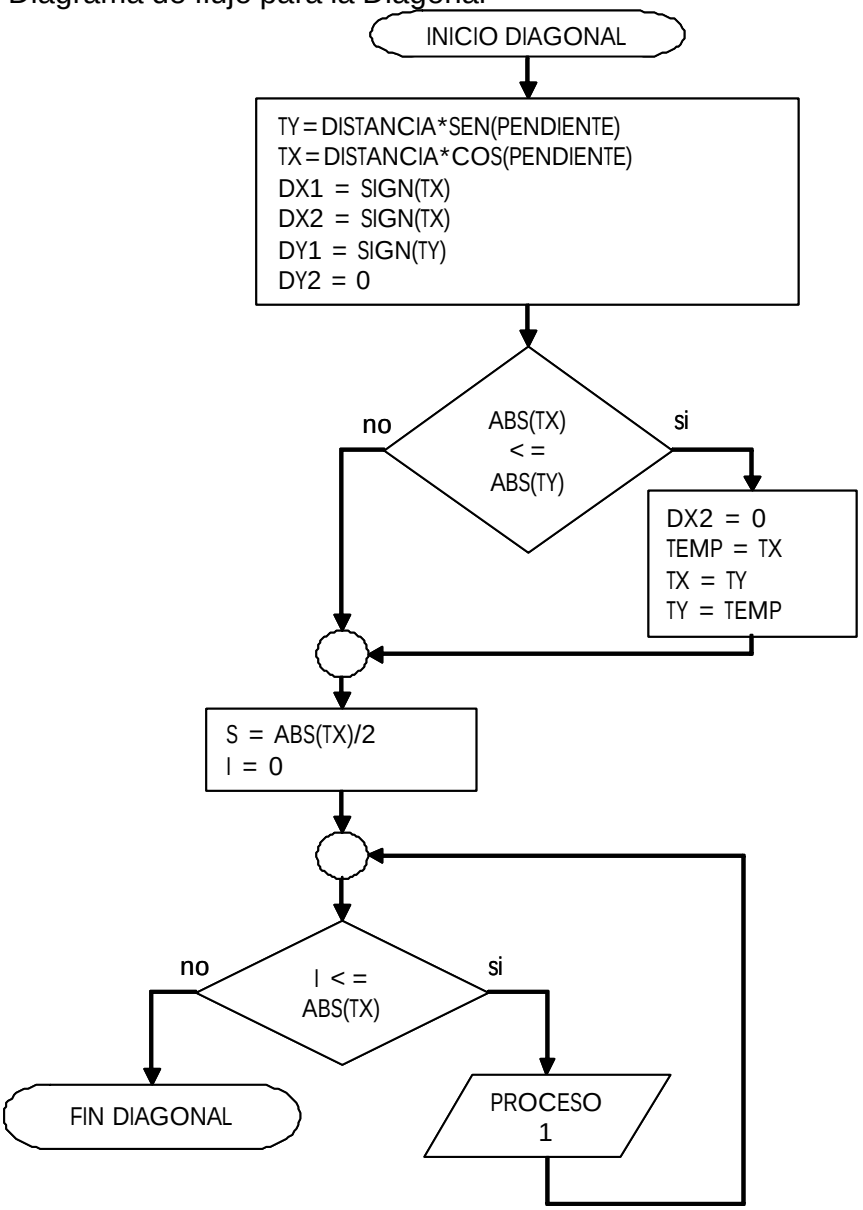
Fuente Vanessa Rodríguez L

Para optimizar el programa se implemento el algoritmo de la diagonal, ya que este permite dibujar líneas con cualquier pendiente, es decir, este está en capacidad de trazar líneas en cualquier dirección, en cualquier ángulo, incluyendo a las líneas horizontales y verticales.

Implantar este algoritmo es un poco tedioso ya que este no representa un ciclo sino que por el contrario es un algoritmo de decisión que debe determinar si la pendiente de la línea es positiva, o negativa, si como también si el valor de esta es mayor o menor de uno.

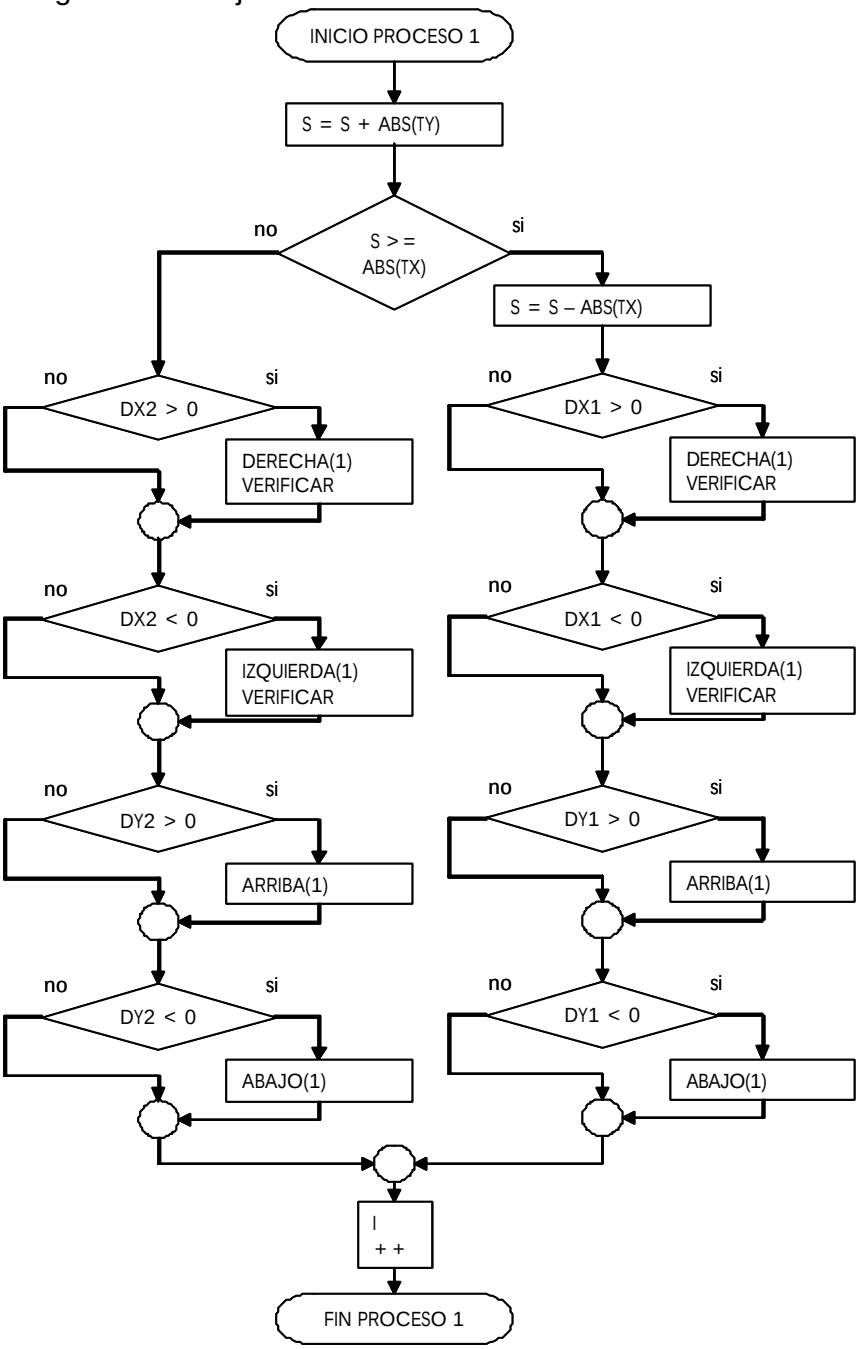
Este algoritmo trabaja empleando el algoritmo de Bresenham, este algoritmo se basa en el hecho que una vez puesto un punto en (X, Y) el siguiente esta a (X+1, Y), (X+1, Y-1) ó a (X+1, Y+1). El algoritmo que traza diagonales y líneas es el siguiente:

Figura 45. Diagrama de flujo para la Diagonal



Fuente Nicolás Gómez V

Figura 46. Diagrama de Flujo Proceso de Decisión 1.



Fuente Nicolás Gómez V

El algoritmo en el que se basó para el trazo de diagonales y líneas es el siguiente:

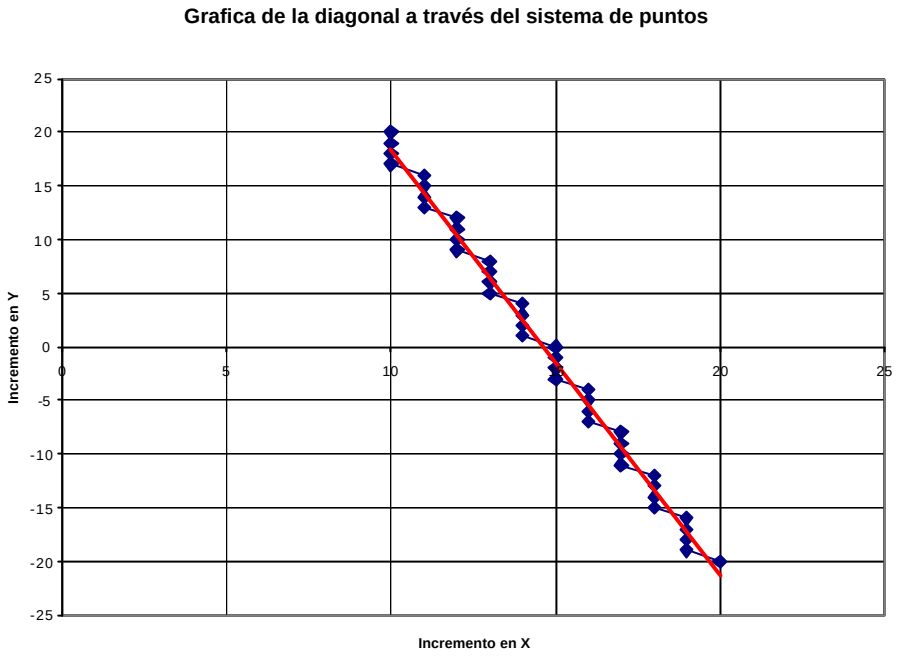
```
int sgn(int x)
{
    if(x>0)  return(1);
    if(x<0)  return(-1);
    return (0);
}
void Line(int x1, int y1, int x2, int y2)
{

    int i, s, u, v, dx1, dx2, dy1, dy2, m, n;
    u=x2-x1;
    v=y2-y1;
    dx1=sgn(u);
    dy1=sgn(v);
    dx2=sgn(u);
    dy2=0;
    m=abs(u);
    n=abs(v);
    if(!(m>n))
    {

        dx2=0;
        dy2=sgn(v);
        m=abs(v);
        n=abs(u);
    }
   s=m/2;
    for(i=0; i<=m; i++)
    {
        PutPixel(x1, y1);
        s=s+n;
        if (!(s<m))
        {
          s=s-m;
          x1=x1+dx1;
          y1=y1+dy1;
        }
        else
        {
            x1=x1+dx2;
            y1=y1+dy2;
        }
    }
}
```

El siguiente gráfico muestra los trazos que hace el algoritmo para obtener una diagonal. Nótese que este algoritmo lo que traza es una serie de escalones. La línea en rojo es una línea de tendencia la cual muestra la trayectoria de la línea original:

Figura 47. Gráfico del círculo a través del sistema de puntos



Fuente Vanessa Rodríguez L

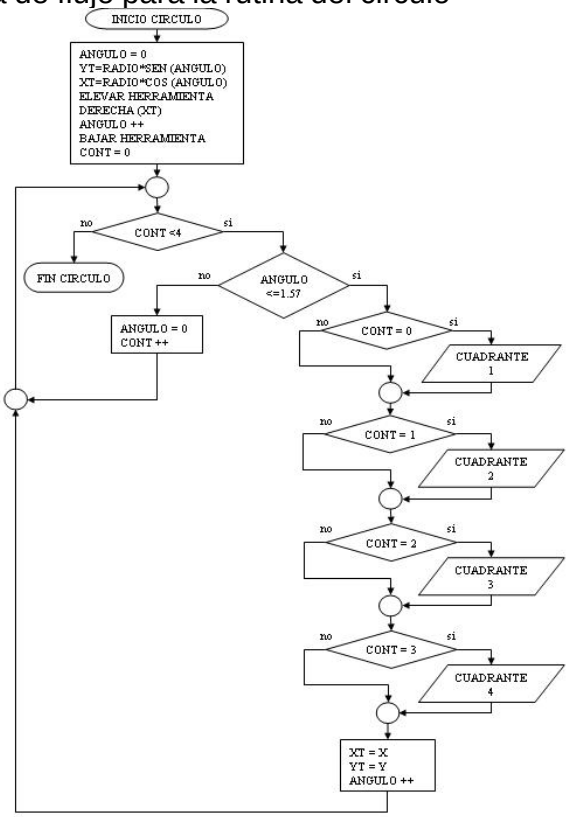
Al principio se asumió que las diagonales se podían hacer de modo similar a los radios del círculo, por lo cual se diseñó un algoritmo modificando al del círculo para crear la rutina de las diagonales, variando el radio en lugar del ángulo, pero a medida que se configuraba la precisión de los ejes se tomó la decisión de realizar esta rutina empleando un algoritmo que trazara las diagonales haciendo uso de diferencias sucesivas.

Circulo: Está es la función es la que hace los movimientos para el dibujo del circulo. Recibe como parámetro el valor del radio y mediante las funciones de seno y coseno encuentra los puntos en la periferia del círculo para luego valerse de las rutinas de movimiento haciendo que se muevan los motores recorriendo el camino de la circunferencia. El primer punto se halla con el radio, el seno y coseno de 0, se hace entonces el movimiento de la herramienta hasta encontrar este punto. El siguiente punto se halla variando el ángulo (en radianes) y la herramienta se mueve haciendo uso de las rutinas de movimiento hasta que la diferencia entre **X0 - X1** y **Y0 - Y1**, sea 0. La dirección de movimiento de la máquina depende del valor del ángulo.

```
void Circle(int cx, int cy, int radio)
{
    float angulo=0;
    int x, y;
    do
    {
        x = cx + radio * cos(angulo);
        y = cy + radio * sin(angulo);
        if((x>=0) && (y>=0))
            PutPixel(x, y);
        angulo+=0.005;
     } while(angulo<6.28);
}
```

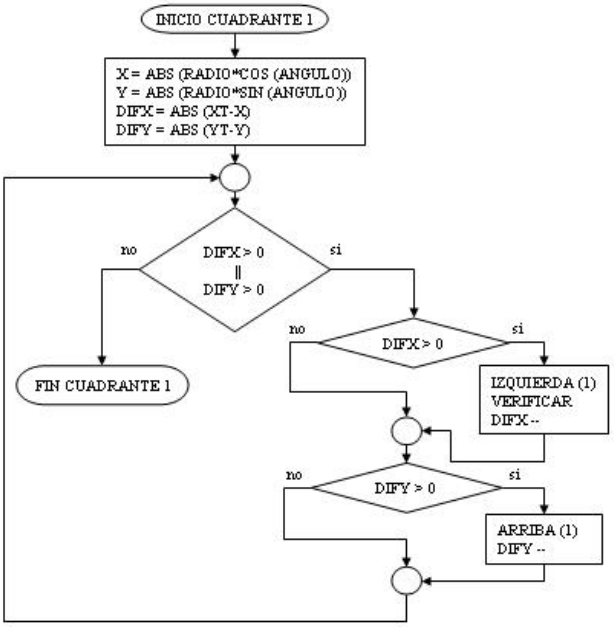
Los diagramas presentados a continuación muestran la forma en la cual este algoritmo diseñado para un entorno gráfico fue modificado de tal forma que pudiera ser representado en un sistema coordenado XY.

Figura 48. Diagrama de flujo para la rutina del circulo



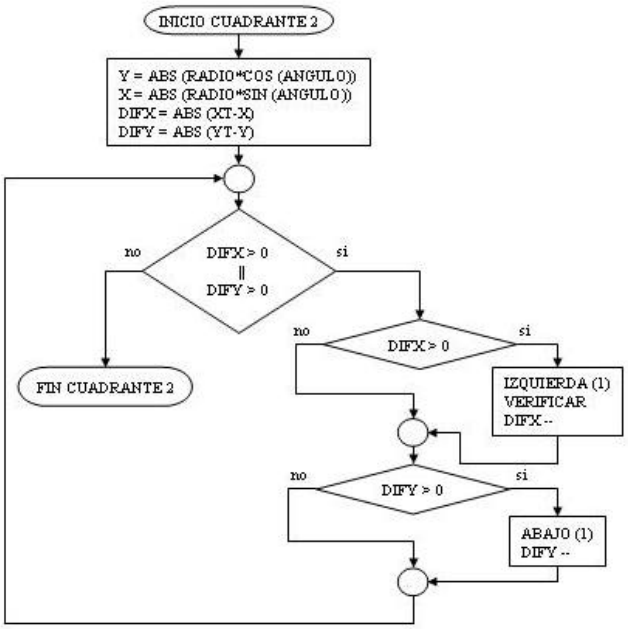
Fuente Nicolás Gómez V

Figura 49. Diagrama de flujo para la rutina del primer cuadrante



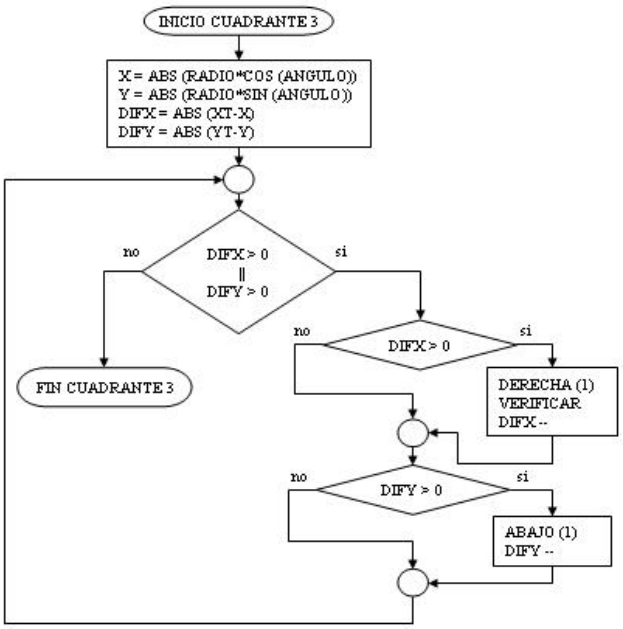
Fuente Nicolás Gómez V

Figura 50. Diagrama de flujo para la rutina del segundo cuadrante.



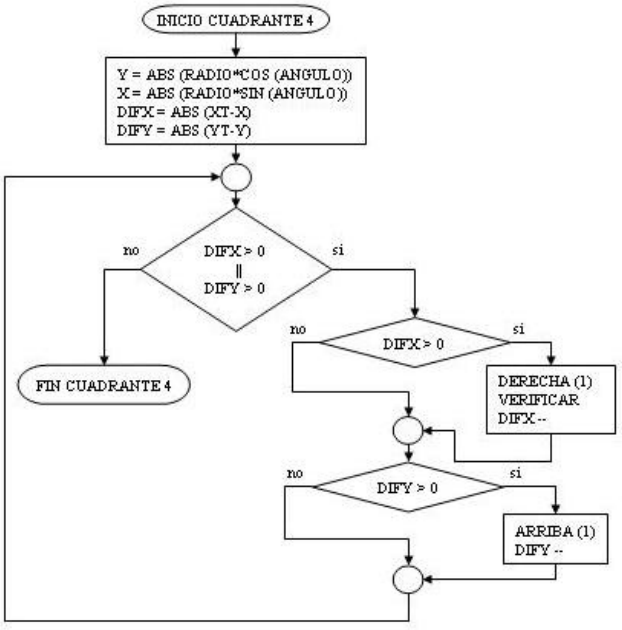
Fuente Nicolás Gómez V

Figura 51. Diagrama de flujo para la rutina del tercer cuadrante.



Fuente Nicolás Gómez V

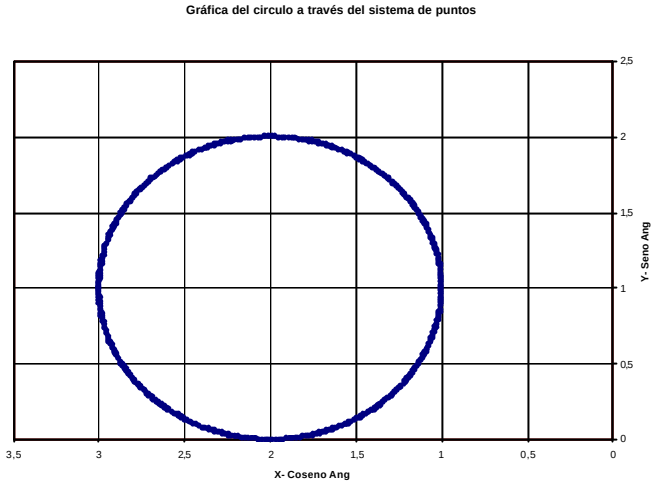
Figura 52. Diagrama de flujo para la rutina del cuarto cuadrante



Fuente Nicolás Gómez V

El siguiente gráfico muestra una circunferencia de radio 1 y coordenadas de centro en (2,1).

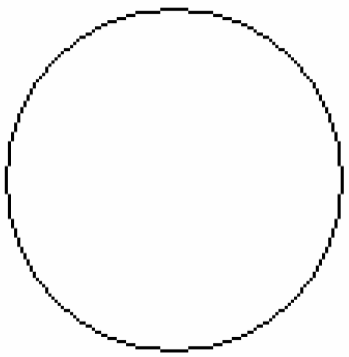
Figura 53. Gráfico del círculo a través del sistema de puntos



Fuente Vanessa Rodríguez L

Cabe aclarar que estos algoritmos no crean círculos completamente redondos, lo que estos hacen es crear una sucesión de líneas muy pequeñas de diferentes longitudes, pero a una misma distancia. El siguiente dibujo fue creado en un ambiente gráfico, al ampliar la imagen se puede evidenciar más fácilmente la sucesión de líneas que se mencionaba.

Figura 54. Circulo dibujado a través de líneas rectas



Fuente Vanessa Rodríguez Lora

Estos tipos de trazos son los trazos básicos, ya que cualquier tipo de figura puede descomponerse en una sucesión de las anteriores, por ejemplo, un cuadrado es una figura compuesta de 2 líneas horizontales y dos verticales, un triángulo está formado por combinaciones de rectas y diagonales, los arcos pueden trazarse de modo similar a los círculos teniendo en cuenta que estos tienen un ángulo que va de 0 a un número menor de 6.28 y así todas las demás figuras.

Los códigos completos de los programas completos se encuentran en los anexos, para que el lector pueda tomarlos de base para futuros trabajos.

4. CONCLUSIONES

El diseño del control automático permitió lograr un manejo más simple y cómodo que permite optimizar su uso debido a lo reducido de su tamaño y el uso de una tarjeta única.

El trabajo de comunicación PC máquina usando el puerto USB, incrementa la funcionalidad de la máquina, además permite que esta pueda ser compatible con las nuevas arquitecturas diseñadas para equipos de computo.

Aunque originalmente se había planteado la posibilidad de tener un control con dos microcontroladores a modo "maestro-esclavo", después de hacer múltiples pruebas con este mecanismo se descartó esta idea porque era necesario contar con un intermediario para que los microcontroladores pudieran comunicarse a través de este. Esto se logró hacer usando un microcontrolador de gama media creado por Microchip, haciendo el control mucho más eficiente.

Después de realizar varios prototipos para el diseño de la herramienta se decidió utilizar el microcontrolador PIC18F2550 ya que permite, gracias a su distribución de pines y a su memoria tener almacenados en un solo dispositivo los programas de control y movimiento empleados.

El sistema de cortes desarrollado nuevo está compuesto de líneas, diagonales y círculos para tener un mecanismo que controlara el movimiento de los motores independiente del software usado en el PC. Al implementar este sistema con estas figuras básicas se quería tener un mecanismo que le dijera a los motores como moverse independiente del software empleado en PC, es decir, el microcontrolador sabe que movimientos debe realizar para una figura en particular, el único dato que necesita ser proporcionado por el PC es el parámetro (Ej. Longitud, radio, dirección). Este servicio cuenta con tres figuras básicas (líneas, diagonales y círculos), ya que estas puestas en forma conjunta pueden formar otras figuras, tales como cuadrados, rectángulos, triángulos, arcos.

Unos de los principales problemas que se encontraron durante el desarrollo de este proyecto fue el poco conocimiento que existe en el ámbito académico de las herramientas tecnológicas disponibles y de la funcionalidad de estas mismas. Es

por esto que gran parte del tiempo programada para el desarrollo de este trabajo de grado debió ser invertido en el aprendizaje de estas herramientas.

5. FUTUROS TRABAJOS

El trabajo que se hizo con este control es el inicio de numerosas oportunidades para el crecimiento del mismo, se puede mejorar este control de muchas maneras diferentes dependiendo al área que quiera enfocarse. En el área de programación, se puede hacer una aplicación más robusta en el PC que permita una variedad de comandos y figuras prediseñadas (perfiles) para cortar con la máquina, también puede diseñarse un compilador que tome un archivo CAD en DXF para que interprete planos y pueda cortarse cualquier figura.

En el área de electrónica pueden hacerse mejoras en la tarjeta, hacerla más amigable al usuario adicionando luces de estado que indiquen si la máquina está encendida o apagada, si está trabajando o se encuentra esperando un dato y algún otro factor que no haya sido tenido en cuenta en este primer desarrollo.

En el área de mecánica se pueden hacer varias modificaciones a la máquina para que haga un trabajo mejor, más fino.. Con la ayuda del control se puede observar que nuevas modificaciones sería interesante hacerle a la máquina ya sean para seguridad o desempeño. Estas pueden ser cambio de los motores empleados, sensores de material, *encoders* que indiquen la Posición, cambiar el soporte de la herramienta para realizar cortes y figuras en 3D.

6. RESULTADOS OBTENIDOS

El trabajo realizado es apenas el inicio del desarrollo para un nuevo camino en la comunicación entre dispositivos y PC, este a su vez abre ese camino y deja las primeras enseñanzas en como trabajar con el protocolo USB ya que se hizo una investigación detallada y profunda del tema debido a que pocas personas en la Universidad lo están empleando, el desarrollo de este proyecto dio como resultado un avance en el conocimiento del USB que ahora está más cerca de los estudiantes y permitirá el desarrollo de nuevos proyectos.

También se hizo una investigación sobre el uso de motores paso a paso para poder crear rutinas basadas en funciones matemáticas que permitieran movilizar la herramienta en una mesa de corte.

Se logró poner en funcionamiento los tres ejes de movimiento teniendo bajos consumos de corriente eléctrica, factor que incidía notablemente en el diseño del control. Esto permitió tener un control más eficiente y económico. Es necesario indicar que inicialmente el control manejaba una fuente de voltaje para cada uno de los motores, con este nuevo control sólo se tiene una fuente para este propósito.

El poder de los nuevos microcontroladores y los compiladores que permiten programarlos amplían las posibilidades para el trabajo didáctico con estos. Los nuevos compiladores permiten al programador enfocarse más en el fondo del programa que en la forma del mismo, permitiendo así que implementen desarrollos y aplicaciones cada vez más complejos.

Fueron trabajadas funciones matemáticas diseñadas para entornos gráficos modificándolas para poner en funcionamiento un sistema XY, este trabajo implicó manejar conceptos matemáticos, trigonométricos y geométricos para poder interpretar las funciones y acoplarlas al sistema empleado por la máquina en las rutinas de movimiento para el corte de las líneas y circunferencias.

BIBLIOGRAFÍA

- ARANGO PARDO, Luis Alberto y MEJIA ARBELAEZ, Juan David. Sistema Interpolador de Trayectorias Para una Mesa de Desplazamiento Horizontal Sistema SINT. Medellín. 1984, 46p. Trabajo de Grado (Ingeniería de Sistemas). Universidad EAFIT. Departamento de Ingeniería de Sistemas.
- AutoCAD 2002. Autodesk inc. 2001

- CARDENAS, Juan Diego y RAMIREZ, Mario. Control Autónomo para la Mesa X-Y Numéricamente Controlada. Medellín. 1986, 94 p. Trabajo de Grado (Ingeniería de Sistemas). Universidad EAFIT. Departamento de Ingeniería de Sistemas.

- Diccionario RAE. Real Academia Española. Madrid, 2005

- Enciclopedia Encarta 2005. Microsoft.

- INSTITUTO COLOMBIANO DE NORMAS TECNICAS Y CERTIFICACION. Tesis y otros trabajos de grado: Normas técnicas colombianas sobre documentación. Norma técnica colombiana NTC 1486 quinta actualización. Santa fé de Bogotá 2002.

- KARAISKOS, Pete; FULTON, Nancy. "Autocad For Mechanical Engineers And Designers" Ed John Wiley & Sons, Inc 1995. USA.

- Microchip Technology. "Embedded Control Handbook Volume 1" 1997, USA

- Microchip Technology. Datasheet PIC16C745/765. 2000. USA.

- Microchip Technology. DataSheet PIC18F2455/2550/4455/4550. 2004. USA

- Mindshare,Inc. ANDERSON Don. "Universal Serial Bus System Architecture" Ed Addison- Wesley Longman, Inc 1998 . USA

- PALLÁS ARENY, Ramón. "Diafonía En Circuitos Impresos. Criterios De Diseño". Revista Mundo Electrónico Diciembre 1993. Boixareu Editores. Barcelona, España.

- RENDON, Elizabeth. Diseño y Construcción de una cortadora CNC de patrones en balsa.. Medellín 2001. 279p. Trabajo de grados Ingeniería Mecánica. Universidad EAFIT. Departamento de Ingeniería Mecánica.

• RENDON, Elizabeth; RUIZ, Oscar y MARTÍNEZ Adriana. Low Level Direct Interpolation for Parametric Curves. En: Revista Universidad EAFIT, Medellín. No. 110 (1998); p. 76-84.

• RENDÓN, Marc. Critical Desing Review. Fist PIC. Noviembre de 2003.

• SÁNCHEZ MILLARES, Álvaro. Sensores y actuadotes. Instituto de investigación técnica. 2004

• TAJADURA, José Antonio; Javier López Fernández. "Autocad Avanzado V.14 Volumen 1". Mcgraw Hill 1998. Madrid, España.

Páginas Web

• Escuela Politécnica Superior de Alcoy, [in line], http://server-die.alc.upv.es, Universidad Politécnica de Valencia, creada el 20 de noviembre de 1996, visitada el 11 de octubre de 2004.

• Todo robot, [in line], http://www.todorobot.com.ar., publicada por Yahoo, creada en agosto de 2000, visitada el 2 de noviembre de 2004

• Beyond Logia, [in line], www.beyondlogic.org, Publicada Australian Electronics Web ring, creada en agosto de 1995, visitada el 28 de enero de 2005.

• PIC USB, [in line], http://www.alecmcnamara.freeserve.co.uk/piccalc/, Publicada por Alec McNamara, creada en septiembre de 2002, visitada el 9 de febrero de 2005

• Primitivas, [in line], http://www.geocities.com/programacion_grafica/primitivas, publicada por Geocities, creada en octubre de 1997, visitada el 15 de abril de 2005

• Philips semiconductors, [in line], http://www.semiconductors.philips.com, publicada por Philips, creada en 2004, visitada el 17 de Julio de 2005.

• Tutorial de administración de archivos, [in line], http://www.itlp.edu.mx/publica/tutoriales/admonarch, publicado por el Instituto tecnológico de la Paz, creada en junio de 1999, visitada el 20 de julio de 2005

• http://members.fortunecity.es . Página visitada el 23 de Julio de 2005

- Maquinas de Control Numérico, [in line], http://www.monografias.com, publicada por Monografías.com, creada en 1997, visita el 30 de Julio de 2005.

- Portal de mantenimiento Industrial, [in line], http://www.solomantenimiento.com, publicada por TPM, creada en julio de 2002, visitada el 30 de Julio de 2005.

- Electrónica 2000, [in line], http://www.electronica2000.250x.com., Publicada por Hugo Mendez, creada en Julio de 2000, visitada el 2 de agosto de 2005.

- Ingeniería mecánica, [in line], http://www.fim.utp.ac.pa., publicada por la Universidad tecnológica de Panamá, creada en 1995, visitada el 3 de agosto de 2005.

- Foros de electrónica, [in line], http://www.forosdeelectronica.com, publicada por Andrés Fernando Cuenca, creada en enero de 2005, visitada el 24 de agosto de 2005

- Aldirsa S.A, [in line], www.aldirsa.com, publicada por Aldirsa S.A, creada en enero de 2003, visitada el 15 de septiembre de 2005

- Marbel Control & automatismos, [in line], .www.marbelonline.com, publicada por marbel, creada en noviembre de 2003, visitada en agosto de 2005.

- Rikaline España, [in line], www.electrokit.com, publicada por Ricaline, creada en diciembre de 2004. visitada el 24 de agosto de 2005.

- Universidad de Granada, [in line], www.ugr.es, publicada por la UGR, creada en 1998, visitada en agosto de 2005.

- Anatronic S.A, [in line], www.anatronic.com, publicada por Anatronic, creada en el 2004, visitada en Julio de 2005.

- Planeta electrónico, [in line], www.planetaelectrónico.com, publicada por Mario Hernández, creada en agosto de 2002, visitada en Julio de 2005.

- Huarpe portal educativo, [in line], http://huarpe.com, publicada por Raul Armando Funes, creada en mayo de 2001, visitada en agosto de 2005.

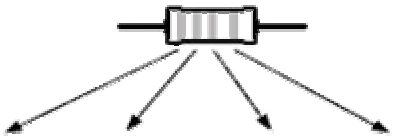
- Centro Nacional de información y comunicación educativa, [in line], www.cnice.mecd.es, publicado por el ministerio de educación y ciencia español, creada en 1996, visitada en abril de 2005.

- Lu1dma Web site, [in line], www.lu1dma.com.ar, publicada por Lu1dma radioaficion, creada en 1997, visitada en junio de 2005.

ANEXOS

Anexo A. Código de color para resistencias

Código de colores para resistencias



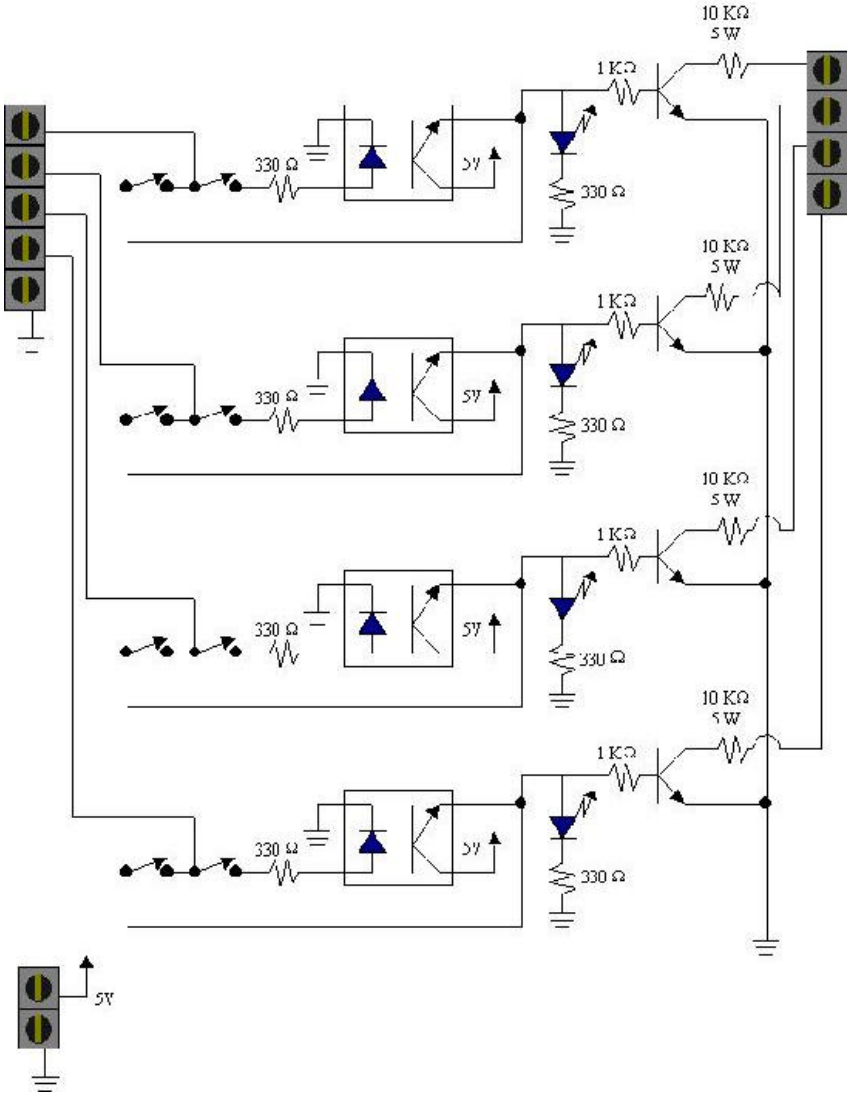
Colores	1ª Cifra	2ª Cifra	Multiplicador	Tolerancia
Negro		0	0	
Marrón	1	1	x 10	±1%
Rojo	2	2	x 10^2	±2%
Naranja	3	3	x 10^3	
Amarillo	4	4	x 10^4	
Verde	5	5	x 10^5	±0.5%
Azul	6	6	x 10^6	
Violeta	7	7	x 10^7	
Gris	8	8	x 10^8	
Blanco	9	9	x 10^9	
Oro			x 10^{-1}	±5%
Plata			x 10^{-2}	±10%
Sin color				±20%

Fuente www.ugr.es

Anexo B. Tarjeta de control de motores paso a paso

Este tipo de tarjeta se encarga de enviar a las bobinas de los motores paso a paso el voltaje necesario para su movimiento el cual es emitido por el microcontrolador. Por el diseño del control original de la máquina se hace necesario emplear una tarjeta para cada uno de los motores, es decir, para este se emplearon tres tarjetas, cada una de ellas con su respectiva fuente de voltaje.

Tarjeta de control de motores paso a paso

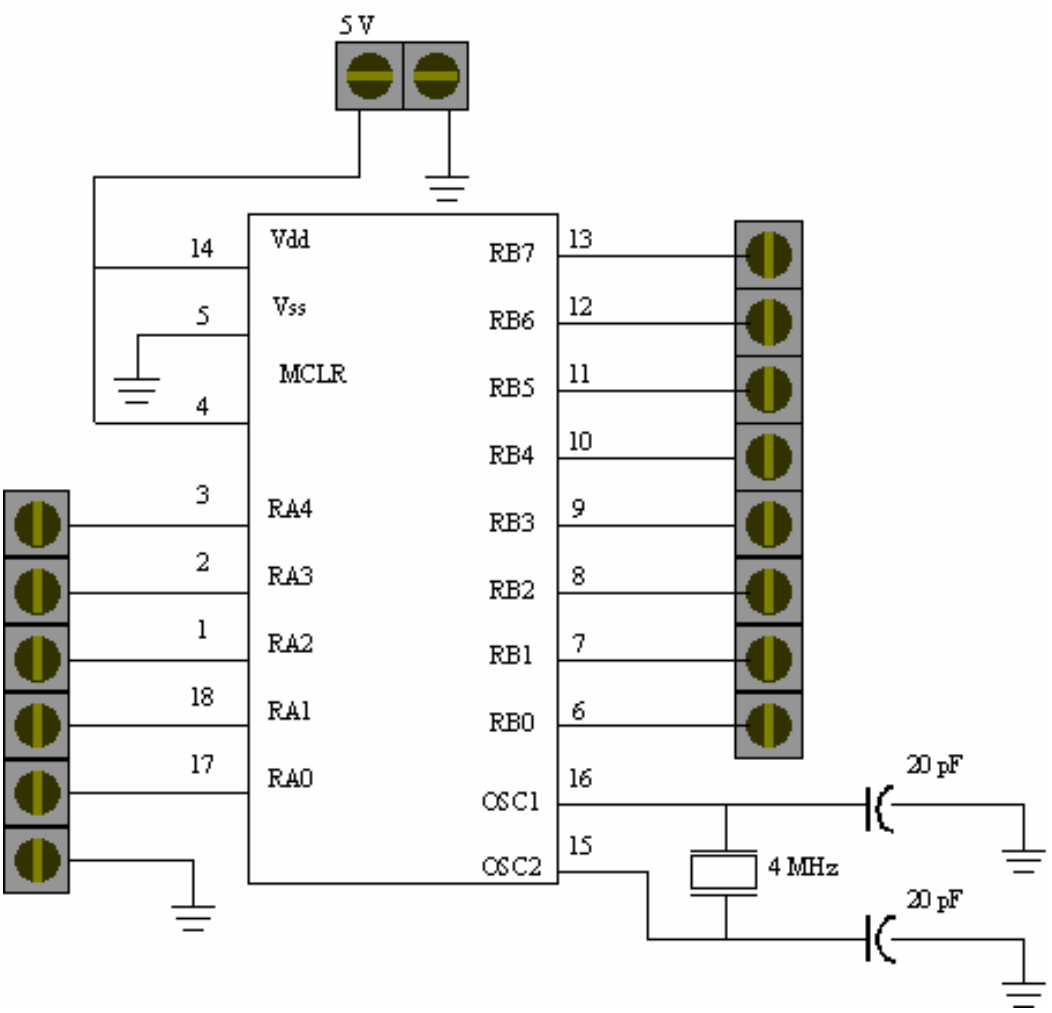


Elaborada por Edwin Giraldo. Laboratorio de Control Digital

Anexo C. Tarjeta para micro controlador PIC 16F84

Esta tarjeta se encarga de transformar las entradas generadas por el computador en una rutina de pulsos que son enviados a los motores a través de sus respectivas tarjetas de control. El control original también emplea tres de éstas tarjetas, una por cada uno de los motores a mover

Diagrama electrónico de la tarjeta para microcontrolador

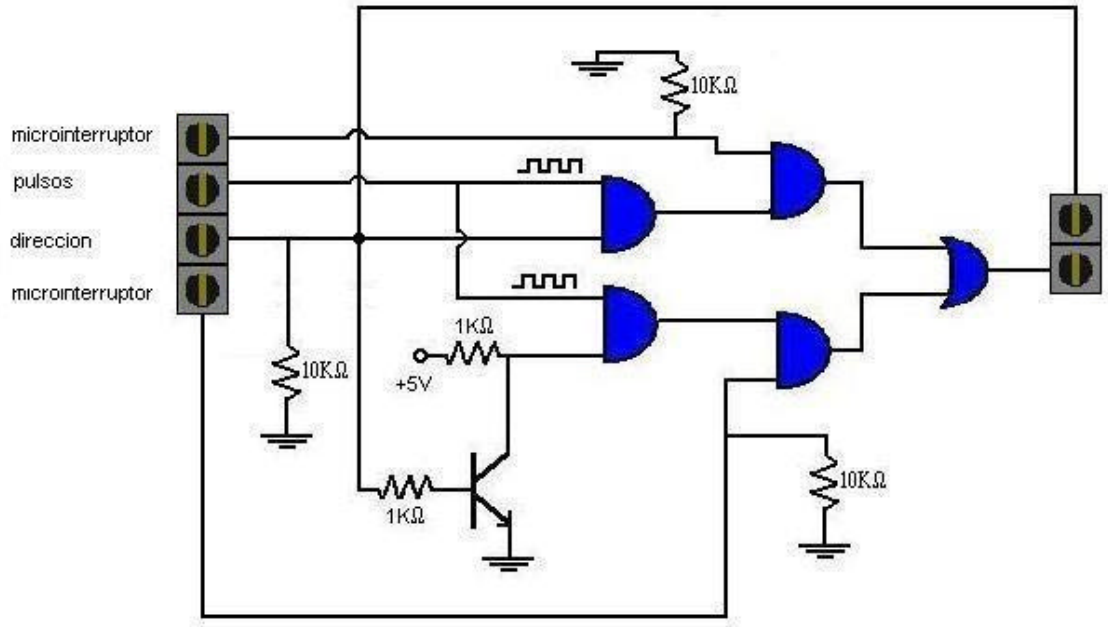


Elaborada por Alejandro Ruiz

Anexo D. Tarjeta de control de microinterruptor

Esta tarjeta es la encargada detener la máquina al recibir las señales de los microinterruptores de fin de carrera. Se requiere una tarjeta por cada eje que se desee controlar; cada eje debe de poseer dos microinterruptores (uno en cada extremo). En el diseño de este control se emplearon dos tarjetas de este tipo.

Diagrama electrónico de la tarjeta de control de microinterruptores

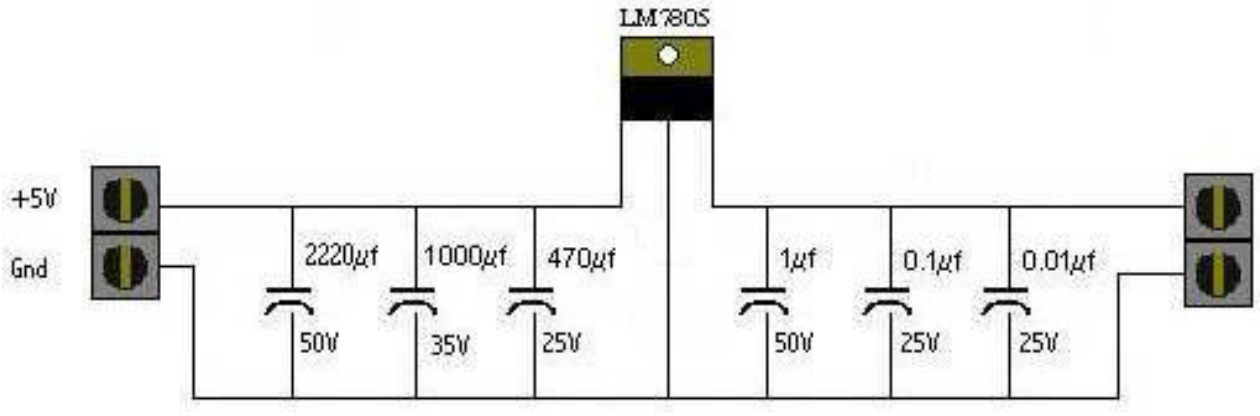


Elaborada por Alejandro Ruiz

Anexo E. Tarjeta reguladora de voltaje

Esta tarjeta se encarga de mantener constante el voltaje que entra de la fuente, evitando que si este pasa de 5 V se quemen los componentes diferentes componentes de las tarjetas y los microcontroladores.

Diagrama electrónico de la tarjeta reguladora de voltaje

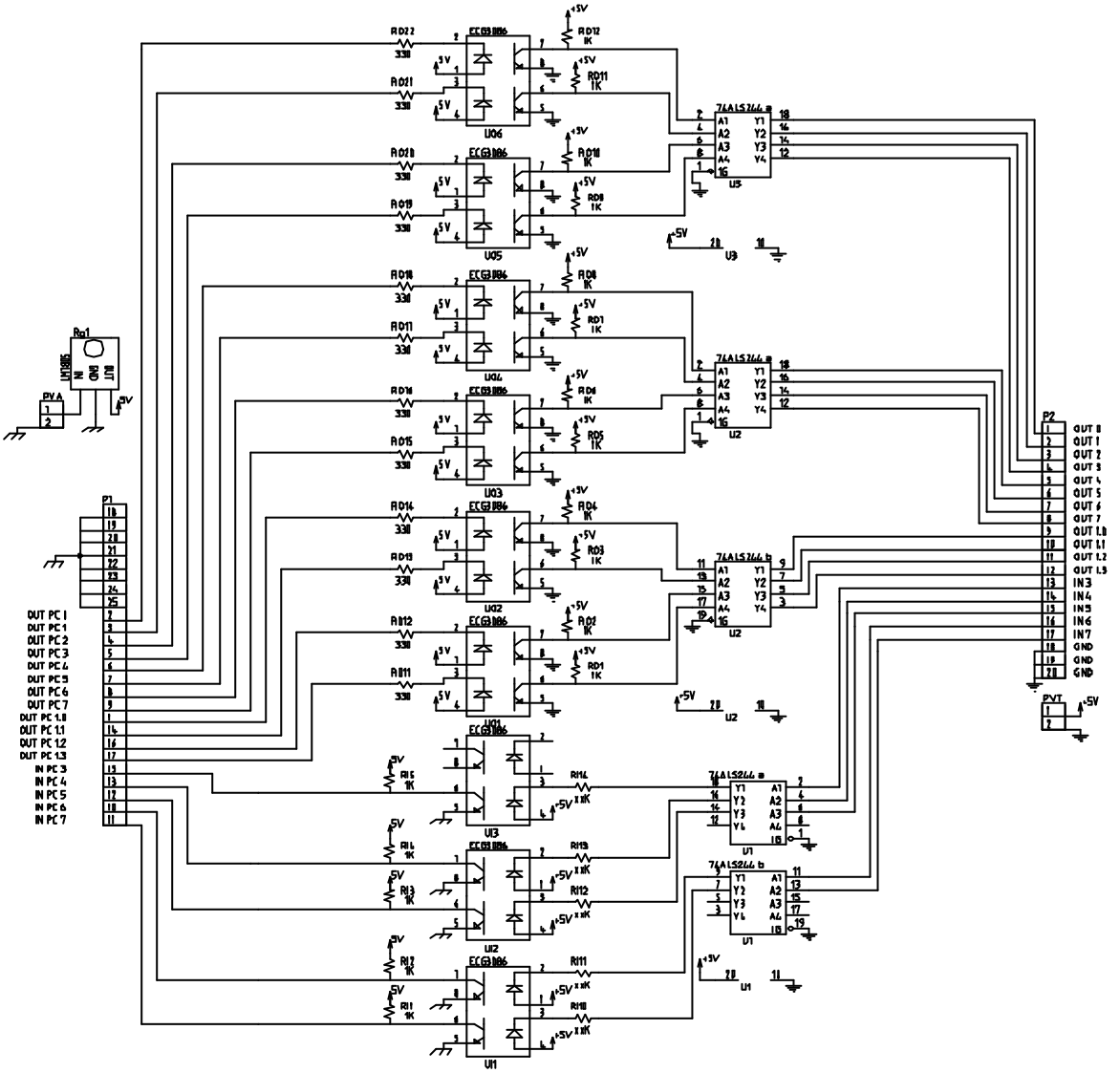


Elaborada por Alejandro Ruiz

Anexo F. Tarjeta de expansión del puerto paralelo

La función de esta tarjeta es la de recibir las señales provenientes directamente del puerto paralelo del computador y enviarlas al resto de las tarjetas, aislando al puerto del computador para protegerlo de daño debido a las corrientes y voltajes existentes en las tarjetas de control.

Diagrama electrónico de la tarjeta de expansión del puerto paralelo



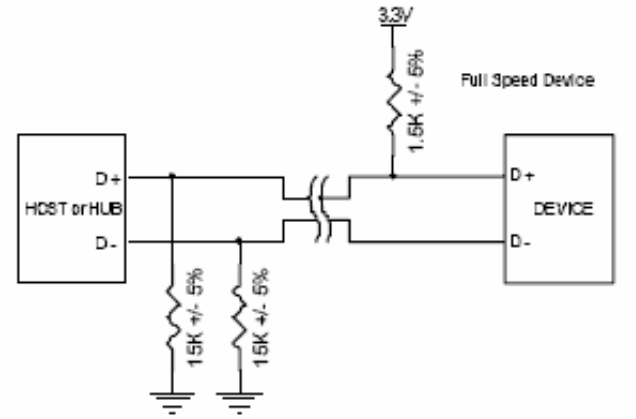
Elaborada por Edwin Giraldo, Laboratorio Control digital

Anexo G. Complemento protocolo USB

Identificación de la velocidad

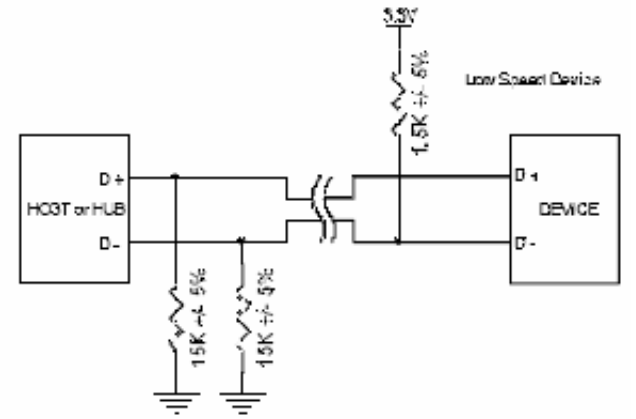
Un dispositivo USB debe indicar su velocidad subiendo D+ o D- a 3.3 V, un dispositivo de alta velocidad usara una resistencia *pull up* pegada a D+ para identificarse como dispositivo de alta velocidad, estas resistencias de *pull up* también serán usadas por el *Host* para detectar la presencia de un dispositivo conectado al puerto, sin el *pull up* USB asume que no hay nada conectado al bus, algunos controladores tienen esta resistencia dentro que es activada y desactivada mediante el uso del *firmware* y otras necesitan una resistencia externa.

Dispositivo Full Speed con resistencia en pull up conectada a D+



Fuente www.beyondlogic.com

Dispositivo Low Speed con resistencia en pull up conectada a D-



Fuente www.beyondlogic.com

Poder (Vbus)
USB permite que los dispositivos tomen la energía que necesitan del bus, no se necesita poder externo ni cables adicionales, de cualquier manera hay que tener en cuenta varios criterios.

Un dispositivo USB especifica el consumo de energía expresado en unidades de 2mA en el descriptor de configuración. Un dispositivo no puede incrementar su consumo de energía, más de lo que ha especificado durante la enumeración, ni siquiera si pierde poder externamente. Hay tres clases de funcionamiento para USB.

- Energizado por el bus, Bajo poder.
- Energizado por el bus, Alto poder.
- Energizado con poder propio.

El funcionamiento de bajo poder energizado por el bus toma toda la energía de V_{BUS} y no puede tomar más de una unidad de carga, la especificación define una unidad de carga como 100mA. El funcionamiento a bajo poder también debe ser diseñado para tomar un voltaje entre 4.40V y 5.25V medido en la parte final del cable, para muchos dispositivos de 3.3v, son necesarios los reguladores LDO.

El funcionamiento de alto poder energizado por el bus toma toda la energía del bus y no puede tomar más de una unidad de carga hasta que ha sido configurado, después de lo cual puede tomar 5 unidades de carga (500mA Máximo). El funcionamiento de alto poder debe ser detectado y enumerado a un mínimo de 4.40V mientras opera a con una unidad de carga.

El funcionamiento con poder propio puede tomar hasta una unidad de carga del bus y tomar el resto de una unidad externa. Si la fuente externa falla, el dispositivo debe tener provisiones para no tomar más de una unidad de carga del bus, la unidad que provee el bus permite la detección y enumeración de las unidades sin estar conectados a la fuente externa.

Ningún dispositivo ya sea energizado por el bus o no, puede conducir el V_{BUS} , si se pierde el voltaje en V_{BUS} el dispositivo tiene 10 segundos para remover el poder de las resistencias de *pull up* en D+ / D- usadas para la identificación de velocidad.

Estado Suspendido

El estado de suspendido es necesario en todos los dispositivos, en el estado de suspendido hay consideraciones adicionales para tener en cuenta.

La corriente máxima mientras está suspendido es proporcional a la unidad de carga, para un dispositivo con una unidad de carga, la máxima corriente mientras está suspendido es de 500uA, incluyendo la corriente de las resistencias de *pull up* en el bus. En el *hub* D+ y D- tienen resistencias de *pull down* de 15K? . Debido al consumo de energía la resistencia de *pull down* en el dispositivo está en el orden de 1.5K? , generando una carga total de 16.5 K? en un pin de voltaje que casi siempre carga 3.3 V, de este modo la resistencia toma 200uA sin siquiera empezar.

Un dispositivo USB entra a estado suspendido cuando no hay actividad en el bus por más de 3.0ms, después tiene 7ms más para apagar el dispositivo y no tomar más de la designada corriente de suspensión y solo debe estar tomando está corriente 10ms después de que ha terminado la actividad en el bus. Para mantenerse conectado a un *host* o *hub* suspendido, el dispositivo debe proveer energía a las resistencias de *pull up* que sirven para la identificación de la velocidad durante la suspensión.

USB tiene un paquete de datos para mantener con vida el dispositivo, el cual se envía periódicamente por el bus, esto previene un bus ocioso de entrar en estado suspendido.

Un bus de baja velocidad tiene un paquete que lo mantiene con vida, es un EOP (*End Of Packet*) el cual se envía cada 1ms solo en la ausencia de datos de baja velocidad.

Protocolo USB

A diferencia de RS-232 u otras interfaces de comunicación serial parecidas, donde el formato de los datos enviados no está definido, USB está hecho sobre varias capas de protocolos, la mayoría de los microchips con capacidad para USB se

ocupan de las capas inferiores de protocolo, haciendo las mismas invisibles para el diseñador.

Cada transacción en USB se compone de un grupo de paquetes

- *Token* (cabecera que define lo que se espera que lo siga).
- Datos opcionales (conteniendo el cuerpo del mensaje).
- Status (usado como *handshake* y provee medios para la corrección de errores).

El USB es un bus centrado en el *Host*, el cual inicia todas las transacciones. El primer paquete también llamado *Token* es generado por el *Host* para describir que es lo que lo sigue y si la transacción es de lectura o escritura además de la dirección del dispositivo y el *endpoint* que va a usar. El siguiente paquete es por lo regular un paquete con los datos y es seguido de un paquete de *handshake* (confirmación), que reporta si los datos o el *token* han sido recibidos exitosamente o si el *endpoint* esta fuera de servicio o no puede aceptar los datos.

Campos de los paquetes en USB

En el bus USB los datos se transmiten empezando por el bit menos significativo, los paquetes están compuestos por los siguientes campos.

Sync todos los paquetes deben empezar con un campo de sincronización, este campo tiene 8 bits y es usado para sincronizar el reloj del receptor con el del transmisor, los últimos dos bits indican donde empieza el campo de PID.

PID (packet ID) este campo es usado para identificar el tipo de paquete que se manda, en la tabla se muestran los posibles valores.

Identificación de paquetes

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

Fuente www.beyonlogic.com

El PID tiene 4 bits, pero para garantizar la correcta recepción los 4 bits son complementados y repetidos, haciendo un PID de 8 bits en total resultando en el siguiente formato.

Estructura de un PID de 8 bits

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

Fuente www.beyonlogic.com

ADDR el campo de dirección especifica a cual dispositivo se le está enviando el paquete, tiene 7 bits lo que le permite soportar 127 dispositivos, la dirección cero no es valida ya que cualquier dispositivo que aún no tiene asignada una dirección debe responder a los paquetes enviados a la dirección cero.

ENDP el campo *endpoint* tiene 4 bits, permitiendo el uso de 16 *endpoints*, de cualquier manera los dispositivos de baja velocidad solo tienen espacio para 2 *endpoints* más además de los establecidos (4 *endpoints* máximo).

CRC los chequeos de redundancia cíclica se hacen al paquete de datos, todos los paquetes de *token* tienen un CRC de 5 bits, mientras que los paquetes de datos tienen uno de 16 bits.

EOP el final de paquete señalado por un (SE0) *single ended zero* por aproximadamente el tiempo de 2 bits y seguido de por una J por el tiempo de un bit.

Tipos de paquete USB

USB tiene 4 diferentes tipos de paquete. Los paquetes *Token* indican el tipo de transacción a seguir, los paquetes de datos contiene la carga de datos, los paquetes de confirmación son usados para confirmar el envío de datos o reportar errores y los paquetes de inicio de marco indican el inicio de un nuevo marco.

Paquetes Token

Hay tres tipos.

- In - Informa al dispositivo USB el deseo del *Host* para leer información.
- Out – Informa al dispositivo USB que el *Host* desea enviar información.
- *SetUp* – Usado para iniciar las transferencias de control.

Formato de los paquetes Token

Sync	PID	ADDR	ENDP	CRC5	EOP

Fuente www.beyonlogic.com

Paquetes de Datos

Hay dos tipos, cada uno capaz de transmitir de 0 a 1023 bytes de datos.

- Data1

- Data2

Formato de los paquetes de datos

Sync	PID	Data	CRC16	EOP

Fuente www.beyonlogic.com

Paquetes de Confirmación

Hay tres tipos de paquetes de confirmación que constan simplemente del PID.

- **Ack** – Confirma que el paquete ha sido recibido exitosamente.

- **Nak** – reporta que el dispositivo no puede mandar ni recibir, datos temporalmente. También usado durante la transacción de interrupciones para informar al *Host* que no hay datos para enviar.

- **Stall** – el dispositivo se encuentra en un estado en el cual requiere intervención del *Host*.

Formato de los paquetes de confirmación

Sync	PID	EOP

Fuente www.beyonlogic.com

Paquetes de Inicio de Marco

Consiste en un número de marco de 11 bits enviado por el *Host* cada 1 mS ± 500 nS

Formato de los paquetes de inicio de Marco

Sync	PID	Frame Number	CRC5	EOP

Fuente www.beyonlogic.com

El Standard de USB hace referencia a funciones USB que pueden ser vistas como dispositivos USB que proveen la capacidad o función de impresoras, drivers Zip, Scanner, módems u otros periféricos. La mayoría de las funciones USB se encargan de las capas bajas del protocolo USB, hasta la capa de transacción en

los chips. Las funciones tienen una serie de *buffers*, normalmente de 8 bytes, cada buffer pertenece a un *endpoint*.

Por ejemplo, el *Host* envía una petición del descriptor de un dispositivo, la función en hardware lee el paquete de *SetUp* y determina del campo de dirección si el paquete es para él, si lo es copia los datos del paquete de datos siguiente al buffer del *endpoint* apropiado dictaminado por el valor del campo *endpoint* en el *Token* de *SetUp*. Después mandará el paquete de confirmación para confirmar la recepción del byte y genera una interrupción interna al semiconductor / microcontrolador para el *endpoint* apropiado lo que significa que ha recibido un paquete. Esto normalmente lo hace el hardware.

Ahora el software recibe la interrupción y debería leer los contenidos del buffer del *endpoint* y tramitar la petición del descriptor.

Endpoints

Los *endpoints* pueden ser descritos como fuentes o bodegas de datos. Como el bus es centrado en el *Host* los *endpoints* ocurren al final del canal de comunicaciones en la función USB. En la capa de software, el *driver* del dispositivo podrá enviar un paquete al *endpoint* 1 del dispositivo. Ya que los datos están saliendo del *Host*, van a terminar en el buffer EP1 OUT, el *firmware* entonces podrá disponer de los datos, en caso de que vaya a devolver datos, la función no puede simplemente escribir en el bus ya que este es controlado por el *Host*, entonces escribe los datos en el buffer EP1 IN, los datos permanecen en el buffer hasta que el *Host* envía un paquete IN a ese *endpoint*, haciendo un requerimiento de los datos. Los endpoint también pueden ser vistos como la interfaz entre el hardware del dispositivo y el *firmware* corriendo en el dispositivo.

Todos los dispositivos deben soportar *endpoint* 0 ya que este es el *endpoint* que recibe los requerimientos de control y status durante la enumeración y mientras que el dispositivo este operando en el bus.

Mientras que el dispositivo envía y recibe datos por los *endpoint*, el software encargado transfiere los datos por tuberías. Una tubería es una conexión lógica entre el *Host* y los *endpoint*. Las tuberías también tienen una serie de parámetros asociados a ellas tales como la que ancho de banda tiene, que tipo de transferencia usa (control, *bulk*, sincrónico, interrupción), la dirección del flujo de datos y el tamaño máximo de buffer y paquete. Por ejemplo la tubería por defecto es una tubería bi-direccional hecha de EP0 IN y EP0 OUT con transferencia tipo control.

USB define dos tipos de tuberías

- ***Stream Pipes*** no tienen definido formato USB, lo que significa que pueden transferir cualquier tipo de dato. Los datos fluyen secuencialmente y tiene una dirección predefinida, ya sea de salida o de entrada, estas tuberías soportan transferencias tipo *bulk*, sincrónico e interrupción. Estas tuberías pueden ser controladas ya sea por el *Host* o por el dispositivo.

- **Message pipes** tienen un formato USB definido. Son controladas por el *Host*, son iniciadas por una petición enviada por el *Host*, después los datos son enviados en la dirección deseada dada por la petición, es así que estas tuberías permiten que los datos fluyan en ambas direcciones pero solo soportan transferencias tipo control.

Tipos de *Endpoint*
La especificación de USB define cuatro tipos de transferencia y *endpoint*.

- Transferencia de Control.

- Transferencia por Interrupción.

- Transferencia Sincrónica.

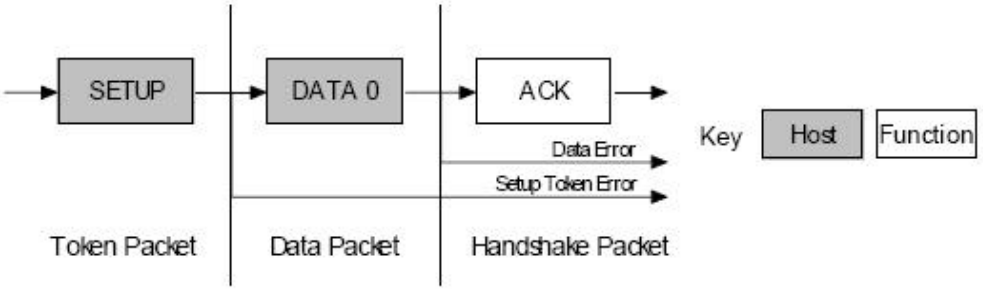
- Transferencia *Bulk*.

Transferencia de Control
Las transferencias de control son normalmente usadas para operaciones de comando y status. Se usan esencialmente para iniciar el dispositivo con todas las funciones de enumeración. Normalmente son paquetes al azar iniciados por el *Host*, el tamaño del paquete en las transferencias de control en dispositivos lentos es de 8 bytes, dispositivos de alta velocidad permiten paquetes de 8, 16, 32 ó 64 bytes y los dispositivos de máxima velocidad tienen que tener un tamaño de paquete de 64 bytes.

Una transferencia de control puede tener hasta 3 partes

La **parte de *SetUp*** es donde se envía la petición la cual consiste de tres paquetes. El *Token* de *SetUp* es enviado primero y contiene la dirección y el número de *endpoint*, luego va el paquete de datos y siempre tiene un PID y es tipo data0 además contiene un paquete de *SetUp* el cual detalla el tipo de petición. El último paquete es una confirmación para indicar la recepción correcta o indicar un error, si la función recibe exitosamente los datos de inicialización responde con un ACK, sino ignora los datos y no envía un paquete de *SetUp*, las funciones no pueden responder con Stall o NAK a un paquete de *SetUp*.

Parte de Control parte de Set up



Fuente www.beyonlogic.com

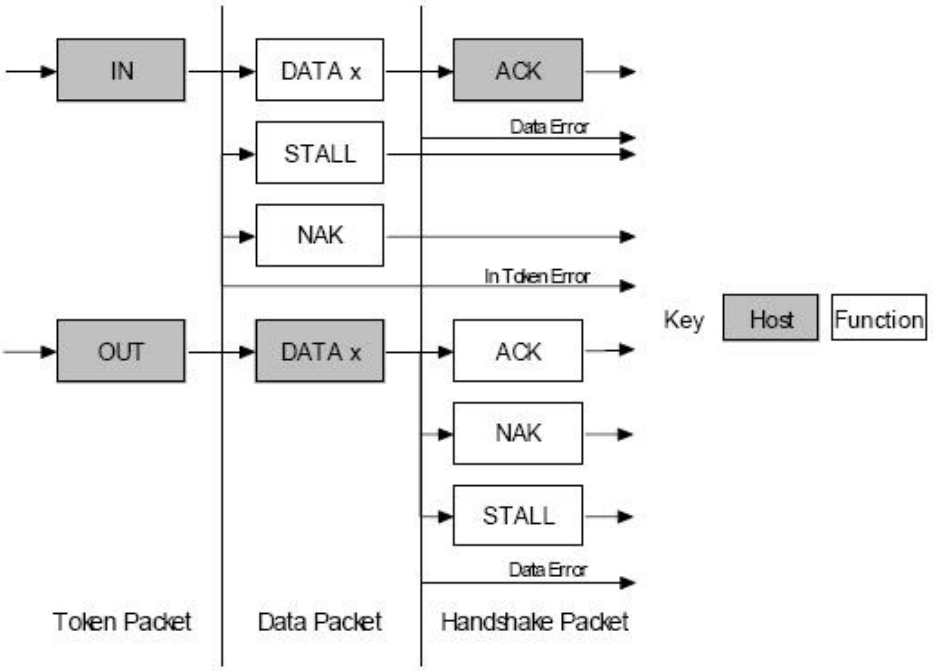
La **parte de datos** es opcional y consiste en una o múltiples transferencias IN u OUT, la petición de *SetUp* indica la cantidad de datos a ser transmitidos en esta parte. Si estos exceden el tamaño máximo del paquete entonces serán enviados en múltiples transmisiones cada una con un paquete de tamaño máximo menos la última.

La parte de datos tiene dos diferentes escenarios dependiendo de la dirección de la transferencia de datos.

- **IN**: cuando el *Host* está listo para recibir los datos de control produce un IN *Token*, si la función recibe el IN *Token* con un error entonces ignora el paquete, si el *Token* se recibió correctamente, el dispositivo puede responder con un paquete de datos conteniendo los datos de control a ser enviados, un paquete *Stall* indicando que el *endpoint* ha tenido un error o

un NAK indicando al *Host* que el *endpoint* está trabajando, pero temporalmente no tiene datos para enviar.

Parte de Control parte de datos



Fuente www.beyonlogic.com

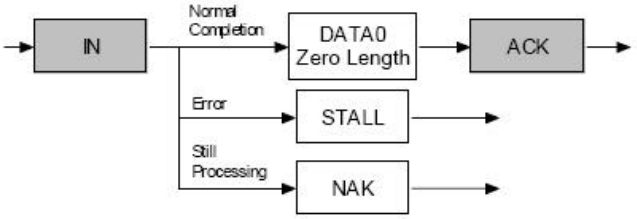
- **OUT**: cuando el *Host* necesita enviarle al dispositivo un paquete de control, envía un *Token* OUT seguido de un paquete de datos conteniendo los datos de control. Si cualquier parte del *Token* o del paquete de datos está corrupto la función ignora el paquete. Si el buffer del *endpoint* en la función estaba vacío y guardó los datos en el buffer, envía un ACK informando al *Host* que ha recibido satisfactoriamente los datos, si el buffer del *endpoint* no está vacío debido al procesamiento del paquete anterior, entonces la función retorna un NAK. Pero si el *endpoint* ha provocado un error y el bit de parada se ha prendido, la función devuelve un *Stall*.

La parte de status reporta el estado de la petición y varía dependiendo de la dirección de la transferencia. El reporte de estado es siempre hecho por la función.

- **IN**: si el *Host* envió *Tokens* IN durante la parte de datos para recibir datos, entonces el *Host* tiene que confirmar la adecuada recepción de dichos

datos. Esto es hecho por el *Host* enviando un *Token* OUT seguido de un paquete de datos de tamaño cero. La función ya puede reportar su estado en la confirmación. Un ACK indica que la función ha completado el comando y está lista para recibir un nuevo comando, si ocurre un error durante el procesamiento de este comando, la función enviará un Stall, pero si la función está aún procesando, devuelve un NAK indicando al *Host* que repita la parte de status después.

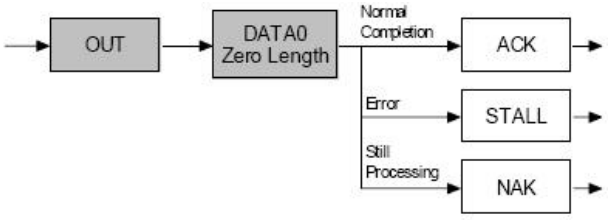
Parte de Control parte de datos IN



Fuente www.beyonlogic.com

- **OUT**: si el *Host* envió *Tokens* OUT durante la parte de datos para transmitir datos, la función confirmará la exitosa recepción de la información enviando un paquete de datos de tamaño cero en respuesta a un *Token* IN. Pero si ocurre un error, deberá enviar un *Stall* o si sigue ocupado procesando datos, deberá enviar un NAK pidiendo al *Host* que trate de nuevo más tarde.

Parte de Control parte de datos OUT



Fuente www.beyonlogic.com

Transferencia por Interrupción

Las interrupciones siempre son generadas por el dispositivo. Pero desde USB si un dispositivo requiere la atención del *Host*, tiene que esperar hasta que el *Host* lo autorice antes de que pueda reportar que necesita atención urgente.
Las transferencias por interrupción

- Garantizan latencia.

- *Stream pipe* unidireccional.

- Detección de error y nuevo intento en el periodo siguiente.

Las transferencias por interrupción no son periódicas ideales para dispositivos pequeños que requieren latencia limitada. Un requerimiento de interrupción es puesto en cola por el dispositivo hasta que el USB autorice al dispositivo y pida información.

- El tamaño máximo de datos para dispositivos de baja velocidad es de 8 bytes.

- El tamaño máximo de datos para dispositivos de alta velocidad es de 64 bytes.

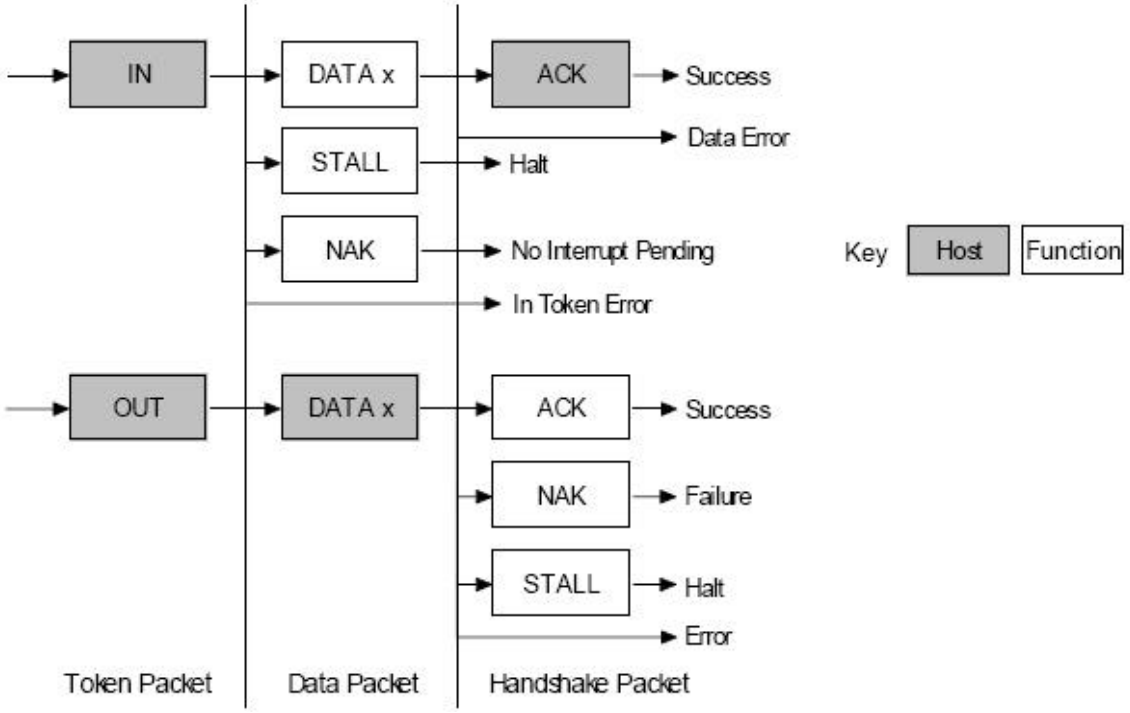
- El tamaño máximo de datos para dispositivos de máxima velocidad es de 1024 bytes.

- **IN**: el *Host* periódicamente autorizará las interrupciones del *endpoint*. Esta frecuencia de autorización está especificada en el descriptor del *endpoint*. Cada autorización envuelve al *Host* enviando un *Token* IN, si el *Token* IN está corrupto la función ignora el paquete y continua monitoreando el bus en espera de nuevos *Tokens*.

  Si una interrupción ha sido puesta en cola por el dispositivo, la función enviará un paquete de datos conteniendo datos relevantes a la interrupción cuando reciba el *Token* IN. Después de una recepción exitosa por el *Host*, este devolverá un ACK. Si los datos están corruptos, el *Host* no devolverá estado. En otro caso si no hay interrupción presente cuando el *Host* autorizó las interrupciones del *endpoint* con el *Token* IN, la función señala su estado enviando un NAK. Si ocurrió un error en este *endpoint* un *Stall* es enviado en respuesta al *Token* IN.

- **OUT**: cuando el *Host* quiere enviar datos de interrupción al dispositivo, envía un *Token* OUT seguido del paquete de datos conteniendo los datos de la interrupción. Si cualquiera de los paquetes está corrupto entonces la función ignora el paquete. Si el buffer del *endpoint* estaba vacío y recibió los datos entonces envía un ACK informando al *Host* que recibió satisfactoriamente los datos, si por algún motivo el buffer no está vacío entonces la función devuelve un NAK, pero si ocurre un error con el *endpoint* y está activado el bit de parada entonces envía un *Stall*.

Transferencia por Interrupción



Fuente www.beyonlogic.com

Transferencia Sincrónica

Las transferencias sincrónicas ocurren periódica y continuamente, normalmente contienen información sensitiva al tiempo tal como cadenas de audio y video. Si hubiera algún retardo o un reenvío en una cadena de audio se esperaría saltos en el audio y errores en el sonido, pero si se pierde algún marco de vez en cuando es menos probable que se de cuenta quien escucha.

Las transferencias sincrónicas proveen

- Acceso garantizado al ancho de banda de USB.
- Latencia limitada

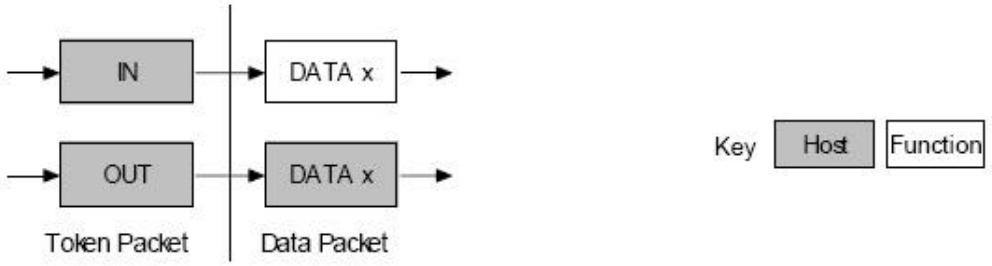
- *Stream pipe* unidireccional.

- Detección de errores por CRC, pero no reenvío ni garantía de entrega.

- Solo para niveles de alta o máxima velocidad.

El tamaño máximo de los datos está especificado en el descriptor del *endpoint* de un *endpoint* sincrónico, este puede ser de hasta 1023 bytes para un dispositivo de máxima velocidad. Ya que el tamaño máximo de los datos afectará los requerimientos de ancho de banda en el bus, es importante especificar un tamaño de datos prudente. Si se está usando un tamaño de datos grande es bueno especificar una serie de interfaces alternativas con tamaños de datos sincrónicos variados. Si durante la enumeración el *Host* no puede admitir el *endpoint* sincrónico preferido debido a restricciones en el ancho de banda, queda algo en que apoyarse en vez de que el dispositivo caiga del todo. Los datos enviados por un *endpoint* sincrónico pueden tener menos tamaño que el prenegociado y pueden variar en tamaño de transacción en tra nsacción.

Transferencia Sincrónica



Fuente www.beyonlogic.com

El diagrama anterior muestra el formato de una transacción sincrónica IN y OUT. Las transacciones sincrónicas no tienen un estado de confirmación y no pueden reportar errores ni condiciones de *Stall* / *Halt*.

Transferencias Bulk

Las trasferencias *Bulk* pueden ser usadas para enviar datos secuenciales grandes, como trabajos de impresión enviados a una impresora o imágenes generadas por un scanner. Las transferencias *Bulk* proveen corrección de errores con un campo CRC16 en el área de datos y mecanismos para detección de errores y retransmisión asegurando así que los datos son enviados y recibidos sin error.

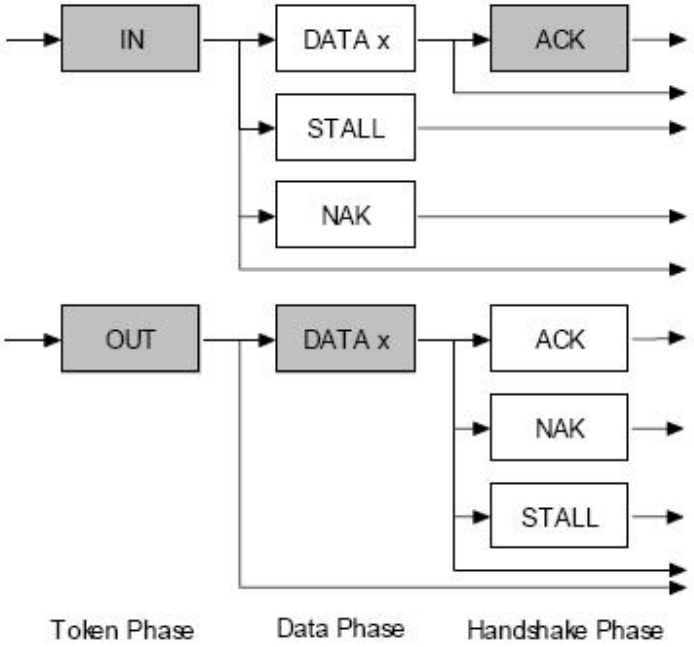
Las transferencias *Bulk* usarán el ancho de banda de sobra en el bus después de que las demás transacciones han sido localizadas, si el bus está ocupado con transacciones sincrónicas o de interrupción entonces los datos *Bulk* viajarán lentamente por el bus. Como resultado las transacciones *Bulk* deben ser usadas solo para comunicaciones no sensibles al tiempo ya que no hay garantía de latencia.

Las transferencias *Bulk*

- Se usan para transferir datos continuos y grandes.
- Detección de errores vía CRC, con garantía de entrega.
- No hay garantía de ancho de banda o mínima latencia.
- *Stream pipe* unidireccional.
- Alta y máxima velocidad únicamente.

Las transferencias *Bulk* son solo soportadas por dispositivos de alta y máxima velocidad. Para *endpoint* de alta velocidad el tamaño de paquete máximo es de 8, 16, 32 ó 64 bytes, para *endpoint* de máxima velocidad el tamaño del paquete puede ser hasta de máximo 512 bytes. Si el tamaño de los datos cae por debajo del tamaño máximo, no necesita ser rellenado con ceros. Una transferencia *Bulk* es considerada completa cuando ha transmitido la exacta cantidad de datos requeridos, transferido un paquete menor al tamaño máximo o transferido un paquete de tamaño cero.

Transferencias Bulk



Fuente www.beyonlogic.com

- **IN**: cuando el *Host* está listo para recibir datos de una transferencia *Bulk* este envía un *Token* IN. Si la función recibe el *Token* con un error, ignora el paquete. Si el *Token* fue recibido correctamente, la función puede o enviar un paquete con los datos tipo *Bulk* o un paquete *Stall* indicando que el *endpoint* tiene un error o un NAK indicando al *Host* que el *endpoint* está trabajando pero no tiene datos para enviar.

- **OUT**: cuando el *Host* quiere enviarle a la función un paquete de datos *Bulk*, envía un *Token* OUT seguido del paquete de datos *Bulk*. Si el paquete de datos o el *Token* están corruptos la función ignora el paquete, si el *endpoint* de la función está vacío y recibió los datos, envía un ACK informando al *Host* la recepción exitosa de los datos. Si el *endpoint* no esta vacío devuelve un NAK. Si ha habido un error en el *endopoint* y el bit de alto está en uno, devuelve un *Stall*.

Administración del Ancho de Banda

El *Host* es responsable de administrar el ancho de banda del bus. Esto lo hace en la enumeración cuando configura los *endpoint* sincrónico y de interrupción y mediante la operación del bus. La especificación limita el bus, no permitiendo que

más del 90% de cualquier marco sea asignado a transferencias periódicas en un bus de alta velocidad. En buses de máxima velocidad la limitación es reducida a no más del 80% de un micro marco puede ser asignado a transferencias periódicas.

Paquete SetUp

Todo dispositivo USB tiene que responder a los paquetes *SetUp* por medio de la tubería por defecto. Los paquetes de *SetUp* se usan para detección y configuración del dispositivo y para llevar a cabo funciones comunes como inicializar la dirección USB del dispositivo, pedir un descriptor de dispositivo o chequear el estado de un *endpoint*.

Un *Host* espera que todos los pedidos sean respondidos en un periodo máximo de 5 segundos, se especifican tiempos más estrictos para algunos pedidos:

- Petición Standard de dispositivo sin parte de datos se debe responder en 50ms.

- Petición Standard de dispositivo con parte de datos se debe responder en 500ms.

    - Cada paquete de datos debe ser enviado en los 500ms siguientes a la transmisión exitosa del paquete anterior.

    - La parte de estado debe ser completada dentro de los 50ms después de la transmisión del último paquete de datos.

- El comando *SetAddress* (que contiene una parte de datos) debe procesar un comando y devolver el estado en 50ms. El dispositivo después tiene 2ms para cambiar la dirección antes de que la siguiente petición sea enviada.

Estos períodos son aceptables hasta para los dispositivos más lentos, pero puede haber una restricción durante la etapa de *debug*.

Cada petición comienza con un paquete de *SetUp* de 8 bytes que tiene el siguiente formato.

Formato del paquete Setup

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-Map	**D7 Data Phase Transfer Direction** 0 = Host to Device 1 = Device to Host **D6..5 Type** 0 = Standard 1 = Class 2 = Vendor 3 = Reserved **D4..0 Recipient** 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	bRequest	1	Value	Request
2	wValue	2	Value	Value
4	wIndex	2	Index or Offset	Index
6	wLenght	2	Count	Number of bytes to transfer if there is a data phase

Fuente www.beyondlogic.com

El campo **bmRequestType** determina la dirección de la petición, tipo de petición y recipiente designado.

El campo **bRequest** determina la petición hecha. El **bmRequestType** es normalmente analizado y su ejecución dividida en un número de *handlers* tales como el *handler* de la petición Standard de dispositivo, el *handler* de la petición estándar de interfaz, el *handler* de la petición Estándar de *endpoint*, el *handler* de la petición de clase de dispositivo etc. El análisis del paquete de *SetUp* puede ser hecho a preferencia del usuario. Otros pueden elegir analizar el **bRequest** primero y después determinar el tipo y el recipiente basado en cada petición.

Las peticiones Estándar son comunes en todos los dispositivos USB. Las peticiones de clase son comunes a clases de *driver*. Por ejemplo todos los dispositivos que caben dentro de la clase HID tendrán un listado de peticiones específicas a la clase, común para todos. Estas son diferentes a las de un dispositivo de la clase de comunicaciones o de la clase de almacenamiento en masa.

Al final de todo están las peticiones definidas por el vendedor. Estas son peticiones que el diseñador del dispositivo puede asignar. Son diferentes de dispositivo a dispositivo y son totalmente diseñadas por el creador del dispositivo. Una petición común puede estar dirigida a diferentes recipientes y basado en el recipiente desarrollar diferentes funciones. Una petición Estándar de *GetStatus* por ejemplo, puede ser dirigida a un dispositivo, interfaz o *endpoint*. Cuando es dirigida a un dispositivo devuelve banderas indicándole estado de despertar remoto y si el dispositivo tiene poder propio. Si la misma petición se envía a una interfaz siempre devuelve cero, y si se envía a un *endpoint* devolverá la bandera de alto para el *endpoint*.

Los campos **wValue** y **wIndex** permiten el envío de parámetros con la petición **wLength,** se usa para especificar el número de bytes a ser transferidos en caso de que haya una parte de datos.

Enumeración

La enumeración es el proceso de determinar que dispositivo se acaba de conectar al bus y que parámetros requiere, tales como el consumo de energía , número y tipo de *endpoint*, clase de producto etc. El *Host* asignará entonces una dirección al dispositivo y activará una configuración permitiendo al dispositivo transmitir datos por el bus.

Una enumeración común en Windows contiene los siguientes pasos.

1. El *Host* o *hub* detecta la conexión de un nuevo dispositivo por medio de las resistencias de *pull up* del dispositivo. El *Host* espera por lo menos 100ms permitiendo al conector ser insertado por completo y a que se estabilice el poder en el dispositivo.

2. El *Host* produce un reset poniendo el dispositivo en el estado por defecto. Ahora el dispositivo puede responder a la dirección por defecto (0).

3. El *Host* de MS Windows pide los primeros 64 bytes del descriptor de dispositivo.

4. Después de recibir los primeros 8 bytes del descriptor, inmediatamente produce otro reset en el bus.

5. El *Host* ahora produce un comando *Set Address* y cambia al dispositivo al estado *Addressed*.

6. El *Host* pregunta por los 18 bytes del descriptor del dispositivo.

7. luego pregunta por los 9 bytes del descriptor de configuración para determinar el tamaño total.

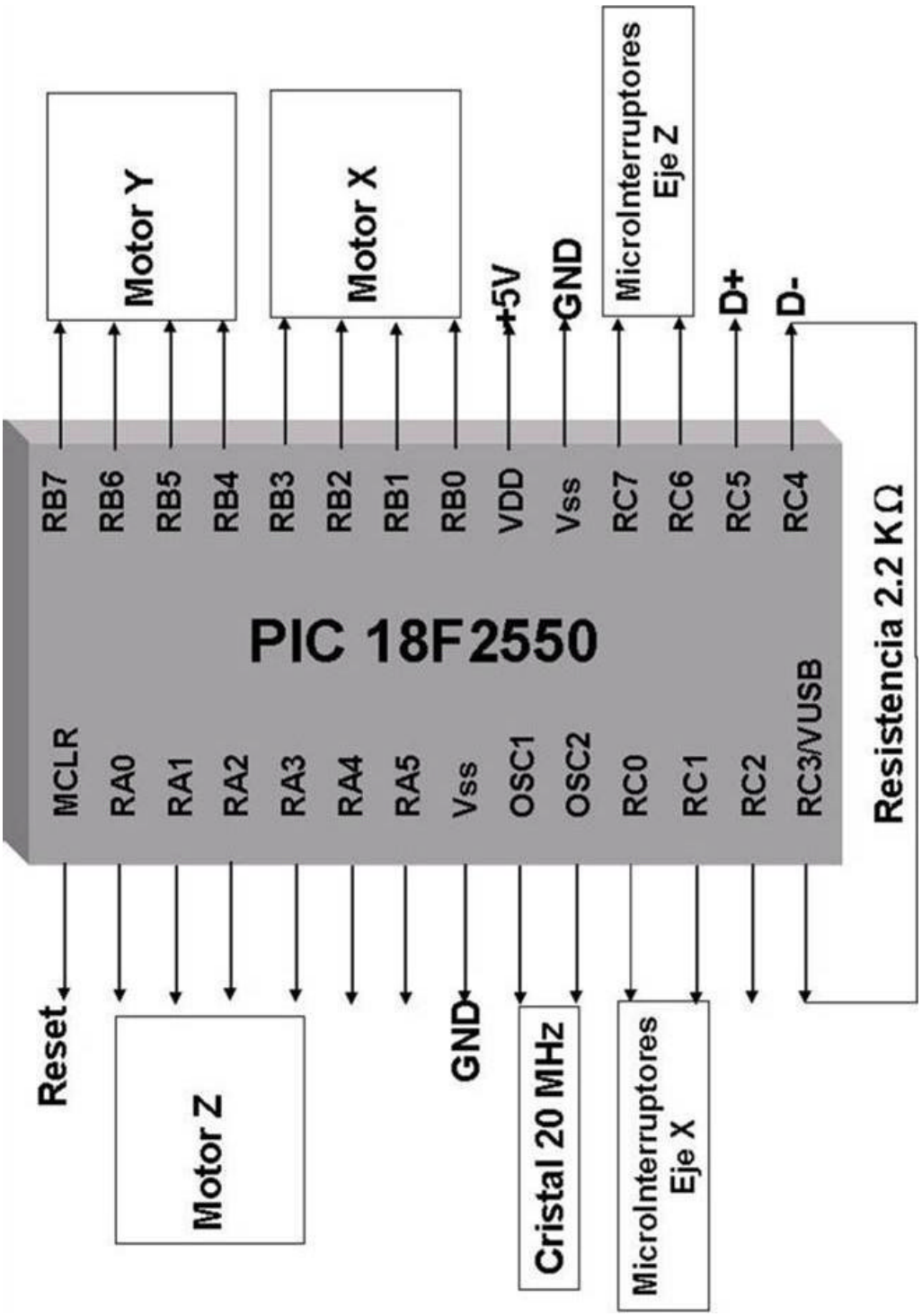
8. El *Host* pregunta por los 255 bytes del descriptor de configuración.

9. El *Host* pregunta por los descriptores de cadena si es que estos fueron especificados.

Al final del paso 9, Windows pregunta por el *driver* del dispositivo. Es común que vuelva a pedir todos los descriptores de nuevo antes de hacer la petición de *Set Configuration*.

Este proceso de enumeración es común para Windows 2000, Xp y 98 SE. Normalmente cuando algo anda mal con un descriptor o la horma como se está enviando, el *Host* intentará leerlo 3 veces con pausas largas entre cada intento. Depuse del tercer intento, el *Host* se rinde reportando un error con el dispositivo.

Anexo H. Distribución de Pines del PIC 18F2550



Distribución de Pines del PIC 18F2550

Fuente Vanessa Rodríguez Lora

Anexo I. Manual del sistema

CORBAL 3000 

Manual del sistema

- **Requisitos del sistema**
    - Sistema Operativo Windows XP
    - Procesador Pentium IV
    - 256 Mb de memoria RAM
    - Espacio en disco 5 Mb
- **Software**
    - Corbal.exe
- **Otros**
    - Máquina cortadora CORBAL 3000
    - Tarjeta controladora
    - Fuente variable 5V 2ª

Conexión

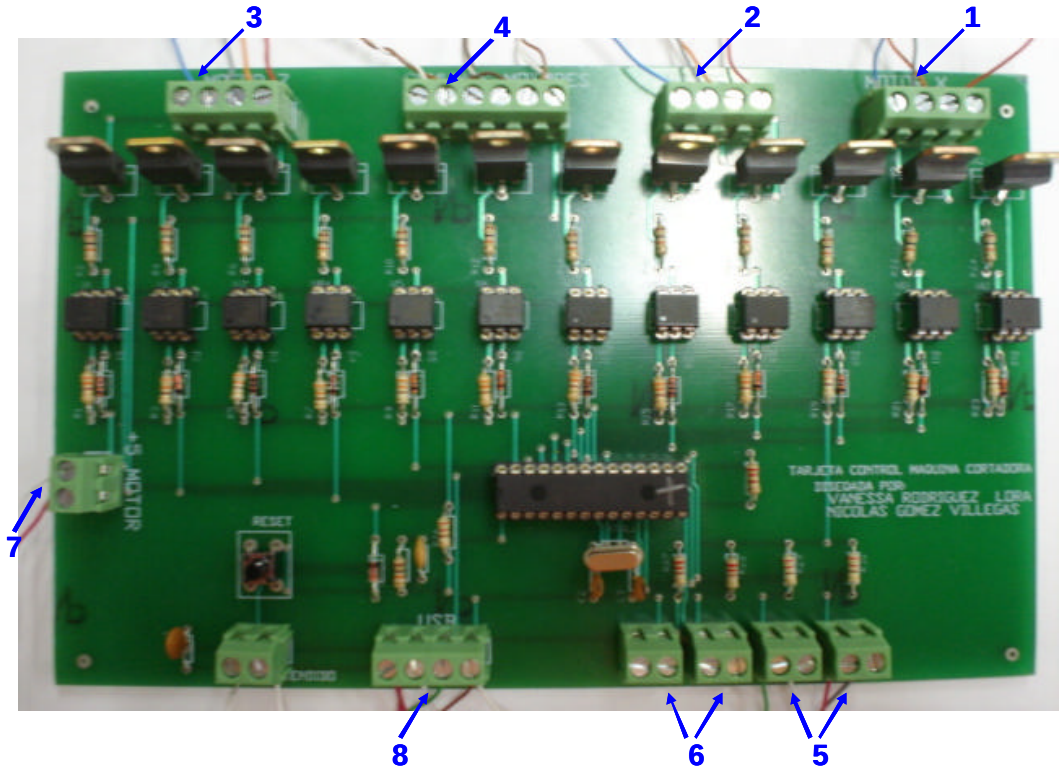
Con el objeto de evitar daños en los componentes del control y por ende problemas en su funcionamiento las conexiones deben mostrarse tal y como se muestran el la figura. Los números indican que conexión debe realizarse.

1. Motor X
2. Motor Y
3. Motor Z
4. Comunes
5. Interruptor eje X

6. Interruptor eje Z

7. Alimentación motores

8. Conexión puerto USB



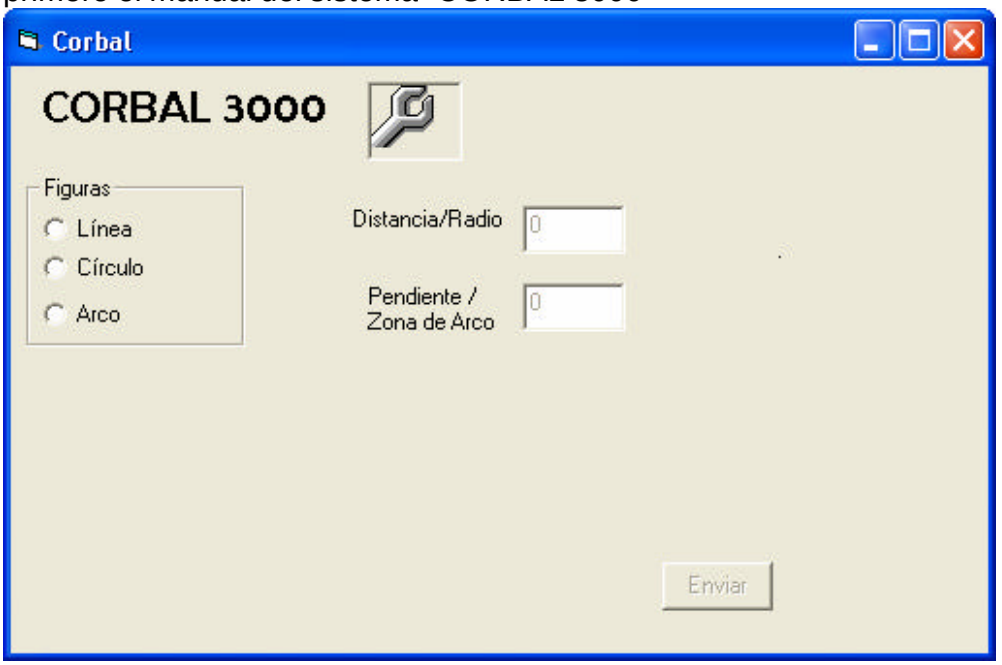
Anexo J. Manual de Usuario

CORBAL 3000

Manual de Usuario

- **Instalación del Programa**
  - No es necesario instalar el programa de prueba, solo basta con correr el ejecutable “CORBAL.exe” cada vez que se necesite emplear la máquina.
- Como usar el programa

Una vez inicializado el programa puede iniciarse su ejecución, pero primero debe cerciorarse que la máquina esté correctamente conectada, para esto leer primero el manual del sistema “CORBAL 3000”



1. Inicializar el programa
2. Seleccionar el tipo de figura a cortar,
3. En el recuadro escribir los parámetros de la figura a cortar en milímetros mm (distancia, radio) y en radianes para la pendiente.
4. Oprimir enviar para mandar los datos y que la máquina inicie el proceso de corte.

Una vez la máquina termina el corte se para su ejecución.

Anexo K. Código programa de prueba en Visual Basic

```
Private Sub Command2_Click()
Dim decenas As Double
Dim centenas As Double
Dim enteroA As Double
Dim dec1 As Double
Dim dec2 As Double
Dim dec3 As Double

Dim buffer() As Byte
'define buffer to use for data
'in this configuration you can only send 8 bytes at a time
ReDim buffer(8)

'trap errors if non numeric data in
'text box, or textbox empty
On Error Resume Next

'disable send button until result received
Command2.Enabled = False
On Error Resume Next
decenas = Text1.Text Mod 100
centenas = Int(Text1.Text / 100)
Label6.Caption = Atn(Text2.Text)


'dec1 = Int((Label6.Caption * 1000) / 100) Mod 10
'dec2 = Int(((Label6.Caption * 1000) Mod 100) / 10)
'dec3 = ((Label6.Caption * 1000) Mod 100) Mod 10

enteroA = Int(Text2.Text)
dec1 = Int((Text2.Text * 1000) / 100) Mod 10
dec2 = Int(((Text2.Text * 1000) Mod 100) / 10)
'dec3 = ((Text2.Text * 1000) Mod 100) Mod 10

'setup buffer to send data to PIC
'circulo
If (Option1.Value = True) Then
    buffer(0) = centenas
    buffer(1) = decenas
    buffer(2) = 0
    buffer(3) = 0
    buffer(4) = 0
    buffer(5) = 0
    buffer(6) = 2
    buffer(7) = 0

    HIDComm1.WriteTo buffer, 8

Else
'linea
If (Option2.Value = True) Then
    buffer(0) = centenas
    buffer(1) = decenas
    buffer(2) = enteroA
```

```
    buffer(3) = dec1
    buffer(4) = dec2
    buffer(5) = 0
    buffer(6) = 1
    buffer(7) = 0

    HIDComm1.WriteTo buffer, 8

Else
'arco
If (Option3.Value = True) Then
    buffer(0) = centenas
    buffer(1) = decenas
    buffer(2) = Text2.Text
    buffer(3) = 0
    buffer(4) = 0
    buffer(5) = 0
    buffer(6) = 3
    buffer(7) = 0

    HIDComm1.WriteTo buffer, 8
End If
End If
End If

Command2.Enabled = True

End Sub

Public Sub Form_Load()
Text2.Enabled = False
Text1.Enabled = False
'connect to the USB device as the program starts
 HIDComm1.Connect

End Sub

Private Sub Form_Terminate()
'disconnect from the USB device as prgoram ends
 HIDComm1.Uninit

End Sub

Private Sub HIDComm1_ConnectSuccess(ByVal Status As Long)
'enable button when device is connected
 Command1.Enabled = True
 Command2.Enabled = True
 Caption = "Corbal - Conectado"

End Sub

Private Sub HIDComm1_Disconnected(ByVal Status As Long)
'disable button when device unplugged
 Command1.Enabled = False
```

```
 Command2.Enabled = False
 Caption = "Corbal - Desconectado"

End Sub

Private Sub Option1_Click()
Text1.Enabled = True
Text2.Enabled = False
End Sub

Private Sub Option2_Click()
Text1.Enabled = True
Text2.Enabled = True
End Sub

Private Sub Option3_Click()
Text1.Enabled = True
Text2.Enabled = True
End Sub

Private Sub Timer1_Timer()
'try and reconnect PIC
 If HIDComm1.Connected = False Then
  HIDComm1.Connect
 End If

End Sub
```

Anexo L. Código programa de prueba en PICC

```
#include <18F2550.h>
#include <MATH.h>
 ////SLOW SPEED
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV3,VREGEN
#use delay(clock=24000000)
#define USB_USE_FULL_SPEED  FALSE
#DEFINE USB_HID_DEVICE  TRUE  //Tells the CCS PIC USB firmware
                              //to include HID handling code.

//turn on EP1 for IN interrupt transfers.  (IN = PIC -> PC)
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8     //max packet size of this endpoint
//the following defines needed for the CCS USB PIC driver to enable the
RX endpoint 1
// and allocate buffer space on the peripheral
#define USB_EP1_RX_ENABLE  USB_ENABLE_INTERRUPT   //turn on EP1 for OUT
bulk/interrupt transfers
#define USB_EP1_RX_SIZE    8  //allocate 8 bytes in the hardware for
reception
#include <pic18_usb.h>   //Microchip PIC18Fxx5x hardware layer for usb.c

#include <usb_desc_hid3.h>    //USB Configuration and Device descriptors
for this UBS device
#include <usb.c>        //handles usb setup tokens and get descriptor
reports
INT8 TIMERx=20,TIMERy=20;
INT8 CONTTA=0,CONTTB=0,CONTDE=0,CONTAB=0,CONTAR=0,CONTIZ=0;
FLOAT PRECX=0.02708;//0.03175;
FLOAT PRECY=0.02747;//0.02954;
BYTE CONST IZQUIERDA[4] = {0b00000001,
                           0b00000010,
                           0b00000100,
                           0b00001000};
BYTE CONST DERECHA[4]   = {0b00001000,
                           0b00000100,
                           0b00000010,
                           0b00000001};
BYTE CONST ABAJO[8]     = {0b00010000,
                           0b00110000,
                           0b00100000,
                           0b01100000,
                           0b01000000,
                           0b11000000,
                           0b10000000,
                           0b10010000};
BYTE CONST ARRIBA[8]    = {0b10000000,
                           0b11000000,
                           0b01000000,
                           0b01100000,
                           0b00100000,
                           0b00110000,
                           0b00010000,
                           0b10010000};
```

```
INT DERECHAR(FLOAT mto){
   MTO=MTO-1.0;
   WHILE (MTO>=0){
      IF(CONTDE<4){
         IF(BIT_TEST(INPUT_C(),7)){
               OUTPUT_B(0b00000000);
               OUTPUT_A(0b00000000);
               RETURN 1;
         }
            ELSE{
               OUTPUT_B(DERECHA[CONTDE]);
               DELAY_MS(TIMERx);
               CONTDE++;
            }
      }
      ELSE{
         CONTDE=0;
      }
      MTO=MTO-1.0;
   }
   OUTPUT_B(0b00000000);
   RETURN 0;
}
INT IZQUIERDAR(FLOAT mto){
   MTO=MTO-1.0;
   WHILE (MTO>=0){
     IF(CONTIZ<4){
      IF(BIT_TEST(INPUT_C(),6)){
            OUTPUT_B(0b00000000);
            OUTPUT_A(0b00000000);
            RETURN 1;
      }
         ELSE{
            OUTPUT_B(IZQUIERDA[CONTIZ]);
            DELAY_MS(TIMERx);
            CONTIZ++;
         }
     }
     ELSE{
      CONTIZ=0;
     }
     MTO=MTO-1.0;
   }
   OUTPUT_B(0b00000000);
   RETURN 0;
}
ARRIBAR(FLOAT mto){
   MTO=MTO-1.0;
   WHILE (MTO>=0){
     IF(CONTAR<8){
      OUTPUT_B(ARRIBA[CONTAR]);
      DELAY_MS(TIMERy);
      CONTAR++;
     }
```

```
      ELSE{
       CONTAR=0;
      }
      MTO=MTO-1.0;
   }
   OUTPUT_B(0b00000000);
}
ABAJOR(FLOAT mto){
   MTO=MTO-1.0;
   WHILE (MTO>=0){
     IF(CONTAB<8){
      OUTPUT_B(ABAJO[CONTAB]);
      DELAY_MS(TIMERy);
      CONTAB++;
     }
     ELSE{
      CONTAB=0;
     }
     MTO=MTO-1.0;
   }
   OUTPUT_B(0b00000000);
}
ARRIBATOOL(){
INT I=50;
   WHILE (I>0){
     IF(CONTTA<4){
      OUTPUT_A(IZQUIERDA[CONTTA]);
      DELAY_MS(TIMERx);
      CONTTA++;
     }
     ELSE{
      CONTTA=0;
     }
     I--;
   }
   OUTPUT_A(0b00000000);
}
ABAJOTOOL(){
INT I=50;
   WHILE (I>0){
     IF(CONTTB<4){
      OUTPUT_A(DERECHA[CONTTB]);
      DELAY_MS(TIMERx);
      CONTTB++;
     }
     ELSE{
      CONTTB=0;
     }
     I--;
   }
   OUTPUT_A(0b00000000);
}
// FUNCIONES PARA LA DIAGONAL
//FUNCION DE SIGNO PARA LA DIRECCIÓN DE LA PENDIENTE
```

```
FLOAT SGN(FLOAT X){
   IF(X>0) RETURN 1;
   IF(X<0) RETURN -1;
}
//FUNCIÓN PARA VERIFICACIÓN DEL FIN DE CARRERA
VERIFICA(INT DIR){
   IF(DIR==1) IZQUIERDAR(5000);
   IF(DIR==2) DERECHAR(5000);
}
//FUNCION PRINCIPAL DIAGONAL PASOX=0.0285 MILIMETROS PASOY=0.04847
MILIMETROS
 DIAGONAL(FLOAT DISTANCIA, FLOAT PENDIENTE){
            FLOAT TEMPX=0,TEMPY=0,S=0,I=1,DX1=0,DY1=0,DX2=0,DY2=0,TEMP=0;
            TEMPY=(DISTANCIA/PRECY)*SIN(PENDIENTE);
            TEMPX=(DISTANCIA/PRECX)*COS(PENDIENTE);
            DX1=SGN(TEMPX);
            DY1=SGN(TEMPY);
            DX2=SGN(TEMPX);
            DY2=0;
            IF(ABS(TEMPX)<=ABS(TEMPY)){
               DX2=0;
               DY2=SGN(TEMPY);
               TEMP=TEMPY;
               TEMPY=TEMPX;
               TEMPX=TEMP;
            }
            S=ABS(TEMPX)/2;
            WHILE(I<=ABS(TEMPX)){
               S=S+ABS(TEMPY);
               IF(S>=ABS(TEMPX)){
                  S=S-ABS(TEMPX);
                  IF(DX1>0) IF(DERECHAR(1)){
                                 VERIFICA(1);
                                 RETURN;
                              }
                  IF(DX1<0) IF(IZQUIERDAR(1)){
                                 VERIFICA(2);
                                 RETURN;
                              }
                  IF(DY1>0) {ARRIBAR(1);}
                  IF(DY1<0) {ABAJOR(1);}
               }
               ELSE{
                  IF(DX2>0) IF(DERECHAR(1)){
                                 VERIFICA(1);
                                 RETURN;
                              }
                  IF(DX2<0) IF(IZQUIERDAR(1)){
                                 VERIFICA(2);
                                 RETURN;
                              }
                  IF(DY2>0) {ARRIBAR(1);}
                  IF(DY2<0) {ABAJOR(1);}
               }
```

```
               I=I+1.0;
            }
 }
//FUNCION PRINCIPAL CIRCULO PASOX=0.03138 MILIMETROS PASOY=0.05847
MILIMETROS
CIRCULO(FLOAT RADIO, INT CONT){
   FLOAT ANGULO=0;
   FLOAT DIFX=0,DIFY=0,XT=0,YT=0;
   FLOAT X=0,Y=0;
   //INT CONT=0;
   IF(CONT==0){
      XT=ABS((RADIO/PRECX)*COS(ANGULO));
      YT=ABS((RADIO/PRECY)*SIN(ANGULO));
    //  ARRIBATOOL();
     // DERECHAR(XT);
      ANGULO=ANGULO+0.04;
    //  ABAJOTOOL();
      WHILE(ANGULO<=1.57){
         X=ABS((RADIO/PRECX)*COS(ANGULO));
         Y=ABS((RADIO/PRECY)*SIN(ANGULO));
         DIFX = ABS(XT-X);
         DIFY = ABS(YT-Y);
         WHILE((DIFX>0) || (DIFY>0)){
            IF(DIFY>0){
               ARRIBAR(1);
               DIFY=DIFY-1.0;
            }
            IF(DIFX>0){
               IF(IZQUIERDAR(1)){
                   VERIFICA(1);
                   RETURN;
               }
               DIFX=DIFX-1.0;
            }
         }
         ANGULO=ANGULO+0.04;
         XT=X;
         YT=Y;
      }
   }

   IF(CONT==1){
      XT=ABS((RADIO/PRECX)*SIN(ANGULO));
      YT=ABS((RADIO/PRECY)*COS(ANGULO));
     // ARRIBATOOL();
     // ARRIBAR(YT);
      ANGULO=ANGULO+0.04;
     // ABAJOTOOL();
      WHILE(ANGULO<=1.57){
         X=ABS((RADIO/PRECX)*SIN(ANGULO));
         Y=ABS((RADIO/PRECY)*COS(ANGULO));
         DIFX = ABS(XT-X);
         DIFY = ABS(YT-Y);
```

```
      WHILE((DIFX>0) || (DIFY>0)){
         IF(DIFX>0){
            IF(IZQUIERDAR(1)){
                VERIFICA(1);
                RETURN;
            }
            DIFX=DIFX-1.0;
         }
         IF(DIFY>0){
            ABAJOR(1);
            DIFY=DIFY-1.0;
         }
      }
      ANGULO=ANGULO+0.04;
      XT=X;
      YT=Y;
   }
}

IF(CONT==2){
   XT=ABS((RADIO/PRECX)*COS(ANGULO));
   YT=ABS((RADIO/PRECY)*SIN(ANGULO));
  // ARRIBATOOL();
  // IZQUIERDAR(XT);
   ANGULO=ANGULO+0.04;
  // ABAJOTOOL();
   WHILE(ANGULO<=1.57){
      X=ABS((RADIO/PRECX)*COS(ANGULO));
      Y=ABS((RADIO/PRECY)*SIN(ANGULO));
      DIFX = ABS(XT-X);
      DIFY = ABS(YT-Y);
      WHILE((DIFX>0) || (DIFY>0)){
         IF(DIFY>0){
            ABAJOR(1);
            DIFY=DIFY-1.0;
         }
         IF(DIFX>0){
            IF(DERECHAR(1)){
                VERIFICA(1);
                RETURN;
            }
            DIFX=DIFX-1.0;
         }
      }
      ANGULO=ANGULO+0.04;
      XT=X;
      YT=Y;
   }
}

IF(CONT==3){
   XT=ABS((RADIO/PRECX)*SIN(ANGULO));
   YT=ABS((RADIO/PRECY)*COS(ANGULO));
  // ARRIBATOOL();
```

```
      // ABAJOR(YT);
       ANGULO=ANGULO+0.04;
      // ABAJOTOOL();
       WHILE(ANGULO<=1.57){
          X=ABS((RADIO/PRECX)*SIN(ANGULO));
       Y=ABS((RADIO/PRECY)*COS(ANGULO));
       DIFX = ABS(XT-X);
       DIFY = ABS(YT-Y);
          WHILE((DIFX>0) || (DIFY>0)){
             IF(DIFX>0){
                IF(DERECHAR(1)){
                    VERIFICA(1);
                    RETURN;
                }
                DIFX=DIFX-1.0;
             }
             IF(DIFY>0){
                ARRIBAR(1);
                DIFY=DIFY-1.0;
             }
          }

          ANGULO=ANGULO+0.04;
          XT=X;
          YT=Y;
       }
    }

}

CIR(FLOAT RADIO){
   FLOAT ANGULO=0;
   FLOAT DIFX=0,DIFY=0,X=0,Y=0,XT=0,YT=0;
   INT CONT=0;
   XT=ABS((RADIO/PRECX)*COS(ANGULO));
   YT=ABS((RADIO/PRECY)*SIN(ANGULO));
   ARRIBATOOL();
   DERECHAR(XT);
   ANGULO=ANGULO+0.04;
   ABAJOTOOL();
    WHILE(CONT<4){
      WHILE(ANGULO<=1.57){
           IF(CONT==0){
               X=ABS((RADIO/PRECX)*COS(ANGULO));
               Y=ABS((RADIO/PRECY)*SIN(ANGULO));
               DIFX = ABS(XT-X)-1;
               DIFY = ABS(YT-Y)-1;
               WHILE((DIFX>=0) || (DIFY>=0)){
                  IF(DIFY>=0){
                     ARRIBAR(1);
                     DIFY=DIFY-1.0;
                  }
                  IF(DIFX>=0){
                     IF(IZQUIERDAR(1)){
```

```
                VERIFICA(2);
                RETURN;
            }
            DIFX=DIFX-1.0;
         }
      }
   }
IF(CONT==1){
   X=ABS((RADIO/PRECX)*SIN(ANGULO));
   Y=ABS((RADIO/PRECY)*COS(ANGULO));
   DIFX = ABS(XT-X)-1;
   DIFY = ABS(YT-Y)-1;
      WHILE((DIFX>=0) || (DIFY>=0)){
         IF(DIFX>=0){
            IF(IZQUIERDAR(1)){
                VERIFICA(2);
                RETURN;
            }
            DIFX=DIFX-1.0;
         }
         IF(DIFY>=0){
            ABAJOR(1);
            DIFY=DIFY-1.0;
         }
      }
}

IF(CONT==2){
   X=ABS((RADIO/PRECX)*COS(ANGULO));
   Y=ABS((RADIO/PRECY)*SIN(ANGULO));
   DIFX = ABS(XT-X)-1;
   DIFY = ABS(YT-Y)-1;
      WHILE((DIFX>=0) || (DIFY>=0)){
         IF(DIFY>=0){
            ABAJOR(1);
            DIFY=DIFY-1.0;
         }
         IF(DIFX>=0){
            IF(DERECHAR(1)){
                VERIFICA(1);
                RETURN;
            }
            DIFX=DIFX-1.0;
         }
      }
   }

IF(CONT==3){
   X=ABS((RADIO/PRECX)*SIN(ANGULO));
   Y=ABS((RADIO/PRECY)*COS(ANGULO));
   DIFX = ABS(XT-X)-1;
   DIFY = ABS(YT-Y)-1;
      WHILE((DIFX>=0) || (DIFY>=0)){
         IF(DIFX>=0){
```

```
                    IF(DERECHAR(1)){
                        VERIFICA(1);
                        RETURN;
                    }
                    DIFX=DIFX-1.0;
                }
                IF(DIFY>=0){
                    ARRIBAR(1);
                    DIFY=DIFY-1.0;
                }
            }
        }

        XT=X;
        YT=Y;
        ANGULO=ANGULO+0.04;
      }
      ANGULO=0;
      CONT++;
    }
}

//USB
void usb_debug_task(void) {
   static int8 last_connected;
   static int8 last_enumerated;
   int8 new_connected;
   int8 new_enumerated;
   new_connected=usb_attached();
   new_enumerated=usb_enumerated();
   last_connected=new_connected;
   last_enumerated=new_enumerated;
}
void usb2(void) {
   int8 in_data[8],SALIDA[8];
   float d=0,an=0,cuad=0;
   if (usb_kbhit(1)) {
            usb_get_packet(1, in_data, 8);
            SALIDA[0]=in_data[0];
            SALIDA[1]=in_data[1];
            SALIDA[2]=in_data[2];
            SALIDA[3]=in_data[3];
            SALIDA[4]=in_data[4];
            if(in_data[6]==1){
               d=(in_data[0]*100.0)+in_data[1];
               an=in_data[2]+(in_data[3]*0.1)+(in_data[4]*0.01);
               diagonal(d,an);
               SALIDA[7]=8;
               usb_put_packet(1, salida, 8, USB_DTS_TOGGLE);
            }
            if(in_data[6]==2){
               d=(in_data[0]*100.0)+in_data[1];
               cir(d);
               SALIDA[7]=10;
```

```
            usb_put_packet(1, salida, 8, USB_DTS_TOGGLE);
         }
         if(in_data[6]==3){
            d=(in_data[0]*100.0)+in_data[1];
            cuad=in_data[2];
            circulo(d,cuad);
            SALIDA[7]=9;
            usb_put_packet(1, salida, 8, USB_DTS_TOGGLE);
         }
   }
}
void main(void) {
   SETUP_ADC_PORTS(NO_ANALOGS);
   SETUP_ADC(ADC_OFF);
   SET_TRIS_B(0);
   SET_TRIS_A(0);
   SET_TRIS_C(0b11000011);
   OUTPUT_A(0b00000000);
   OUTPUT_B(0b00000000);
   while (1) {
      usb_task();
      usb_debug_task();
      if (usb_enumerated()) {
         usb2();
      }
   }
}
```