UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE CIENCIAS

TESIS

PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS APLICADAS

IMPLEMENTACIÓN DE UN SISTEMA OPTO-MECÁNICO PARA FABRICACIÓN DE MASCARILLAS

PRESENTA:
OSCAR MUÑOZ CRUZ

ASESORES:
DR. SALVADOR GUEL SANDOVAL
M. C. OSCAR FERNANDO NÚÑEZ OLVERA

SAN LUIS POTOSÍ, S. L. P. AGOSTO 2006

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE CIENCIAS

OSCAR MUÑOZ CRUZ

SINODALES:

DRA. AMPARO RODRÍGUEZ COBOS

DR. GUSTAVO RAMÍREZ FLORES

DR. SALVADOR GUEL SANDOVAL

M. C. OSCAR FERNANDO NÚÑEZ OLVERA

SAN LUIS POTOSÍ, S. L. P. AGOSTO 2006

AGRADECIMIENTOS

Antes que nada quiero agradecer a Dios, que es el que permite que sucedan las cosas, a mi esposa Laura por su comprensión y apoyo para cumplir este proyecto, a mi familia (mamá, abuelita y hermanos) por su apoyo incondicional en todas las metas que me he trazado.

Al Consejo Nacional De Ciencia y Tecnología (CONACYT) por el apoyo económico brindado.

A mis asesores Dr. Salvador Guel Sandoval y M.C. Oscar Fernando Núñez Olvera por su gran apoyo para concluir la maestría y por sus conocimientos impartidos.

A cada uno de mis profesores del Instituto, los cuales contribuyeron a la formación integral de mi persona.

INDICE

i.- INTRODUCCIÓN

CAPITULO I.-DISEÑO DEL SOFTWARE PARA GENERACIÓN DE MASCARILLAS

CAPITULO II.- DISEÑO MECÁNICO DE LA MESA X-Y

CAPITULO III.-TARJETA DE CONTROL DE MOTORES

i.- INTRODUCCIÓN.

i. 1.- ANTECEDENTES:

La tecnología de la electrónica del estado sólido se está transformando a grandes saltos en las últimas décadas, las oportunidades son notables en las nuevas industrias que han sido incubadas por los dispositivos electrónicos y ópticos.

Las técnicas aplicadas a la fabricación de dispositivos semiconductores están siendo continuamente revisadas, modificadas y mejoradas. En años recientes, se ha hecho énfasis principalmente en aumentar la tasa de rendimiento, expandir los niveles de automatización y aumentar los niveles de densidad, la cantidad de pasos en el proceso de fabricación se ha incrementado dos o tres veces, y cada proceso es más sofisticado.

En los primeros días el fabricante de IC diseñaba, construía y mantenía el equipo empleado en el ciclo de producción. Sin embargo, hoy día han aparecido nuevas industrias que han asumido la responsabilidad de introducir los últimos avances tecnológicos en el equipo de proceso. El equipo disponible de las compañías periféricas tiene un precio muy alto (no es raro que el costo de las unidades sea mayor a un millón de dólares) y la operación a 24 horas es casi una necesidad para asegurar una buena política económica.

La automatización está llegando a ser cada vez más importante en el ciclo de producción.

Una gran cantidad de controles basados en microprocesador, introducidos en forma de "direccionamiento por casete", ha reducido significativamente la posibilidad de errores debidos a transferencia incorrecta de información a la unidad de procesamiento. También tiene una sensibilidad al proceso que se está desarrollando que no está disponible por medio de la curva de respuesta humana.

Para un control de proceso mayor y un mejor rendimiento, la mayor parte de la fabricación ha pasado a operaciones de computadora sin papel, con terminales junto al equipo de procesamiento o hasta con el equipo conectado directamente a la computadora anfitrión. El mayor nivel de automatización también reduce la cantidad de "manejo" y contacto con la oblea, reduciendo, por tanto, la cantidad de fuentes contaminantes y aumentando el factor de producción. [1]

i. 2.- DEFINICIÓN DE MASCARILLA.

Una mascarilla o mascara es un patrón establecido de acuerdo al diseño del circuito, esta mascarilla es cortada en una capa de laminado plástico dimensionalmente estable llamado RUBYLITH, que consiste en una base transparente de milar cubierta con una sobrecapa opaca separable de rubí.

El dibujo final, o máscara, de cada capa de circuitos es creada por un ploter, o generador de patrones, controlado por ordenador.

Estos dibujos obtenidos con el ploter se reducen después al tamaño real del circuito, una máscara maestra practicada sobre vidrio con relieve en cromo, y luego se reproducen en una placa de trabajo que sirve para la impresión por contacto o proyección sobre la oblea.

Estas máscaras delimitan el modelo de las zonas conductoras y aisladoras que se transfieren a la oblea mediante fotolitografía.

Los CI más sencillos utilizan 6 máscaras o más para completar procesos de modelado, pero lo común es que se empleen de 10 a 24 máscaras.

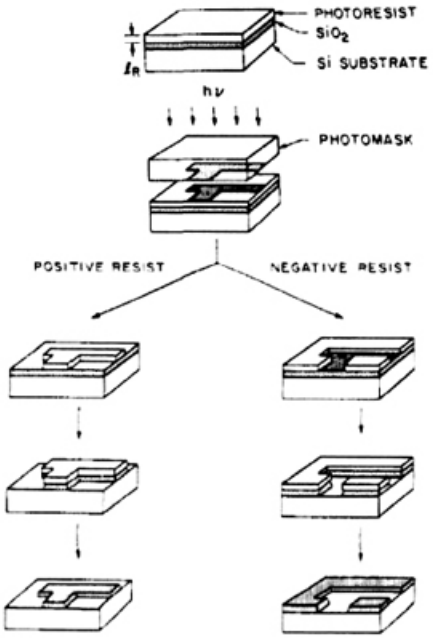
i. 3.- LITOGRAFÍA.

Una de las etapas de fabricación de circuitos integrados es la de impresión de los circuitos eléctricos que permitirán que el semiconductor pueda operar de acuerdo a los requerimientos de diseño. Esta impresión se realiza sobre la oblea de semiconductor por medio de un proceso de litografía.

A grandes rasgos, el proceso de litografía consiste en lo siguiente:

a).-Sobre la oblea de silicio, primeramente se deposita la foto resina.
b).-Después se coloca encima, la foto-máscara, la cual tiene la función de dejar pasar la luz UV solo en ciertas regiones.
c).- Finalmente se hace incidir luz UV para grabar los motivos (circuitos) de la máscara en la resina.
d).-Después mediante químicos se forma sobre la oblea de silicio el patrón requerido para que el semiconductor pueda realizar la función para la que fue diseñado.

i.4.- Proceso de Litografía y Grabado.



(a) Se cubre la oblea con una fotoresina + o -

Se hace incidir luz U.V. a través de una mascara

(b)

Se ablanda (+) o se endurece (-) la resina expuesta

(c) Se elimina la fotoresina no polimerizada con tricloroetileno

(d) Grabado: se ataca con HCl o HF y se elimina el SiO_2 no protegido por la fotoresina

(e) Se elimina la fotoresina con un disolvente Sulfúrico SO_4H_2

i.5.- OBJETIVO DE TESIS.

Por lo anteriormente mencionado el IICO (Instituto de Investigación en Comunicación Óptica), que cuenta con áreas de investigación referente a los dispositivos semiconductores, se ve en la necesidad de implementar un sistema automático para el diseño y generación de mascarillas, las cuales serán utilizadas en el proceso de litografía y grabado de las obleas. El presente trabajo de tesis consistió en implementar un sistema opto-mecánico para la generación de mascarillas, en donde las partes fundamentales del sistema son:

- Software para el diseño de la mascarilla
- Mesa x-y para el trazado de la mascarilla
- Tarjeta de Control de motores de la mesa x-y

CAPITULO I.- DISEÑO DEL SOFTWARE PARA GENERACIÓN DE MASCARILLAS.

1.1.-FUNCIONAMIENTO GENERAL.- El algoritmo se centra fundamentalmente en la creación de ciertas figuras geométricas, empleando programación grafica en C++, las cuales servirán para realizar el diseño de las mascarillas.

El software cuenta con un menú principal el cual tiene las siguientes opciones:

- Editar Mascarilla
- Recuperar Mascarilla
- Imprimir Mascarilla
- Salir

Para realizar el diseño de una mascarilla se elige la primera opción, inmediatamente después aparece una pantalla de edición, misma que cuenta con diferentes puntos a elegir:

- Iniciar figura
- Terminar figura
- Ayuda
- Paso
- Borra última línea
- Borra figura
- Guardar mascarilla
- Salir

Para continuar el proceso de diseño, seleccionamos el punto uno Iniciar Figura, abriéndose una ventana donde elegiremos el tipo de figura y el color de

contorno de esta, podemos elegir diferentes tipos de figuras dependiendo del diseño de la mascarilla que queramos realizar.

Al terminar un diseño se puede guardar en disco y posteriormente se puede recuperar con la posibilidad de hacer modificaciones. El software cuenta con algunas otras opciones que serán explicadas a detalle más adelante.

1. 2.- LENGUAJE DE PROGRAMACIÓN UTILIZADO.

1. 2. 1.- Historia del lenguaje C.

El lenguaje de programación C se diseñó para aumentar la economía de expresión en una amplia variedad de aplicaciones. El lenguaje C se usa principalmente para programación de sistemas, que fue como se usó para producir el 90% del código del sistema operativo UNIX. Los operadores, tipos de datos y librerías son muy ricos y transportables, lo que hace que sus aplicaciones sean poderosas.

La historia de C va paralela a la de UNIX, en 1969 los laboratorios Bell buscaron una alternativa al sistema operativo Multics para la computadora PDP-7. Se escribió una versión original del sistema operativo UNIX en lenguaje ensamblador. Al mismo tiempo, Kenneth Thompson estaba desarrollando un lenguaje experimental llamado B, posterior a b se escribió el primer lenguaje de programación de sistemas BCPL. En 1972 se diseño el lenguaje C como una extensión del B. La diferencia esencialmente radicaba en que C contenía una extensa condición de tipos estándares, mientras que B era un lenguaje sin tipos. El creador de C fue Dennis M. Ritchie.

Aunque C fue diseñado inicialmente como un lenguaje de programación de sistemas, ha ganado popularidad en diversas áreas de aplicación. Ha sido utilizado en aplicaciones numéricas de procesamiento de texto y de base de datos. Su transportabilidad y amplia librería de funciones permite la realización de diversas aplicaciones.

1. 2. 2.- Estructura de un programa en C

1.- Declaración de archivos de biblioteca.- son archivos que contienen funciones ya implementadas por el compilador para uso del programador. Por ejemplo:

#include<stdio.h>
Que contiene la definición de funciones como: printf, scanf, fwrite, etc. O
#include<math.h>
Que contiene funciones como: sqrt, sin, tan, log, etc.

Además el programador puede implementar sus funciones y encapsularlas en un archivo. Este archivo se declara casi igual, con la diferencia de que el nombre no se encierra entre paréntesis angulares <nombre.h>, sino entre comillas dobles "nombre.h" para denotar que fue implementada por el programador.

2.- Declaración de constantes simbólicas y macros.- Cuando se tienen datos que no cambiaran durante la corrida del programa es conveniente definirlos en esta sección. También cuando se manejan limites en algunas estructuras de datos (arreglos) y se tiene una dependencia directa con ellos en diferentes partes del programa, es necesario no tener "números mágicos". La definición de constantes simbólicas se hace de la siguiente manera:

#define FALSO 0
#define RUTA "c:\\"

Otra utilidad de esta declaración son las macros. Las cuales son bloques de programa nombrados por medio de una etiqueta (un nombre generalmente en mayúsculas) al cual procede dicho bloque, tomando en cuenta que si es más de una línea, al final de cada línea se agrega una diagonal invertida, menos en la última.
#define MIN(a<b)?a:b

```
#define MAX 10
#define CICLO for(i=0;i<MAX;i++)\printf("%d\n",i);
```

3.- Declaración de funciones prototipo.- Estas funciones son los encabezados de la definición de las funciones. Todas las funciones del programador deben ser declaradas aquí, tomando en cuenta que terminan con punto y coma (;)

```
int suma (int a, int b);
void imprimir(int resultado);
```

Es importante entender la diferencia de una declaración y una definición de funciones. Con la primera se le dice al compilador que se usará una función, mostrando su nombre (que indica que hace), tipo de dato que regresa y sus parámetros formales; en la segunda, se muestra la implementación, que y como lo hace.

4.- Declaración de tipos.- Esta sección se utiliza para declarar tipos definidos por el programador. Generalmente, tipos compuestos como arreglos, registros, archivos, apuntadores, etc. Una razón mas para la declaración de tipos, es el de no usar un nombre de tipo demasiado extenso o para darle un nombre más significativo. Ejemplo de esto pude ser:

```
typedef int bolean;
typedef struct nodo *ptrnodo;
```

5.- Función main.- Esta es la función principal del programa, la cual, debe denotar lo que el programa hace. Cuando se corre un programa, en esta función se inicia y termina la ejecución de las sentencias.

```
void main()
{
...
...
}
```

6.- Definición de funciones.- Cada función debe contener:

a).- Cabecera.- Nombre, argumentos o parámetros y valor que regresa.

b).- Declaraciones.- Tipos de datos a utilizar.

c).- Sentencias.- Expresiones o proposiciones [5]

1. 2. 3.- PROGRAMACIÓN GRÁFICA

1. 2. 3. 1.- Conceptos Fundamentales o Antecedentes.

Sistema visual u ocular. Las imágenes que percibimos son debidas a nuestro sistema visual: ojo, cerebro, y nervio óptico, principalmente. El ojo, que forma parte de este sistema visual, se caracteriza por sus componentes oculares. El iris controla la cantidad de luz que es permitida entrar al ojo.

La lente ocular concentra la luz permitida en el ojo para formar una imagen. La proyección de la imagen es situada en la retina - una estructura de dos dimensiones. La retina se compone principalmente de conos y bastoncillos, que son células especiales.

Estos conos y bastoncillos tienen la función de recibir estímulos de luz para la visión diurna y nocturna, respectivamente. La luz es descompuesta en las intensidades que corresponden a los colores rojo, verde, y azul.

Este trío de colores se denomina los colores primarios. A partir de los colores primarios se consiguen otros tres colores: cian, amarillo, y magenta, llamados los colores complementarios. Cian, amarillo, y magenta son complementarios a los colores rojo, verde, y azul, respectivamente, ya que combinados formarán el color blanco.

Resolución gráfica. En un sistema gráfico, se representan colores para formar una imagen. Los colores son combinados para formar tonos y otros colores en un área pequeña.

Esta área es llamada "píxel", que viene del inglés: *picture element* o elemento pictográfico (o de imagen). Este elemento o píxel es la parte fundamental más pequeña que contiene información para crear una imagen en un sistema gráfico.

Un conjunto de píxeles forma una malla para crear una imagen completa. La mayoría de los sistemas gráficos (por no decir todos) usa una malla rectangular con las mismas dimensiones de una superficie rectangular: anchura y altura. La anchura es el número de columnas y la altura, el número de filas de tal superficie gráfica.

El número de píxeles en la anchura y en la altura constituyen la resolución (gráfica) de la imagen. Por ejemplo, 640 x 480 significa que la resolución gráfica de la imagen constituye 640 columnas y 480 filas de píxeles formando una malla de 307.200 píxeles. Analizando otro ejemplo: 800 x 600, comprobamos que obtenemos una imagen de 480.000 píxeles.

Esta imagen contiene muchos más píxeles y por tanto mayor resolución pictográfica. Con una mayor resolución gráfica se obtiene una mejor calidad de imagen.

Cuando se habla de mejor resolución gráfica, se habla de usar un número mayor de líneas de píxeles en la horizontal y vertical. Realmente, lo que está pasando es que cuanto más se junten los píxeles, es más difícil distinguir entre uno y otro píxel vecino.

Modelos de colores. En sistemas gráficos, los colores son descompuestos y representados por combinaciones de valores numéricos.

Generalmente, se habla de un color diferente para cada combinación de valores. En realidad, se está hablando de tonos diferentes de colores, pero en el argot informático cada tono es tratado como un color diferente.

Por ejemplo, si se habla de un sistema gráfico de 15 bits por píxel (bpp) se dispondrá de 32.768 colores, aunque en realidad sean unos cuantos colores y varios miles de tonos. De igual forma, podemos manipular 24 bpp obteniendo un conjunto de 16.777.216 colores.

Existen varios modelos de descomposición de colores, especialmente usados en tecnología. Los modelos más populares y usados son RGB/A, CMYK, HLS, e YUV:

RGB. Estas siglas provienen del inglés *red*, *green*, y *blue*; esto es, *rojo*, *verde*, y *azul*, respectivamente. Este trío de colores intenta modelar el sistema visual humano. La mayoría de los monitores y televisores se basan en este modelo, debido a que están basados en la emisión de luz.

Este modelo también se le atribuye la propiedad de *sistema aditivo*, ya que se añaden las intensidades para formar un color.

RGBA. Se trata del mismo modelo RGB, pero con otra propiedad: canal *alfa*. Este canal se usa como un índice de la transparencia en un píxel. Esto nos sirve a la hora de mezclar varios colores designados para un solo píxel.

Este modelo es más reciente y se suele usar para crear efectos y técnicas visuales como "anti-aliasing", niebla, llamas, y objetos semi-transparentes: cristal, agua, vidrieras, etcétera.

CMYK. Las siglas representan, en inglés, *cyan*, *magenta*, *yellow*, y *black*; esto es, *cian*, *magenta*, *amarillo*, y *negro*, respectivamente. Los tres primeros colores son complementarios a los de RGB. Este modelo se usa más en la imprenta para crear imágenes a partir de tintas de colores.

Para obtener un color más vivo, se le añade tinta negra también. Este modelo tiene la propiedad de *sistema sustractivo*, ya que se restan las intensidades a la luz. El pigmento se obtiene porque el material absorbe la energía de la luz y por tanto una parte de su espectro. La energía que transmite se percibe como un color determinado. Por ejemplo, nosotros percibimos el color verde,

porque tal pigmento absorbe todos los "colores" del espectro de la luz emitida y transmite/refleja el color verde.

HLS. En inglés, las siglas son *hue*, *lightness* o *luminance*, y *saturation*; esto viene a ser, *matiz*, *brillo*, y *saturación*, respectivamente. Este modelo se basa en lo empírico de nuestra percepción visual. En lugar de usar un modelo tricolor para formar otros colores y tonos, el modelo HLS se basa en tres propiedades que sirven para definir los colores que percibimos. El matiz es el color que solemos denominar: azul, violeta, rojo, dorado, etcétera. El brillo describe la vividez y brillo de un tono de color. La saturación diferencia la percepción de un tono "puro" a otro tono que ha sido mezclado con blanco para formar un tono pastel - suave. Este modelo es usado por artistas, principalmente.

YUV también escrito **YCbCr** o incluso **YPbPr**. Este modelo se basa en el modelo RGB, pero restringiendo y descomponiendo algunos valores del RGB. YUV se usa en señales de televisión y en equipos de vídeo y grabación como S-Vídeo, y MPEG-2 y por tanto DVD. Originalmente, la principal razón de usar este modelo fue para mantener compatibilidad con los televisores analógicos en blanco y negro, cuando se introdujeron los televisores en color.

El componente Y es el brillo total. U y V son componentes cromáticos (colores) basados en los componentes rojo y azul del modelo RGB. El proceso comienza con una imagen en blanco y negro (Y). Luego, se obtiene el componente U, mediante una resta entre Y y el componente azul del RGB original, y V mediante una resta entre Y y el componente rojo.

Modelar. Sobra decir que la herramienta para crear gráficos es al fin y al cabo nuestro ordenador (o computadora). Esto implica que se debe trabajar con valores numéricos y por consiguiente habrá que realizar cálculos para conseguir una imagen.

Los valores numéricos que aquí se usan representan vértices, líneas, vectores, vórtices (vértices en tres dimensiones), y demás conceptos matemáticos,

al igual que la manipulación de bits para colores, combinación de imágenes, etcétera.

Siguiendo la misma lógica de la representación gráfica, debemos representar un objeto usando estos conceptos matemáticos. Este concepto de modelado se basa en la composición de objetos primitivos como vértices y líneas. Por ejemplo, Si queremos representar un cuadrado, tendremos que guardar información acerca de los vértices y líneas para poder recrear la imagen de un cuadrado.

Se dibuja una línea desde un vértice a otro representando un lado de este cuadrado. Luego, se dibujará una segunda línea desde el último vértice a uno nuevo; y así sucesivamente hasta dibujar los cuatro lados del cuadrado.

Se acaba de definir, en términos de objetos primitivos, el modelo de un cuadrado. Del mismo modo, podemos definir las características de un cubo: un conjunto de 8 vértices, 4 aristas, y 6 caras o planos. Con esta información, es posible crear una imagen desde cualquier punto de vista: de lado, desde arriba, abajo, etcétera.

También es posible girar el objeto, trasladarlo, cambiarlo de tamaño, y deformarlo, con tan sólo aplicar fórmulas a la información definida. Al tratar con modelos de objetos de dos o tres dimensiones (o más). Esta es la aproximación a un nivel más matemático, y por tanto, con ayuda del ordenador será posible realizar los cálculos necesarios.

Mapear. Ya que se está usando conjuntos de píxeles para representar imágenes, a veces es necesario que la imagen represente dimensiones más "humanas". Digamos que queremos crear una imagen de un mapa.

Se requiere que este mapa nos dé información precisa. Por ello, cada línea en el mapa debe representar una longitud determinada; por ejemplo, cada tramo

de carretera equivale a 10 kilómetros. Si esto es cierto para todos los tramos de la imagen, entonces el mapa está a escala.

Análogamente, es posible tener varias imágenes pero de distintos tamaños. Se necesita combinar estas imágenes para que tengan la misma escala. Esto se denomina mapear, del inglés *mapping*, o cambiar de escala.

El cambio de escala también puede producirse debido al modelo matemático. Por ejemplo, si se requiere representar una ecuación matemática:

$y = 2x + 3$, podemos calcular los valores de y a partir de valores escogidos de x. Estas coordenadas se usarán para poder seleccionar un píxel que las represente. Sin embargo, no todos los valores van a ser posibles representar, según los valores escogidos. Realmente se está hablando de representar una imagen en dimensiones matemáticas en un sistema gráfico de píxeles. Se debe realizar un cambio de escala y seguramente de coordenadas.

El mapeo es necesario al tratar con objetos de diferentes dimensiones y orientaciones. Se debe transformar el estado de todos los objetos a una base para que sean compatibles. En el ejemplo de la ecuación matemática, se acabaría transformando las coordenadas cartesianas calculadas a coordenadas en píxeles.[7]

1.3.- ESTRUCTURA DEL SOFTWARE.

1.3.1.- Menú Principal:

Para iniciar el dibujo de una mascarilla, el usuario se encontrará primeramente con el Menú Principal, el cual como se menciono al inicio cuenta con las siguientes opciones:

- Editar Mascarilla
- Recuperar Mascarilla
- Imprimir Mascarilla
- Salir

Editar Mascarilla.- El usuario elegirá esta opción cuando quiera iniciar el Diseño de una mascarilla, abriéndose la ventana principal que contiene el área de trabajo (representada por una cuadricula de 10 x 10).

Recuperar Mascarilla.- Cuando ya se tenga el diseño de una mascarilla guardado en disco y se quiera realizar alguna modificación, se elegirá esta opción. Posteriormente se debe escribir el nombre del archivo con la extensión ".doc".

Imprimir Mascarilla.- Una vez que ya se tenga terminado el diseño de la mascarilla deseada, se mandara imprimir por hardware eligiendo esta opción.

Salir.- Nos permite salir del Menú Principal.

La Fig. 1.3.1.a nos muestra el diagrama de bloques del Menú Principal.

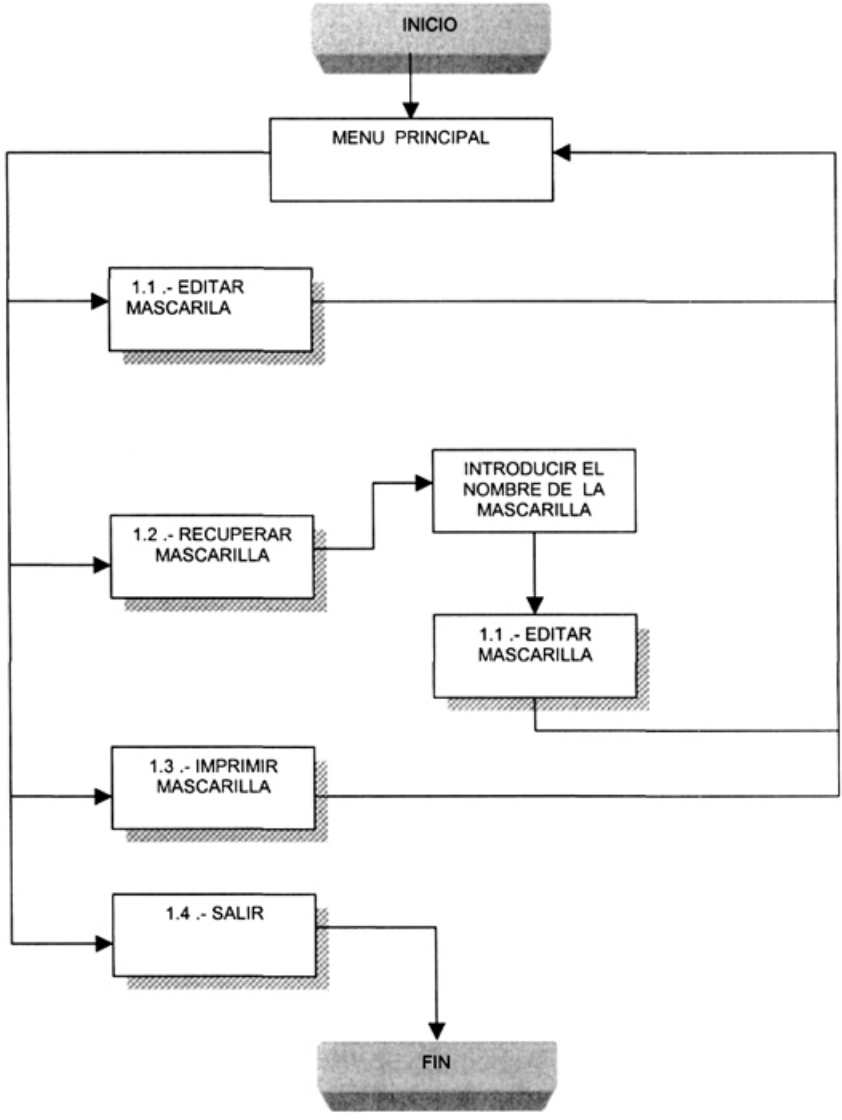


Fig. 1.3.1. a.- Diagrama de bloques del Menú Principal

1.3.2.- Menú Edición:

Al seleccionar en el Menú la opción Editar Mascarilla, nos aparecerá una ventana de edición, desplegando diferentes opciones a elegir:

- Iniciar Figura
- Terminar Figura
- Ayuda
- Paso
- Borra Última Línea
- Borra Figura
- Guardar Mascarilla
- Salir

Iniciar Figura.- Da Inicio al diseño de una mascarilla, en el cual se pude elegir cuatro tipos diferentes de figuras (polígono, arco, circulo y sector) y el color de contorno de cada figura.

Terminar Figura.- Esta opción une el punto final de la figura con el punto inicial, logrando que se concluya la edición de la figura.

Ayuda.- Proporciona Información sobre cada uno de los puntos señalados, además de mostrar las teclas auxiliares para la edición.

Paso.- Se refiere a incrementar en cierta cantidad la unidad de medida que se este utilizando. Consiste en cuantos pasos puede terminar una figura entre mayor sea el número de paso, más rápido se termina una figura.

Borra última línea.- En el caso de que alguna de las coordenadas no sea la necesaria para la figura en edición, esta opción permite borrar la última línea generada.

Borra figura.- Borra cada una de las figuras realizadas, en caso de existir algún error en el diseño.

Guardar mascarilla.- Terminando el diseño de una mascarilla se pude guardar en disco con la extensión ".doc", para poder recuperarla posteriormente y realizar los cambios pertinentes si así se requiere.

Salir.- Nos permite salir de la ventana de edición.

La Figura 1.3.2.a nos muestra el diagrama de bloques de Edición.

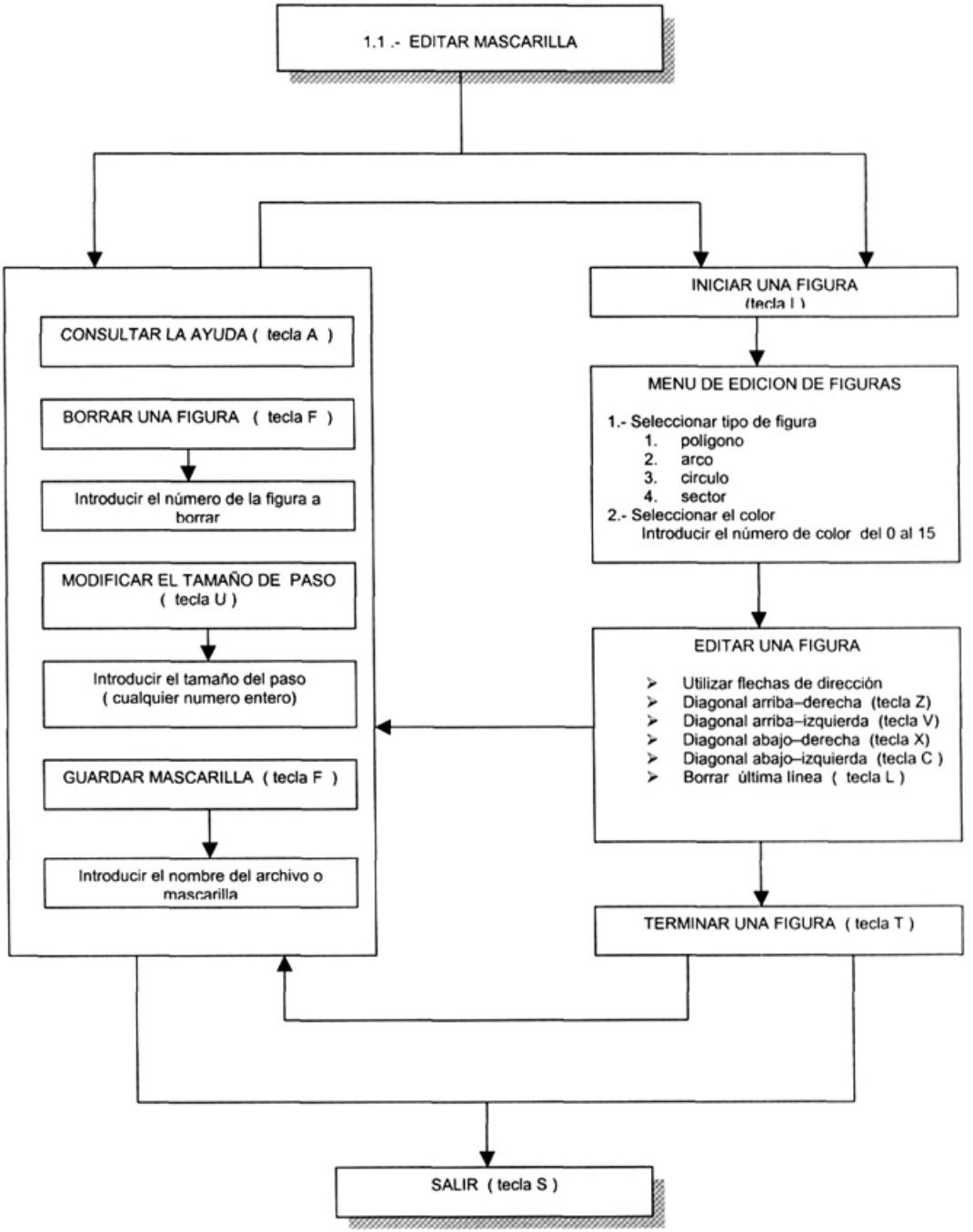


Fig. 1.3.2.a.- Diagrama de Bloques de edición

1.3.3.- Algoritmo para el modo de edición

El algoritmo esta basado fundamentalmente en identificar la acción que quiere realizar el usuario mediante la tecla pulsada, básicamente el algoritmo consiste en un ciclo infinito que permanece ejecutándose mientras el usuario no quiera salir (tecla s).

El editor puede estar en 2 modos: Modo dibujo y Modo normal (Pantalla)
En la figura 1.3.3.a se muestra el diagrama de bloques del Algoritmo.

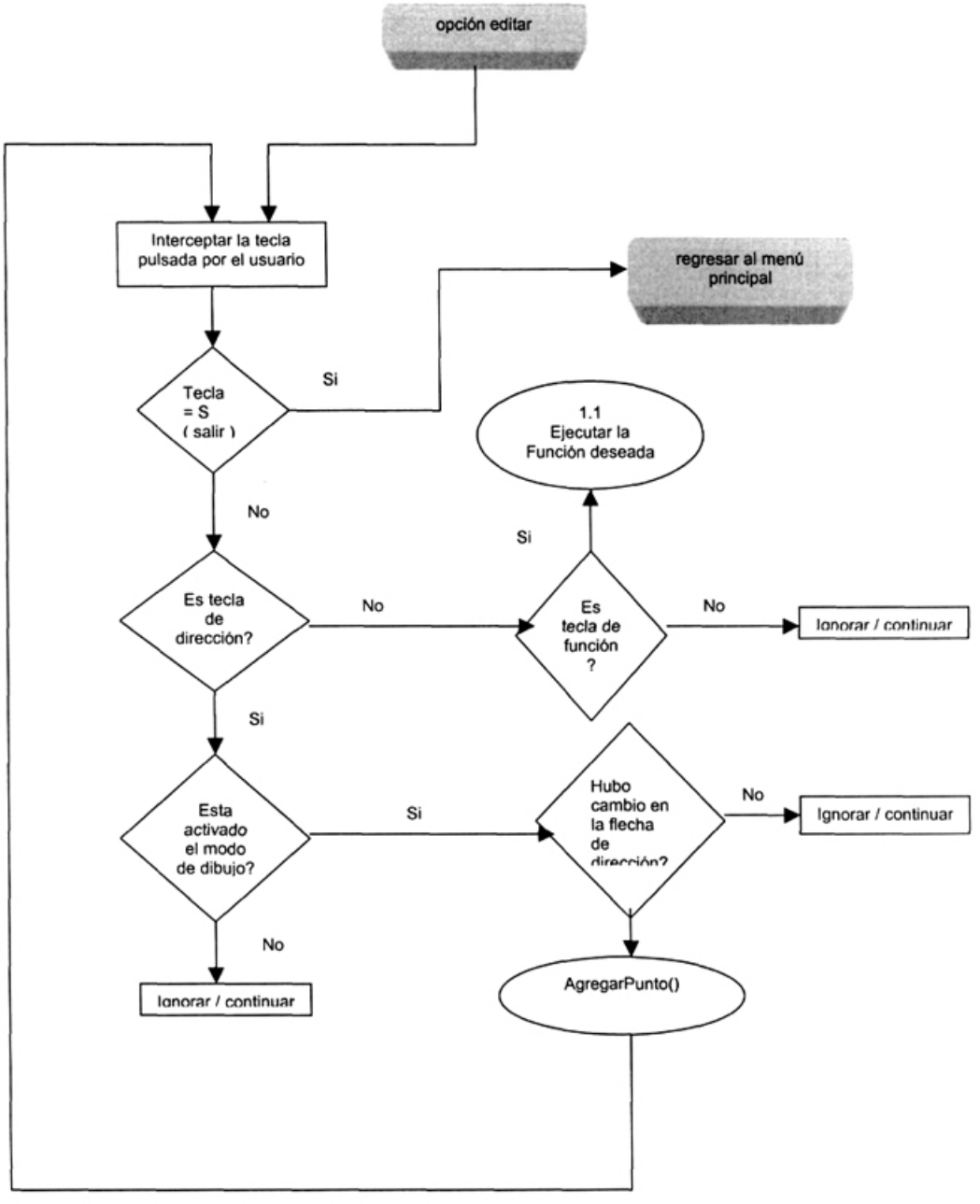


Fig. 1.3.3.a. –Diagrama de bloques del algoritmo para el modo de edición

1.3.4.- Diagrama de bloques para la función deseada

Este proceso se ejecuta en cualquiera de los 2 modos de ejecución.

Los rectángulos de la primera columna representan los nombres de las funciones principales que se encargan de ejecutar la acción adecuada y posteriormente se describen los procesos auxiliares a cada una de estas, generalmente son diálogos de interfaz que se utilizan para la comunicación gráfica con el usuario, con el fin de modificar o proporcionar los parámetros de funcionamiento de todo el programa o simplemente para obtener información.

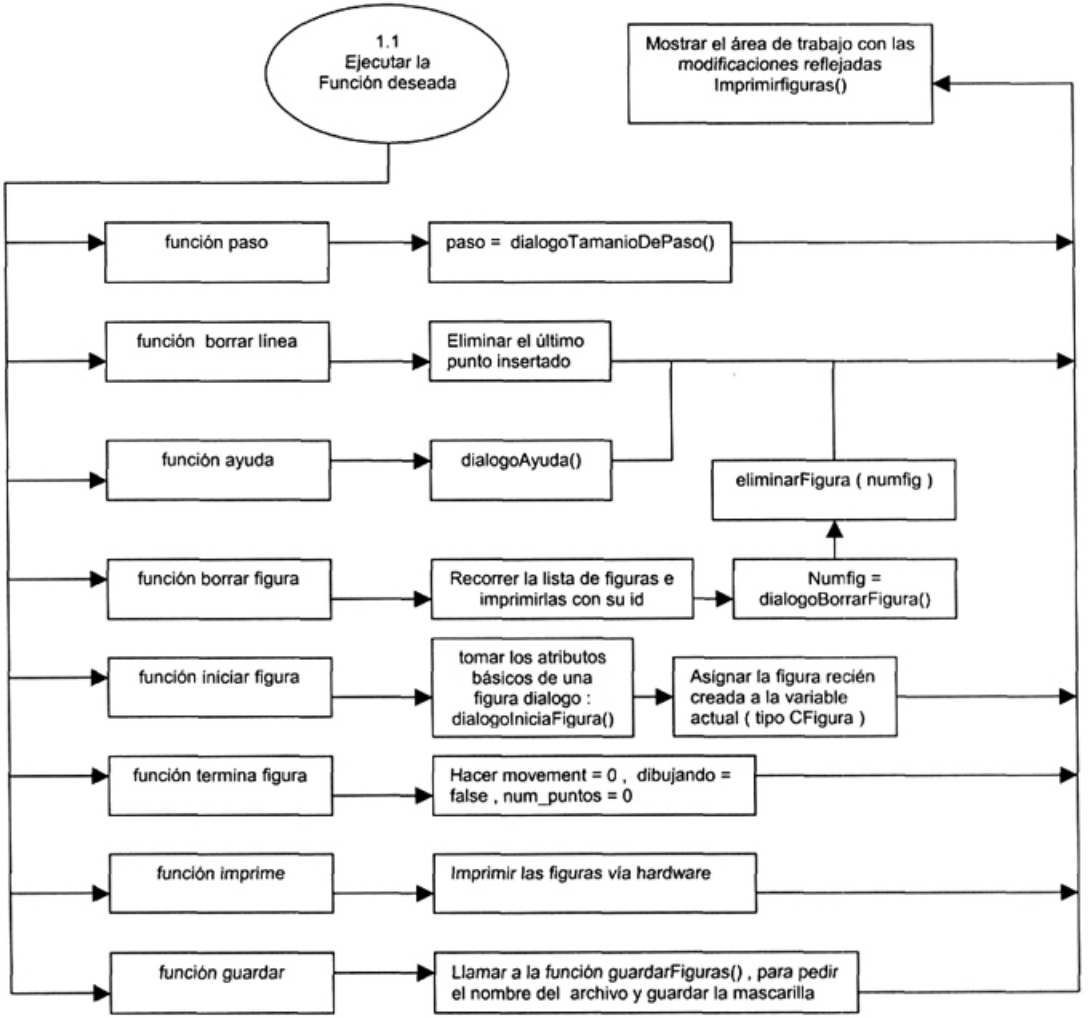


Fig. I.3.4.a.- Diagrama de bloques para la ejecución de la función deseada.

1.3.5.- ALGORITMOS Y FUNCIONES PARA LA CREACIÓN DE FIGURAS. [3]

En este apartado se muestra parte del algoritmo utilizado para la creación de las diferentes figuras, así como las funciones matemáticas básicas empleadas.

1 .- Algoritmo utilizado para dibujar un polígono

En el algoritmo implementado para la creación de cualquier tipo de polígono, se utiliza la función predefinida ***drawpoly***, la cual requiere como parámetros principales el número de puntos del polígono y el conjunto de coordenadas en donde se dibuja el polígono (array de puntos).

A continuación se muestra la implementación del algoritmo.

Variables:

cont : representa el numero de puntos del polígono.

poly : representa el arreglo de puntos alternando los valores x , y.

Algoritmo :

```
drawpoly(cont+1,poly);   //dibuja el polígono en pantalla
```

2 .- Algoritmo utilizado para dibujar un circulo

Para dibujar un circulo en pantalla, se utiliza la función predefinida ***circle,*** que requiere como parámetros el centro desde donde se inicia el radio del circulo (coordenadas x e y) y el radio, para calcular el radio del circulo se utiliza la ecuación fundamental de este, utilizando las funciones matemáticas de raíz cuadrada y potencia (***sqrt , pow***).

Se muestra el código implementado en el algoritmo.

Variables :

xc = centro en x.

yc = centro en y.

px = extremo en x.

py = extremo en y;

Algoritmo :

```
radio = (int) sqrt ( pow( (double) (px-xc) , 2) + pow ( (double) (py-
yc) , 2 ) );     //ecuación  del círculo
circle( xc , yc radio );          //dibuja el circulo con la función de
biblioteca
```

3 .- Algoritmo utilizado para dibujar un arco (la mitad de un círculo)

Este algoritmo utiliza principalmente la función ***arc*** que requiere como parámetros principales: El ángulo de inicio, las coordenadas x y y de origen, ángulo final y el radio del semicírculo, el arco se comienza a trazar desde el ángulo de inicio y se concluye en el ángulo final.

Se calculan otras variables que solamente son auxiliares para los cuatro principales parámetros mencionados anteriormente, como se trata de un semicírculo la diferencia en grados desde el inicio al final siempre es de 180^0.

Como se puede observar en el código, el algoritmo cuenta con 3 secciones diferentes (**DIAMETRO VERTICAL, DIAMETRO HORIZONTAL Y DIAMETRO INCLINADO**) cada sección fue implementada para poder dibujar el arco en diferentes orientaciones, así la sección DIAMETRO VERTICAL puede dibujar un arco vertical hacía la derecha o hacía la izquierda, de igual manera las otras dos secciones dibujan el arco de acuerdo a la orientación especificada.

Variables :

p1x : primer coordenada en x.
p1y: primer coordenada en y.
p2x: segunda coordenada en x.
p2y: segunda coordenada en y.
dx: distancia entre las dos coordenadas en x.
dy: distancia entre las dos coordenadas en y.
m: inclinación o pendiente del diametro inscrito en el arco.
centro_x : la mitad del diametro en x.
centro_y : la mitad del diametro en y.
stangle : ángulo de inicio para trazar el arco (start angle).

endangle : ángulo para finalizar el trazo del arco (end angle).

Algoritmo :

```
dx = p2x-p1x;
dy = p2y-p1y;
dy*=-1;          //CONVERTIR EJE 'Y' DE PANTALLA A EJE 'Y'
CARTESIANO

// m=32000 cuando es vertical simulando la cantidad infinita
m=32000; if(dx!=0){ m = (double)((double)dy/(double)dx);}; ///
CALCULAR LA PENDIENTE
radianes =  (atan(m));
grados = (int)(((double)(radianes*180))/pi); // DETERMINAR EL
ANGULO ///
diametro = (int) sqrt( pow((double)(dx),2) + pow
((double)(dy),2));//calcula el diametro
radio = (int) (diametro/2); //calcula el radio

//////////  DETERMINAR EL CENTRO EN X  Y CENTRO EN Y
////////////////
if(dx==0)  centro_x = p1x; else  centro_x = p1x + (dx/2);
if(dy==0)  centro_y = p1y; else centro_y = p1y - (dy/2);

if(m==32000) ////////  DIAMETRO VERTICAL //////////////
  {
    if(dy>=0) //abajo hacia arriba
      {
        stangle=90;   endangle=270;
      }
    else    // arriba hacia abajo
      {
        stangle= 270;         endangle=90;
```

```
    }
  }
  else
  if(m==0) ////////////  DIAMETRO HORIZONTAL ////////////////
   {
      if(dx>=0)  // izquierda a derecha
      {
        stangle=0;
        endangle=180;
      }
      else      // derecha a izquierda
      {
        stangle = 180;
        endangle =0;
      }
   }
  else
   {    ////////////////  DIAMETRO INCLINADO  ////////////////
      if(dx>=0)
       {
        stangle= grados;
        endangle=180+grados;
       }
      else
       {
        stangle = 180 + grados;
        endangle = grados;
       }
   }

  setcolor(color_frontera);
  arc(centro_x,centro_y, stangle, endangle,radio);  //llamada a la
función de biblioteca
```

4 .- Algoritmo utilizado para dibujar un sector (la cuarta parte de un circulo)

La implementación de el algoritmo es muy similar a la de la sección anterior, solamente que aquí para dibujar el sector, la diferencia en grados desde el ángulo inicio hasta el ángulo final es de 90^0.

También se utiliza la función predefinida ***arc***, además de contar también con tres secciones diferentes (**RADIO VERTICAL, RADIO HORIZONTAL Y RADIO INCLINADO)**, para de esta manera dar la orientación deseada al sector que se quiera dibujar.

Variables :

p1x : primer coordenada en x.
p1y: primer coordenada en y.
p2x: segunda coordenada en x.
p2y: segunda coordenada en y.
dx: distancia entre las dos coordenadas en x.
dy: distancia entre las dos coordenadas en y.
m: inclinación o pendiente del radio inscrito en el arco.
centro_x : la mitad del diametro en x.
centro_y : la mitad del diametro en y.
stangle : ángulo de inicio para trazar el arco (start angle).
endangle : ángulo para finalizar el trazo del arco (end angle).

Algoritmo :

```
dx = p2x-p1x;
dy = p2y-p1y;
dy*=-1;  // CONVERTIR EJE 'Y' DE PANTALLA A EJE 'Y'
CARTESIANO

//m=32000 cuando es vertical
m=32000; if(dx!=0){ m = (double)((double)dy/(double)dx);};  ///
CALCULAR LA PENDIENTE  ///////////////
```

```
  radianes =  (atan(m));
  grados = (int)(((double)(radianes*180))/pi); // DETERMINAR EL
ANGULO ///
  radio = (int) sqrt( pow((double)(dx),2) + pow
((double)(dy),2));//calcula el diametro

  //////////  DETERMINAR EL CENTRO EN X  Y CENTRO EN Y
///////////////
  centro_x = p2x;
  centro_y = p2y;

  if(m==32000)  ////////  RADIO VERTICAL ////////////////
  {
   if(dy>=0) //abajo hacia arriba
    {
         stangle=180;
         endangle=270;
       }
    else    // arriba hacia abajo
       {
        stangle= 0;
        endangle=90;
       }
   }
   else
   if(m==0)   ////////// RADIO HORIZONTAL ////////////////
    {
       if(dx>=0)  // izquierda a derecha
        {
          stangle=90;
          endangle=180;
        }
       else      // derecha a izquierda
```

```
        {
          stangle= 270;
          endangle=0;
        }
    }
  else
    {
       if(dx>=0)    ////////  RADIO INCLINADO   ////////////////////
        {
          stangle  = grados;
          endangle = 360 - grados;
        }
       else
        {
          stangle = 90 + grados;
          endangle = grados;
        }
    }


  setcolor(color_frontera);
  arc(centro_x,centro_y, stangle, endangle,radio);   // llamada a la
función de biblioteca
```

1.4.- INTERFAZ CON EL USUARIO.

1.4.1.-VENTANA DEL MENÚ PRINCIPAL.

Como se mencionó en los apartados anteriores el software cuenta con un menú principal, el cual proporciona cuatro opciones a elegir.

La pantalla es la que se muestra a continuación:

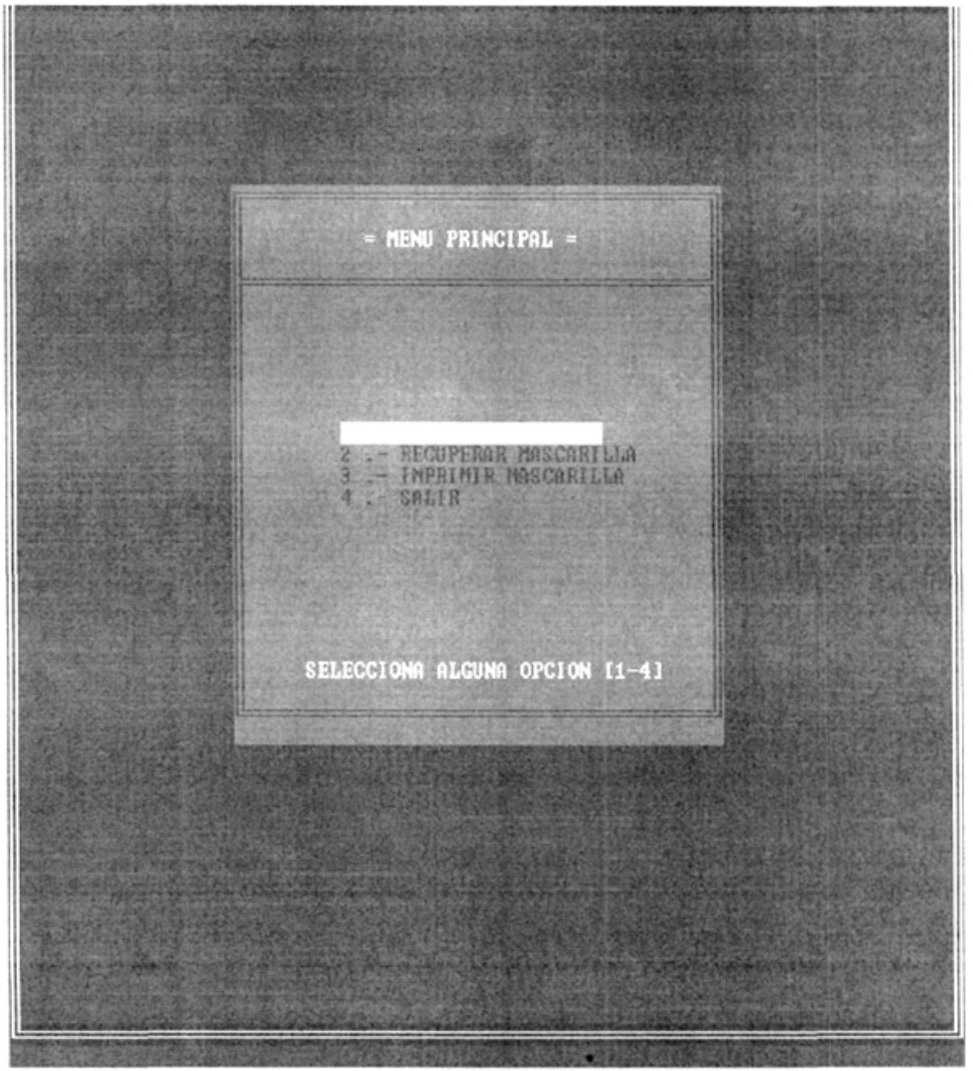


Fig. 1.4.1.a.- Ventana del Menú Principal

1.4.2.- PANTALLA DE DISEÑO.

Para iniciar un diseño de mascarilla el usuario elegirá editar mascarilla, la cual abrirá una pantalla de edición, esta pantalla como se ha venido mencionando cuenta con diferentes puntos a elegir, el diseño de la mascarilla se realizara sobre un área de trabajo cuadriculada de 10x10 (450 x 450 píxeles), también cuenta con un contador sobre las coordenada en el eje de las X y eje de las Y, el cual se localiza en el lado izquierdo inferior de la pantalla.

A continuación se presentan estas pantallas que son la Interfaz con el usuario.

Pantalla de Edición de Figuras.

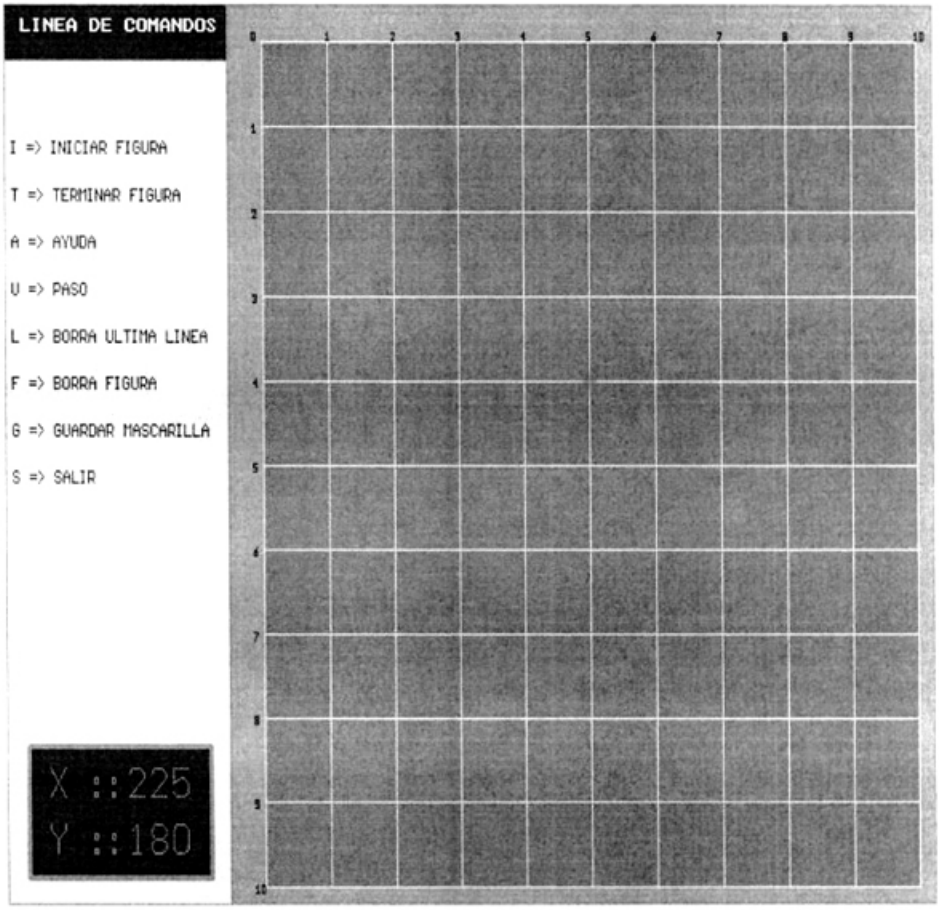


Fig. 1.4.2.a.- Pantalla de Diseño de Mascarillas

1.4.3.- MENÚ DE AYUDA

El software también cuenta con un Menú de Ayuda, mostrando las teclas que se utilizan para acceder a los comandos de uso general, así como a las de los comandos de Edición de Figura. Esta ventana se muestra a continuación en la siguiente figura.

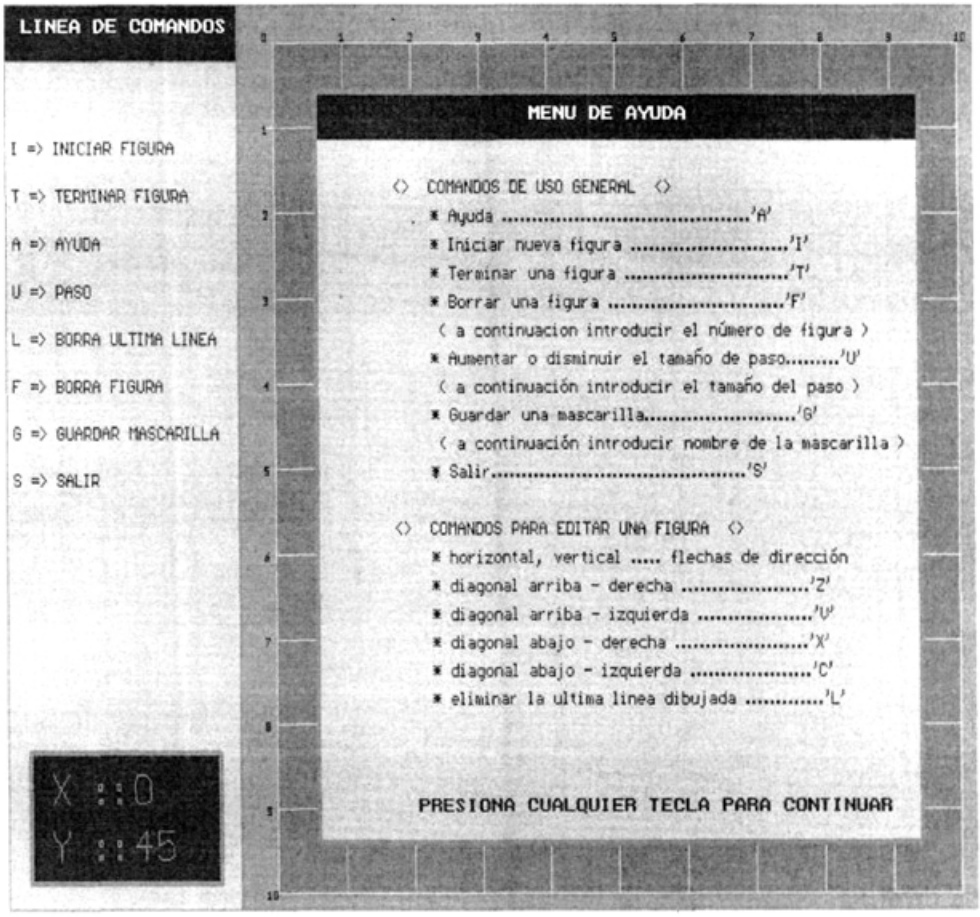


Fig. 1.4.3.a.-Ventana Menú de Ayuda

1.4.4.-VENTANA INICIAR FIGURA.

Siguiendo con el proceso para el diseño de una mascarilla, al seleccionar el usuario Iniciar Figura se abre una nueva ventana, en esta nos pide el tipo de figura que se quiere diseñar, se elige un número del 1 al 4 de acuerdo a la figura que deseemos, posteriormente nos pide elegir el color de contorno de la figura que elegimos, entre los cuales se pueden seleccionar 16 colores (del 0 al 15) , una vez realizado este procedimiento, se accede nuevamente a la pantalla de edición para la realización de la figura. Se muestra la ventana correspondiente.

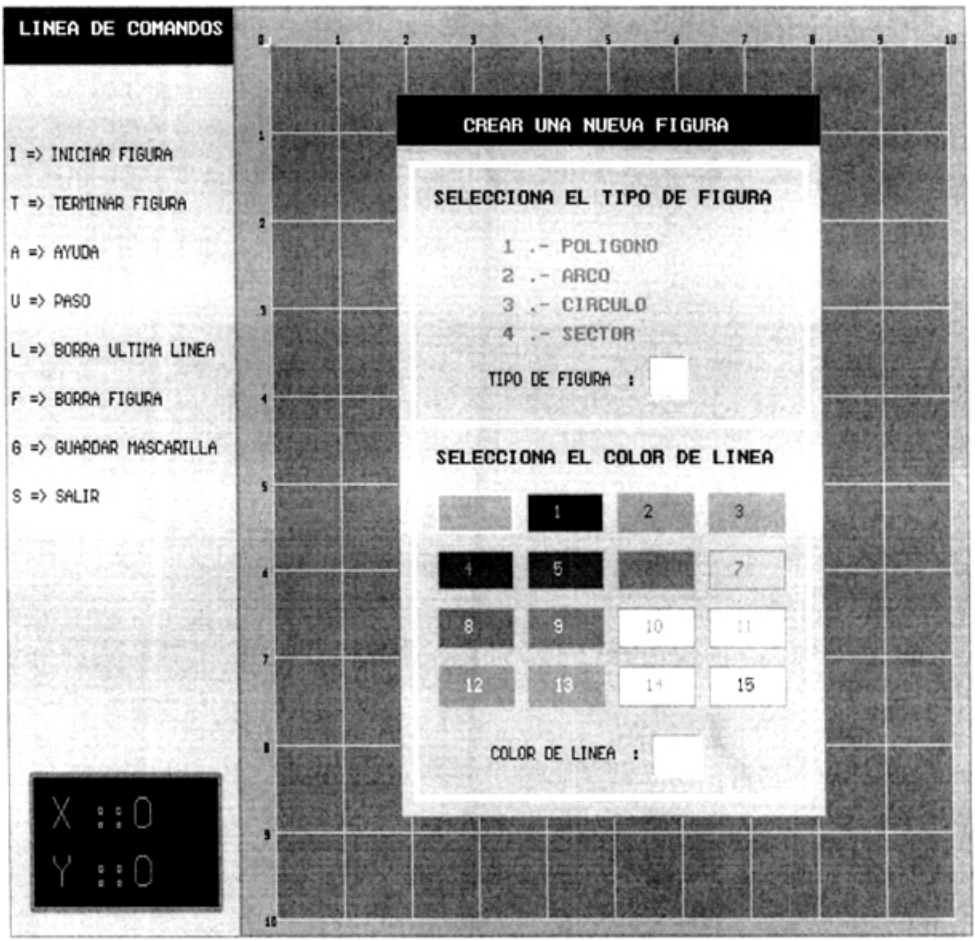


Fig. 1.4.4.a.- Ventana Iniciar Figura

CAPITULO II.- DISEÑO MECÁNICO DE LA MESA X-Y

2. 1.- Antecedentes:

Antes de iniciar el diseño de la mesa, se realizó un análisis de lo que se requería, en base al trazado del dibujo realizado en la PC, realizando este trazado sobre el material llamado rubylith.

Una vez definidos los aspectos técnicos de la mesa trazadora, se eligió el mejor material para la construcción de esta, siendo este material el aluminio.

En base a lo anterior, se realizó la cotización del material requerido con diferentes proveedores y una vez contando con el material, se procedió a manufacturar las piezas en el taller mecánico del IICO.

2. 2.- Características

De acuerdo al uso que se requiere, las características fundamentales de la mesa son las que se enumeran a continuación.

- Estructura en Aluminio
- Peso Aproximado 30 Kg.
- Área de Trabajo de 40 x 40 cm.
- Transmisión por Tornillo (espárrago)
- 2 Motores de CD para el movimiento en el plano X-Y

2. 3.- Componentes

2. 3. 1.- Motores de Corriente Directa

En el proyecto se utilizaron motores de corriente directa para el movimiento en los dos ejes. Estos motores trabajan con un voltaje máximo de 33 Volts de corriente continua.

Se eligieron motores de CD porque estos tienen mayor velocidad que los motores de pasos, además se les han acoplado encoders, con los cuales se cierra el lazo de control de movimiento, lo cual hace posible asegurar que la mesa llegará a la posición requerida.

En la figura 2.3.1. se pueden observar algunos motores de C.D.

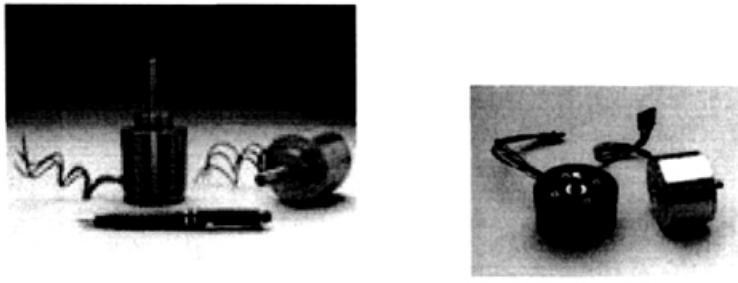


Figura 2.3.1. Motores de C.D

2.3.1.a.-Principio de Funcionamiento.

Un motor de corriente continua (CC) está compuesto de un estator y un rotor. En muchos motores, generalmente los más pequeños, el estator está compuesto de imanes para crear un campo magnético. En motores más grandes este campo magnético se logra con devanados de excitación de campo.

El rotor es el dispositivo que gira en el centro del motor y está compuesto de bobinas de alambre magneto por donde circula la corriente continua. Esta corriente continua es suministrada al rotor por medio de las "escobillas" generalmente fabricadas de carbón.

Cuando un conductor por el que fluye una corriente continua es colocado bajo la influencia de un campo magnético, se induce sobre él (el conductor) una fuerza que es perpendicular tanto a las líneas de campo magnético como al sentido del flujo de la corriente.

Haciendo circular una corriente por una espira situada en un campo magnético, cada conductor se verá sometido a una fuerza de direcciones contrarias, por serlo el sentido de la corriente.

El par de fuerzas generado hará girar la espira que, al disponer de un colector de delgas, hará que la corriente circule siempre en el mismo sentido manteniendo el sentido del par y por tanto del giro.

Si se invierte el sentido de la corriente cambiando la polaridad de la alimentación, se conseguirá cambiar el sentido de giro.

La fuerza con la que el motor gira (el par motor) es proporcional a la corriente que hay por los conductores. A mayor tensión, mayor corriente y mayor par motor.

Se muestra el principio de funcionamiento de los motores de C.D en la figura 2.3.1.a

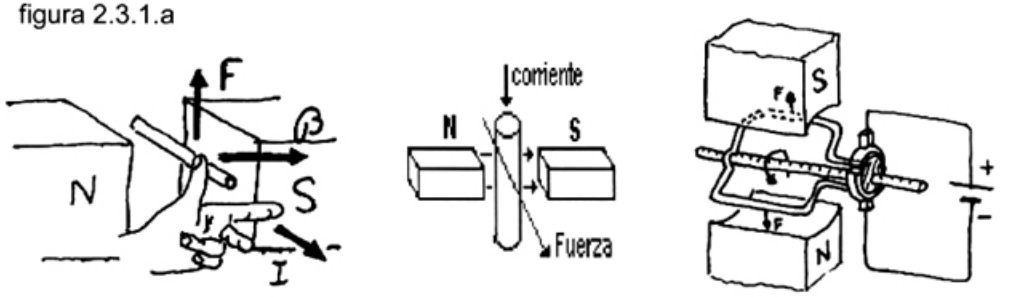


Figura 2.3.1a. Principio de funcionamiento

2.3.2.-Tornillos Espárragos.

Este es uno de los componentes principales de la mesa ya que son los que dan el movimiento a cada eje. En el caso del eje "X" así como del eje "Y", el largo de los espárragos es de 60.5 centímetros con un grosor de 3/8 de pulgada por 18 hilos por pulgada.

Los espárragos van unidos al eje del motor; que al girar mueven la mesa.

2.3.3.- Baleros lineales.

Para el movimiento uniforme en el eje X y eje Y sobre las barras de acero, se requirió de este tipo especial de baleros, ya que estos transmiten un movimiento lineal al proporcionarles un movimiento rotacional, es decir que al momento del giro de los motores estos baleros se desplazaran a lo largo de las barras de los ejes X y Y.

En la mesa se usaron 4 baleros lineales para cada eje. Los baleros son los súper 8 de la marca Thompson, estos tienen un diámetro interno de 1/2 pulgada, con diámetro exterior de 0.875 pulgadas y un largo de 1.25 pulgadas.

Estos baleros tienen un deslizamiento sobre las barras paralelas de cada eje. Dos de estos están colocados dentro de un porta balero.

En la siguiente figura se muestra el balero lineal. [6]

Figura 2.3.3 balero lineal.

2.3.4.-Barras de acero.

Las barras de acero de alta dureza, tienen un diámetro de 1/2 pulgada por 60 centímetros de largo, utilizadas para los ejes "XY".

La figura 2.3.4 muestra una sección de la mesa en la sección "Y"; en la cual se aprecian las barras con los baleros lineales.

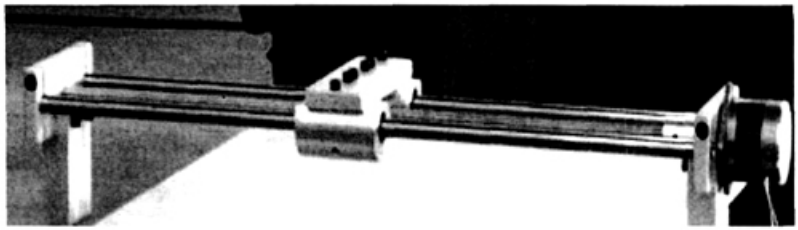


Fig. 2.3.4 Barras de Acero

2.3.5.-Encoders.

Cada motor tiene acoplado un Encoder de la marca Microtech, modelo MES-20-200P; el cual tiene una resolución de 200 pulsos por revolución.

El Encoder utilizado es que se muestra en la siguiente figura el cual tiene un eje de ¼ de diámetro.

Fig. 2.3.5 Encoders

2.4.-Funcionamiento del Sistema.

Un mecanismo es una estructura o un complejo ordenado de las partes de una máquina, adaptada para producir un efecto. Es decir, un mecanismo es un sistema físico que nos permite a través de él realizar un trabajo útil.

Para el caso de la mesa, el trabajo que requiere hacer es el de corte de un material llamado rubylith para el diseño de mascarillas, utilizadas en el proceso foto-litográfico de dispositivos.

Para llevar a cabo este trabajo, es necesario contar con un mecanismo que permita mover la herramienta que va a cortar.

El mecanismo utilizado en este proyecto está compuesto por una base móvil de seis travesaños (eje X, Y) en el cuál cada uno tiene un motor de corriente directa acoplado, para dar el movimiento de la herramienta de corte a lo largo del plano X-Y.

Cada motor de CD es manejado por la tarjeta de control, la cual administra el voltaje y la corriente que pasa por los devanados del motor, consiguiendo así un movimiento preciso y coordinado de la herramienta de corte.

Como se mencionó anteriormente, los motores de corriente directa tienen que mover a lo largo del plano X-Y la herramienta de corte, la cual a su vez realizará el trazado de la figura sobre el rubylith.

Para que el movimiento sea posible, cada motor cuenta con un mecanismo de tornillo esparrago.

Al girar el motor, el tornillo gira junto con él, solamente que el tornillo proporcionará un movimiento lineal a la base que sostiene la herramienta de corte, tal movimiento depende de la velocidad del motor en RPM y del número de hilos por pulgada del tornillo. En este caso tenemos un tornillo de 18 hilos/in.

A continuación se muestran algunas imágenes de la mesa x-y.

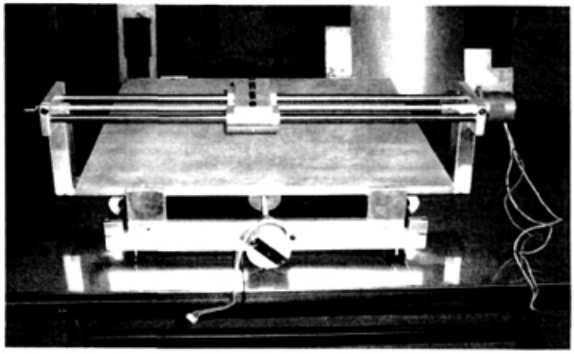


Fig. 2.4.1.-Vista de la mesa X-Y

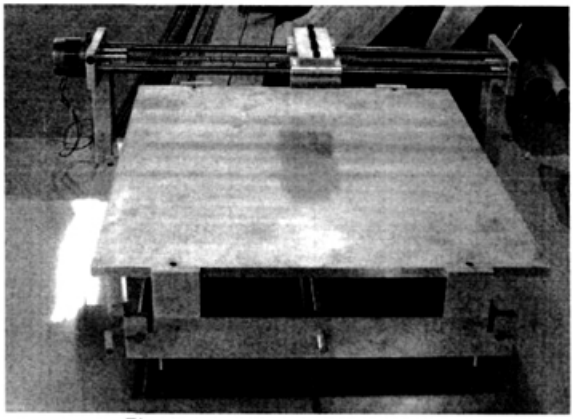


Fig. 2.4.2.- Vista Frontal de la mesa X-Y

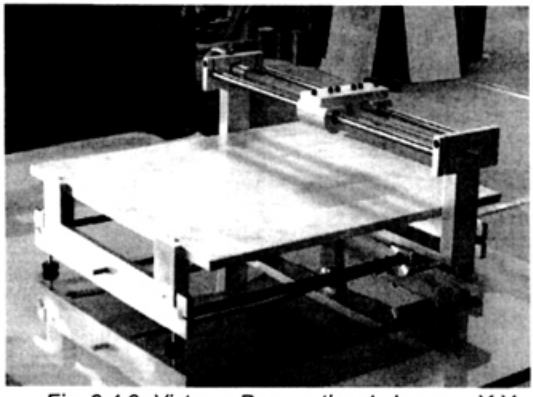


Fig. 2.4.3.-Vista en Perspectiva de la mesa X-Y

CAPITULO III.- TARJETA DE CONTROL DE MOTORES.

3.1.-Características.

Las características de esta tarjeta son las siguientes:

- Microcontrolador PIC16F877A
- 3 controladores LM629N
- 3 Puente H LMD18200
- Fuente de alimentación regulada de 5v
- Comunicación Serial, tipo RS-232
- Manejo de hasta de 3 motores de corriente directa.

3.2.-Componentes.

3.2.1.-Microcontrolador PIC16F877A

El cerebro de la tarjeta de control de motores es un microcontrolador, este microcontrolador envía a cada LM629N los parámetros de posición, velocidad y aceleración requeridos en el sistema.

El microcontrolador PIC16F877A de Microchip pertenece a una gran familia de microcontroladores de 8 bits (bus de datos) que tienen las siguientes características generales que los distinguen de otras familias:

- Arquitectura Harvard
- Tecnología RISC
- Tecnología CMOS

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

3.2.1.1.- Características generales del PIC16F877

La siguiente es una lista de las características que comparte el PIC16F877 con los dispositivos más cercanos de su familia:

PIC16F873	PIC16F874	PIC16F876	PIC16F877

Entre las principales características del PIC 16F877A se encuentran las siguientes:

- Set de 35 instrucciones.
- Todas las instrucciones se ejecutan en un ciclo de instrucción excepto los saltos que toman dos ciclos.
- Velocidad de operación de 20 MHz. Reloj de entrada.
- 200 ms. Ciclo de instrucción.
- Memoria del programa (Flash): 8Kb (14 bits).
- Memoria de datos: 368 x 8 bits.
- Memoria de datos (EEPROM): 256 x 8 bits.

3.2.1.2.- Diagrama de Bloques del PIC16F877A

En la siguiente figura se muestra a manera de bloques la organización interna del PIC16F877A, Se muestra también junto a este diagrama su diagrama de patitas, para tener una visión conjunta del interior y exterior del Chip.

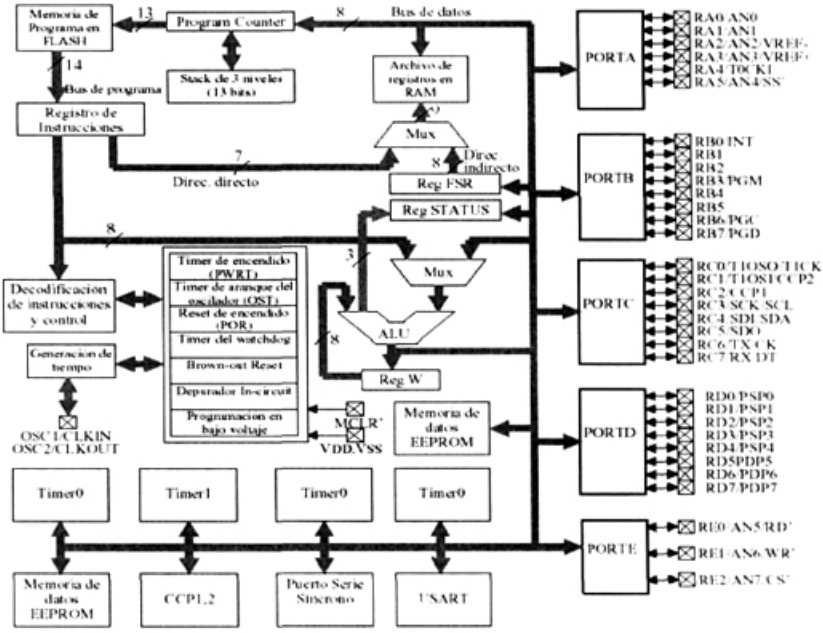


Fig. 3.2.1.2.- Diagrama de Bloques del PIC16F877A

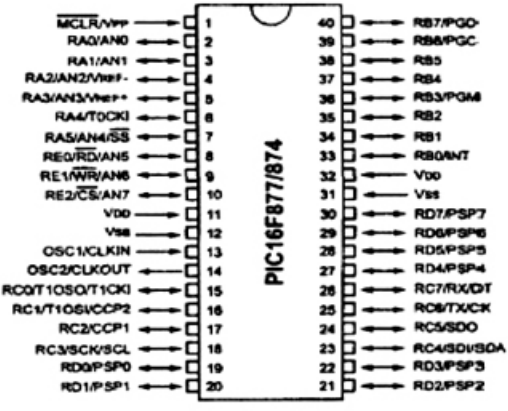


Fig. 3.2.1.3 Diagrama externo (de patitas) del PIC16F877A

3.2.2 LM629N

En la tarjeta que se diseño se cuenta con 3 LM629N, para nuestro proyecto de tesis solo se utilizaran 2 procesadores.

Este procesador es el que maneja el motor de corriente directa a través del puente H, en este proyecto de tesis como se utilizan dos motores de cd uno para cada eje, utilizamos dos procesadores LM629N.

El control del motor está dado en la programación del PIC 16F877A quien envía a cada LM629N los parámetros de posición, aceleración y velocidad. Estos parámetros son verificados con el encoder que está unido a cada motor.

Las principales características del LM629N son:

- Registros de posición, velocidad y aceleración de 32 bits.
- Filtro PID digital programable con coeficiente de 16 bits.
- Salida PWM de señal-magnitud de 8 bits.
- Generador interno de velocidad trapezoidal.

Algunas de las características que es importante señalar del LM629N es la del Generador interno de velocidad trapezoidal, llamado así porque tiene la peculiaridad de que al inicio de arranque del motor lo acelera hasta llegar a su velocidad máxima y la mantiene prácticamente constante hasta que se aproxima al punto deseado empezando a desacelerar, creando de esta manera un trapecio.

El LM629 recibe la señal del encoder que va acoplado al motor de cd por el decodificador (pin3-A, pin4-B), el cual envía esta señal a un punto de error, este compara los parámetros reales con los parámetros calculados, la salida de este punto de error es enviada internamente a un filtro digital PID (Proporcional, Integral, Derivativo) de 16 bits, para posteriormente enviar 2 señales (signo y magnitud) al Puente H (dirección y PWM) y de esta manera controlar el sentido de giro y velocidad del motor de corriente directa.

3.2.3.-Micro encoder MES-20-200P.

El encoder utilizado tiene una resolución de 200 pulsos por revolución. Este va acoplado al eje de cada motor y envía las señales de retroalimentación al LM629N.

Algunas de las principales características del micro encoder son las que se mencionan a continuación.

- 200 pulsos por vuelta.
- Fases de salida A, B y Z.
- Salida de onda cuadrada.
- Respuesta hasta 100KHz.
- Soporta 6000 revoluciones por minuto.

3.2.4.-Puente H LMD18200.

La etapa de potencia para cada motor se basa en un puente HLMD18200 de la marca National. Soporta hasta 55 Volts de entrada y puede proporcionar hasta 3 Amperes de corriente de salida. El LMD18200 recibe las señales de control (magnitud modulada en ancho de pulso y dirección) del microcontrolador LM629N y envía al motor la alimentación correspondiente para que este funcione de la manera requerida.

3.3 Comunicación con la PC

3.3.1.-Comunicación Serial

La comunicación con la PC, se realiza de forma serial, utilizando para ello el C.I Max 232 y el estandar RS232.

El puerto serial, también conocido por el estándar que lo norma, el RS-232, fue creado con el único propósito de contar con una interfaz entre los equipos terminales de datos (Data Terminal Equipment, DTE), y el equipo de comunicación de datos (Data Communications Equipment, DCE) empleando intercambio serial de datos binarios. De esta forma el equipo terminal de datos es el extremo cliente de los datos y el equipo de comunicación de datos es el dispositivo que se encarga de la unión entre los terminales, tal como un módem o algún otro dispositivo de comunicación.

El RS-232 fue originalmente adoptado en 1960 por la Asociación de Industrias de la Electrónica, conocida también por sus siglas en inglés EIA, Electronic Industries Association. El estándar evolucionó a través de los años y en 1969 la tercera revisión, el RS-232C, fue el estándar elegido por los fabricantes de computadoras personales compatibles con IBM. En 1987 se adoptó la cuarta revisión, el RS-232D, también conocida como EIA-232D. En esta nueva revisión se agregaron 3 líneas de prueba.

El estándar RS-232 original especifica una velocidad máxima de 19,200 baudios y una longitud máxima de cable en 50 pies, aproximadamente 16 metros, lo cual resultaba conveniente para la época; sin embargo el paso del tiempo y la evolución de la tecnología obligaron el aumento de estos parámetros, emergiendo el RS422 y el RS485, que utilizan líneas balanceadas para eliminar algunos problemas que se presentan a mayores velocidades de transmisión.

La mayoría de los equipos que implementan puertos RS-232 utilizan un conector DB-25 aún cuando la documentación original del estándar no especifica un conector en especial, la mayoría de las computadoras comenzaron a utilizar el conector DB-9 dado que 9 son los conectores que se requieren para la

comunicación asíncrona. Es necesario notar que el documento especifica la cantidad de postes o terminales y su asignación, 20 para las señales, 3 reservados y 2 sin uso. Normalmente el conector macho es en el lado de la terminal y el conector hembra es en el de comunicaciones, aún si este no es el caso común.

La característica especial del RS-232, y que lo hiciera popular en el mundo de las computadoras es su diseño simple, en el cual los datos viajan como voltajes referidos a una tierra común, haciendo factible que pueda ser utilizado para vínculos síncronos como SDLC, HDLC, Frame Relay y X.25, además de la transmisión síncrona de datos.

3.4.-Tarjeta de Control de Motores.

Antes de mostrar el esquemático y la imagen de la tarjeta de control, se presenta el diagrama de flujo (Fig. 3.4.1), que nos muestra en forma general la relación que tiene cada elemento entre sí.

Una vez que se ha realizado el diseño de la mascarilla en la computadora, el archivo generado (coordenadas de la figura) será enviado al PIC16F877A por modo serial, El PIC16F877A enviara por el puerto B los comandos y datos a cualquiera de los LM629N, los cuales a su vez controlaran la velocidad y dirección de los motores de CD a través del Puente H, la retroalimentación a los LM629N se realiza por medio de los encoders que van acoplados a cada uno de los motores, los sensores de fin de carrera paran los motores cuando estos llegan al limite de cada uno de los ejes (X y Y)

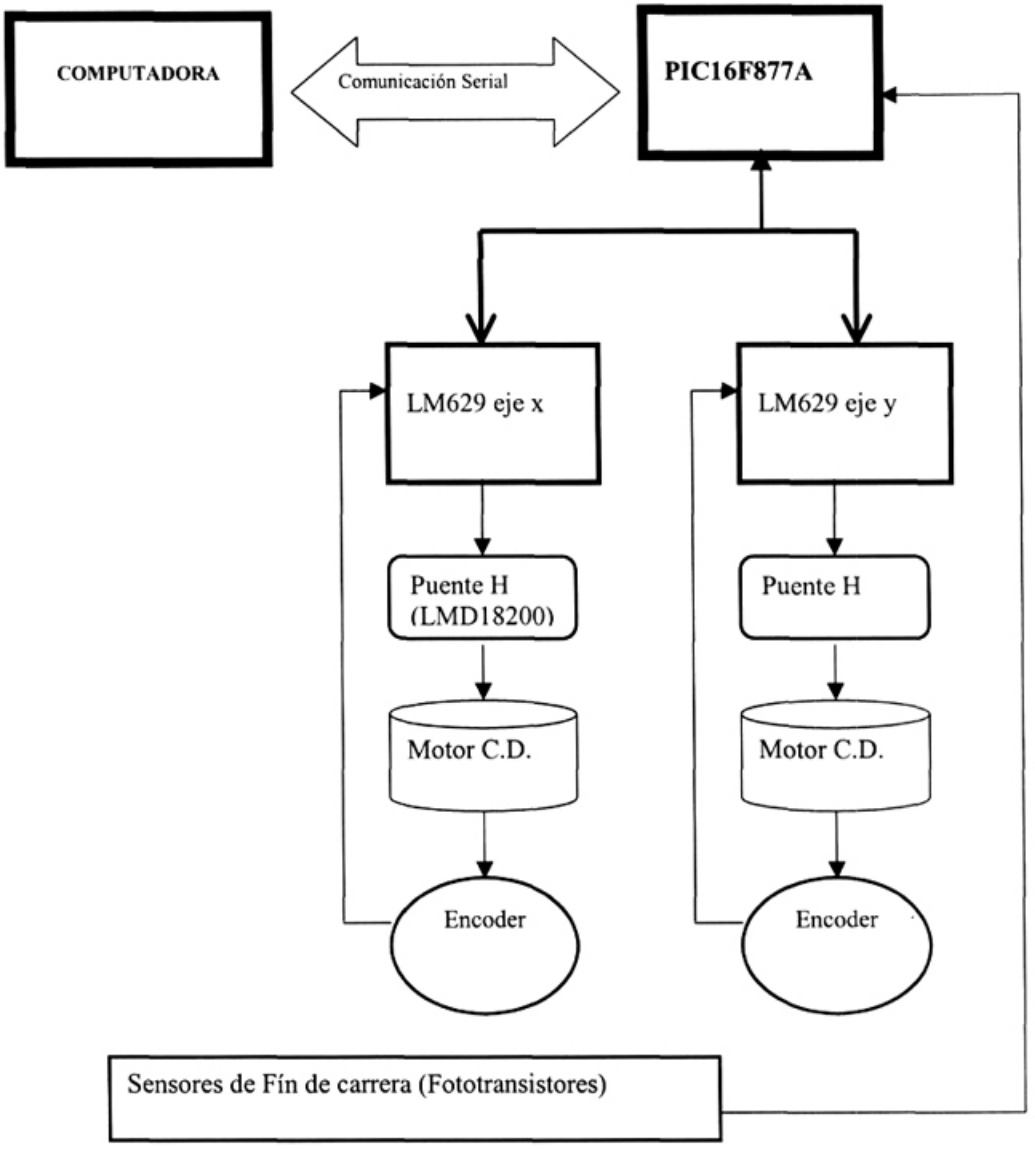


Fig. 3.4.1.- Diagrama de Flujo de la tarjeta de Control

A continuación se muestra el diagrama esquemático generado en el software de diseño de circuitos impresos, Eagle versión 4.15.

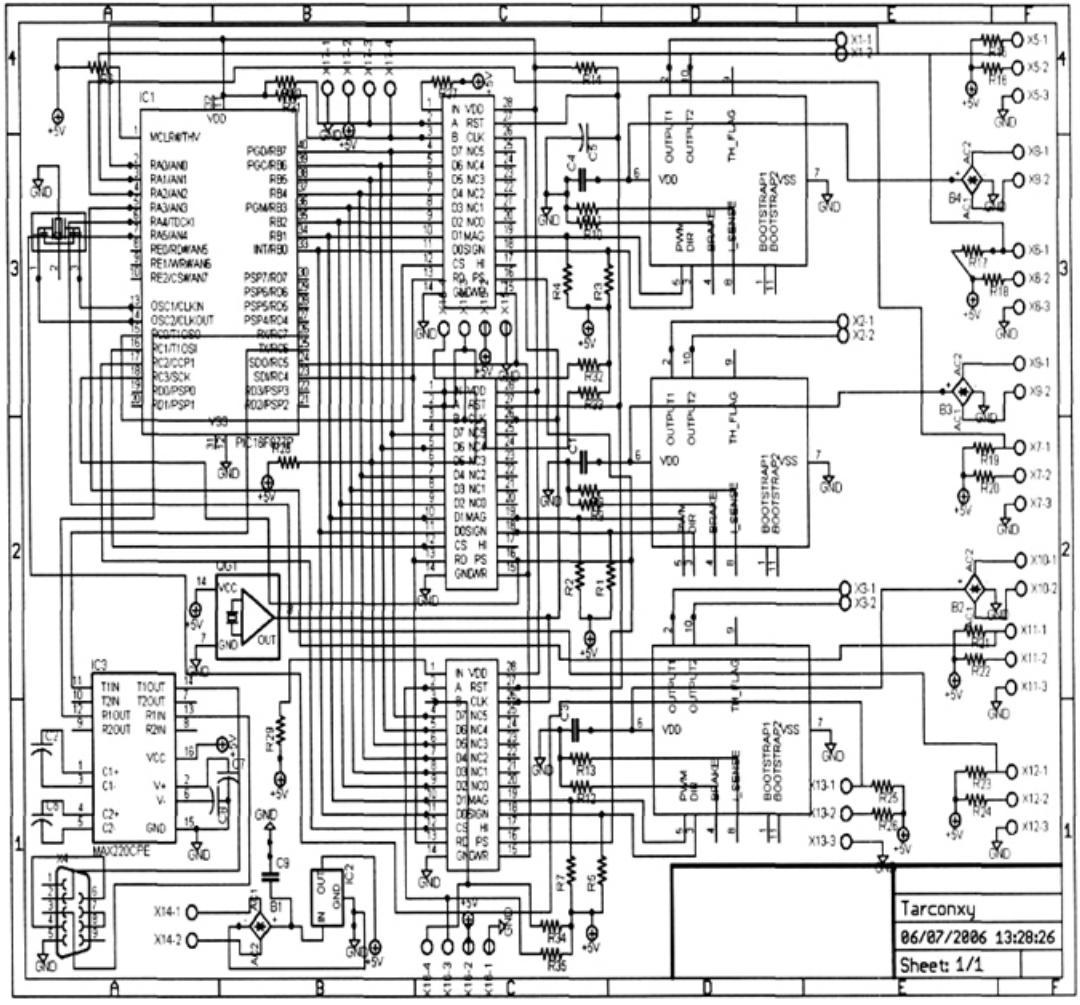


Fig. 3.4.2- Esquemático de la tarjeta de control

También se muestra una imagen, tomada de la tarjeta ya implementada, una vez realizado todo el proceso de fabricación en los laboratorios del Instituto.

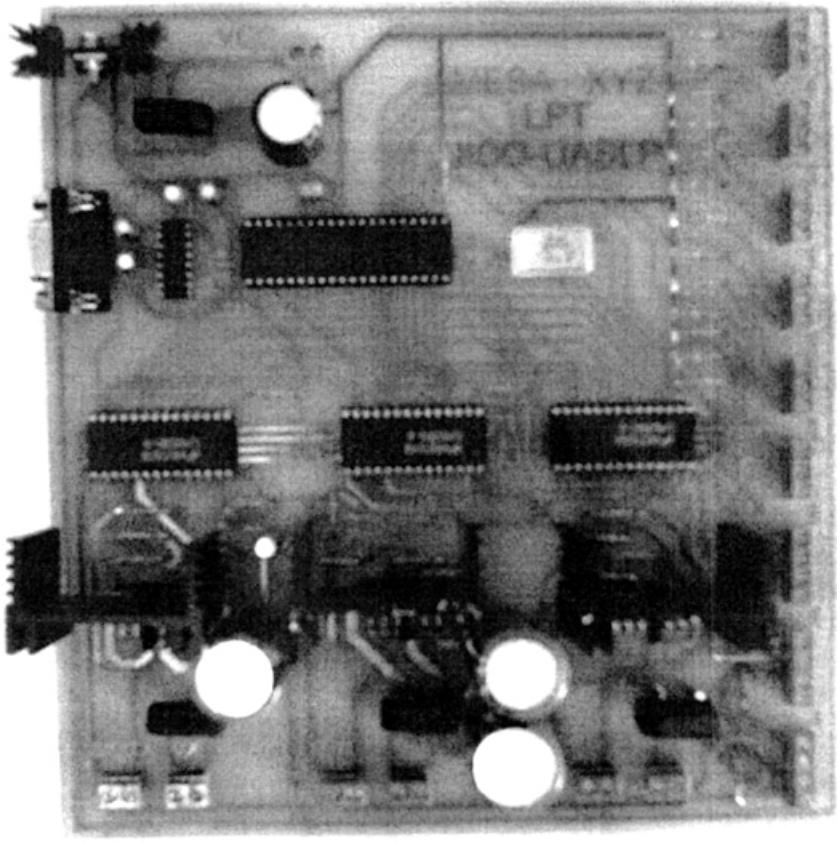


Fig. 3.4.3-Tarjeta de control

3.5.- Funcionamiento.

A continuación se da una explicación del funcionamiento de la tarjeta de control, la cual se dividió en 5 secciones principales, siendo las siguientes:

- Fuente de Alimentación
- Circuito de control
- Control de motores
- Etapa de potencia
- Comunicación serial

3.5.1 Fuente de Alimentación.

Para la alimentación de los componentes electrónicos se utiliza un voltaje de 5 V.

Se utiliza un transformador (externo) que reduce el voltaje de 110 V a 12 V de corriente alterna. La corriente pasa a través de un puente de diodos que rectifica el voltaje y el capacitor C9 lo filtra. El IC2 es un regulador LM705 entrega a su salida un voltaje de 5V que alimenta la mayoría de los componentes de la tarjeta de control.

Se muestra a continuación el circuito esquemático de la fuente de alimentación:

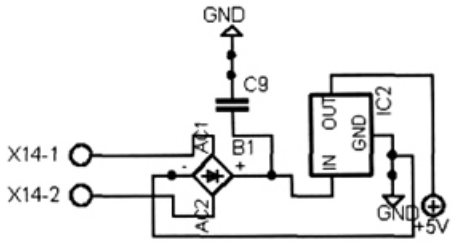


Fig. 3.5.1.- Fuente de Alimentación

3.5.2.-Circuito de control.

El PIC 16F877A es el principal componente de esta tarjeta. El puerto A es utilizado para recibir las señales de fin de carrera de los fototransistores, los cuales están colocados en la mesa para detectar cuando la mesa llega al final o inicio de su recorrido en cualquiera de los ejes.

Mediante el puerto B se envían los comandos y datos a los microcontroladores LM629N y a través de el también se lee su registro de status.

Para seleccionar uno de los microcontroladores LM629N se envía un cero lógico a su terminal chip select (CS) y solamente un dispositivo puede ser seleccionado a la vez. Estas señales de control son enviadas por el PIC16F877A a través de los pines RC0, RC1 y RC2.

El pin RC3 se utiliza para activar la terminal "Port Select" (PS) de los tres microcontroladores.

A través de los pines RC4 y RC5 se manejan las señales "read" y "write". TX/RC6 y RX/RC7 son los pines utilizados para la comunicación serial con ayuda del MAX232. TX/RC6 es la terminal de transmisión de datos y RX/RC7 la terminal de recepción.

En la siguiente figura se muestra el PIC16F877A conectado a un resonador de 20 MHz.

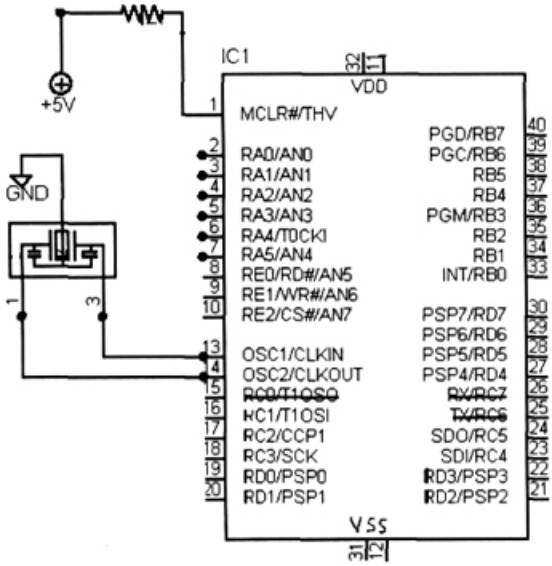


Fig. 3.5.2.- PiC16F877A

3.5.3.-Control de motores.

En la figura 3.5.3 se muestra el circuito esquemático de uno de los microcontroladores LM629N.

Por los pines 2 y 3 reciben las señales A y B del encoder que está acoplado al motor.

A través de los pines 4 a 11 se maneja la información de datos y comandos que se intercambia con el PIC 16F877A.

La señal "chip select" es recibida por el pin 12, la señal "read" por el pin 13, la señal "write" por el pin 15 y la señal "PS" por el pin 16.

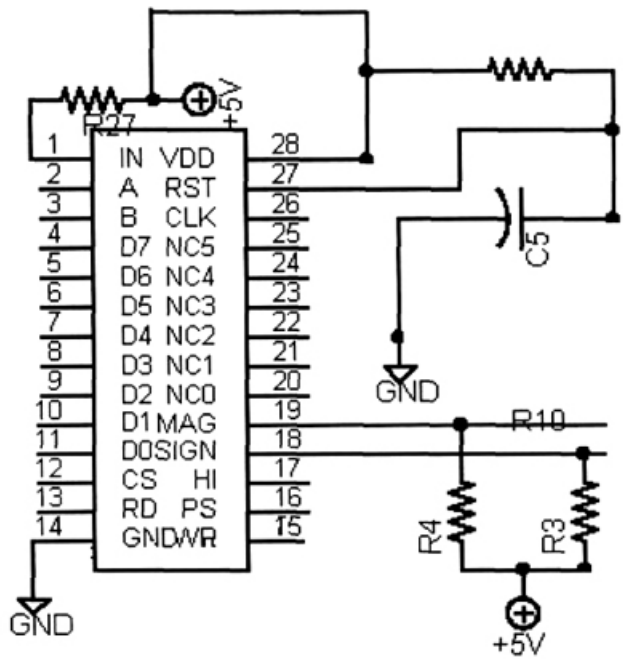


Fig. 3.5.3.- LM629N

3.5.4.-Etapa de potencia.

El LMD18200 es el puente H utilizado, se muestra a continuación su diagrama esquemático:

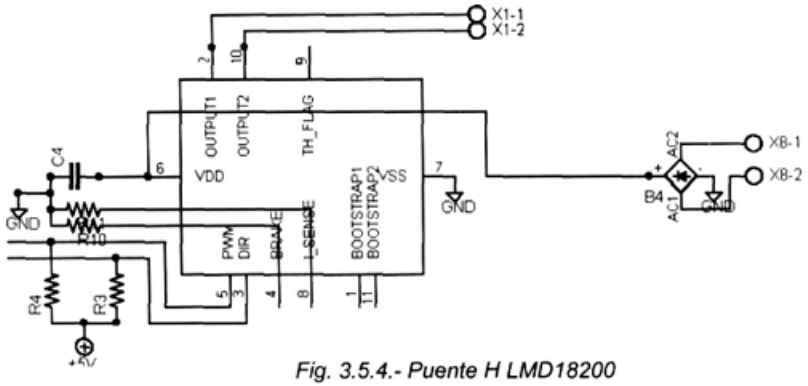


Fig. 3.5.4.- Puente H LMD18200

La fuente de alimentación entrega al puente H un voltaje de 33 V por los pin es 6 y 7.

Las salidas 2 y 10 del LMD18200 entregan el voltaje de alimentación al motor de corriente directa.

Para darle el sentido de giro al motor depende de la señal recibida por el pin de dirección (3); mientras que por el pin 5 recibe la señal de magnitud modulada en ancho de pulso (PWM).

3.5.5.-Comunicación serial.

El PIC 16F877A se comunica con la computadora de forma serial utilizando el estándar RS-232, a través del MAX 232 y de un conector DB9 (Fig. 3.5.5b)

Como se muestra en la figura (Fig. 3.5.5a) el cable serial utiliza 3 líneas para efectuar la comunicación. Una línea para transmisión de datos, una para recepción de datos y una línea de referencia (tierra). Los jumpers de las líneas 4-6 y 7-8 son utilizados con el fin de deshabilitar el modo de comunicación de Handshaking por hardware.

La tarjeta de control está configurada como un sistema DTE (Equipo Terminal de Datos) que es un dispositivo que actúa como fuente de datos.

En este caso la computadora adquiere la función de un sistema DCE (Equipo de Comunicación de Datos) que se definen como dispositivos que proporcionan las funciones necesarias para establecer, mantener y terminar una conexión para transmisión de datos.

Los parámetros requeridos para la comunicación son los siguientes:

- 9600 Baudios
- 8 bits de datos
- 1 bit de parada
- Sin paridad.

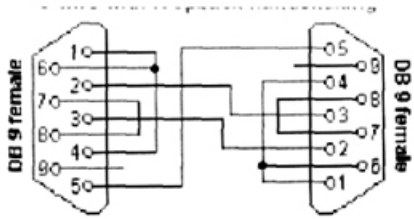


Fig. 3.5.5 a

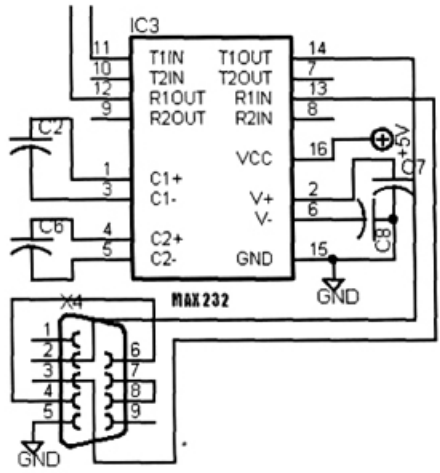


Fig. 3.5.5- Max 232 y Conector DB9

CONCLUSIONES:

Una vez realizado el software de generación de mascarillas en lenguaje C++, el cual tomó la mayor parte del tiempo del proyecto de tesis debido a las características que se requería de él, implementado la mesa X-Y y la tarjeta de control de motores, se puede decir que se ha dado un gran avance al sistema requerido para la generación de mascarillas.

Uno de los puntos críticos en el proyecto en cuanto al tiempo, fue la cotización de partes con diferentes proveedores y los tiempos de entrega de los materiales lo cual afectó definitivamente el avance del proyecto.

El software de generación de mascarillas puede llegar a fallar a causa de la memoria, para evitar esto se configura el compilador de la siguiente manera:

- ✓ Options/ Debuger/ Program/ Head Size, en esta sección se recomienda tener 512 kbytes de memoria para evitar que falle el programa durante la ejecución.

Al inicio del proyecto se planteó utilizar motores a pasos para la transmisión de movimiento a la mesa, pero al realizar pruebas mecánicas y eléctricas al sistema, se observó que los motores a pasos funcionan a un velocidad relativamente baja, lo cual hace lento el avance de la mesa, por lo que se optó por el uso de motores de corriente directa e implementar la tarjeta de control para este tipo de motores.

La tarjeta de control se implementó para poder controlar 3 motores de corriente directa, aunque en el proyecto de tesis solo se requiera de 2, pero se esta analizando si se requiere un tercer motor para dar el sentido de giro al cortador de rubylith.

TRABAJO A FUTURO:

Como trabajo complementario al presente proyecto de tesis, será implementar el software, que convierta el archivo generado al guardar la mascarilla en píxeles, a coordenadas reales y de esta manera enviarlas al PIC16F877A en forma serial desde la PC.

También buscar el mejor diseño a la mesa para el cortador del rubylith, de manera que realice un corte perfecto sobre el material.

Una vez contando con estos 2 puntos, se pude decir que el sistema estará al 100% para su objetivo final, que es la de realizar en forma automática el diseño de mascarillas en su primera etapa, para el proceso de Litografía muy utilizado en trabajos de investigación en el IICO.

ANEXOS:

ANEXO A [3]

FUNCIONES GRÁFICAS

arc

```
int  centrox, centroy, angulocomienzo, angulofin, radio;
arc  (centrox, centroy, angulocomienzo, angulofin, radio);
```

Dibuja un arco circular; los valores angulocomienzo y angulofin deben especificarse en grados (0-360); *centrox, centroy* y *radio* deben ir en pixels.

bar

```
int  izquierda, arriba, derecha, abajo;
bar  ( izquierda, arriba, derecha, abajo );
```

Dibuja una barra rectangular sólida, haciendo uso de los valores vigentes del color y patrón de relleno.

bar3d

```
int  izquierda, arriba, derecha, abajo, profundidad, indicatapa;
bar3d  ( izquierda, arriba, derecha, abajo, profundidad, indicatapa );
```

Delimita el contorno de una barra rectangular tridimensional haciendo uso del color y del estilo de línea vigentes. Posteriormente procede a su relleno con los valores vigentes del color y patrón de relleno.

circle

```
int  centrox, centroy, radio;
arc  (centrox, centroy, radio);
```

Dibuja un círculo del color vigente, centrado en las coordenadas especificas y con el radio dado (en pixels).

cleardevice

```
cleardevice ( );
```

Borra la pantalla gráfica y desplaza la posición vigente (PV) a la posición (0,0).

clearviewport

```
clearviewport ( );
```

Borra la ventana gráfica vigente, desplazando la posición vigente (PV) a la posición (0,0) de la ventana gráfica.

closegraph

```
closegraph ( );
```

Utiliza graphfreemem para liberar la memoria reversada para el sistema gráfico y restituye la pantalla al modo de texto detectando cuando se llamó a la función initgraph.

detectgraph

```
int controladorgrafico, modografico;
detectgraph (&controladorgrafico, &modografico);
```

Suele ser llamada por initgraph para examinar el hardware. Devuelve sus valores al controlador gráfico. También devuelve el mayor de los modos válidos.

drawpoly

```
int puntos;
int poligono [ ] = { 100, 100, 200, 100, 200, 200, 100, 200, 100, 100 };
drawpoly ( puntos, poligono )
```

Dibuja el contorno de un polígono con el valor del color vigente; *puntos* proporciona el número de vértices del polígono; *poligono* apunta a una secuencia de pares enteros; cada uno de ellos define el par coordenado *x, y* de cada vértice del polígono. Si se desea dibujar una figura cerrada de *n* vértices, *puntos* debe valer *n+1*, y el *n-ésimo* par coordenado (el último) debe ser igual al primer par coordenado.

En caso de error, graphresult devuelve -6.

getaspretratio

```
int aspx, aspy;
getaspectratio ( &aspx, &aspy );
drawpoly ( puntos, poligono )
```

Devuelve los aspectos según los ejes x e y. El factor de aspecto se calcula como *aspx/aspy* y se utiliza como factor de escala en las funciones arc y circle, así como en las rutinas pieslice para conseguir que los círculos salgan redondos en la pantalla.

Cada controlador gráfico y cada modo gráfico tienen un factor de aspecto asociado determinado por la altura y la anchura relativa de los pixels.

getbkcolor

```
int colorfondo = getbkcolor ( );
```

Devuelve el valor vigente del color de fondo.

getmaxcolor

```
int MaxColores = getmaxcolor ( ) + 1;
```

Devuelve el mayor de los colores validos (tamaño de la paleta -1) para el modo gráfico vigente.

getmaxx

```
int maxX = getmaxx ( );
```

Devuelve la máxima coordenada de pantalla según el eje x (máx PV) para el controlador y el modo gráfico vigente.

initgraph

```
int controladorgrafico = DETECT, modografico;
char *caminocontrolador;
initgraph (&controladorgrafico, &modografico, caminocontrolador);
```

Inicia el sistema gráfico mediante la carga del controlador gráfico y la restitución del modo gráfico.

line

```
int comienzox, comienzoy, finx, finy;
line (comienzox, comienzoy, finx, finy);
```

Dibuja una línea entre los dos puntos especificados, con el color, el estilo de línea y la anchura vigentes, sin modificar la posición vigente (PV)

outtext

```
outtext ( "Cadena para la ventana gráfica" )
```

Visualiza sobre la ventana gráfica una cadena que comienza en PV. Para ello hace uso de los valores vigentes de la fuente, el color, el tamaño del carácter, la dirección y la justificación del texto.

putpixel

```
int posx, posy, color;
putpixel (posx, posy, color );
```

Carga en el pixel dado por (*posx, posy*) el color especificado.

ANEXO B.

HOJA DE DATOS LM629N.

LM628 Programming Guide

National Semiconductor
Application Note 693
Steven Hunt
January 1999

INTRODUCTION

The LM628/LM629 are dedicated motion control processors. Both devices control DC and brushless DC servo motors, as well as, other servomechanisms that provide a quadrature incremental feedback signal. Block diagrams of typical LM628/LM629-based motor control systems are shown in *Figures 1, 2*.

As indicated in the figures, the LM628/LM629 are bus peripherals; both devices must be programmed by a host processor. This application note is intended to present a concrete starting point for programmers of these precision motion controllers. It focuses on the development of short programs that test overall system functionality and lay the groundwork for more complex programs. It also presents a method for tuning the loop-compensation PID filter. (Note 1)

REFERENCE SYSTEM

Figure 15 is a detailed schematic of a closed-loop motor control system. All programs presented in this paper were developed using this system. For application of the programs in other LM628-based systems, changes in basic programming structure are not required, but modification of filter coefficients and trajectory parameters may be required.

1. PROGRAM MODULES

Breaking programs for the LM628 into sets of functional blocks simplifies the programming process; each block executes a specific task. This section contains examples of the principal building blocks (modules) of programs for the LM628.

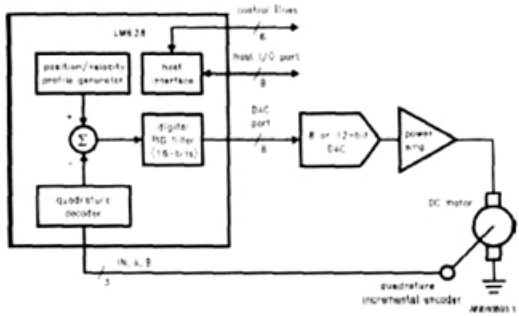


FIGURE 1. LM628-Based Motor Control System

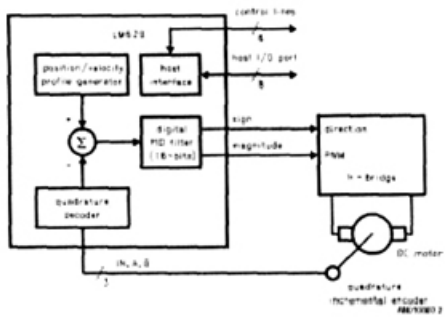


FIGURE 2. LM629-Based Motor Control System

Note 1: For the remainder of this paper, all statements about the LM628 also apply to the LM629 unless otherwise noted.

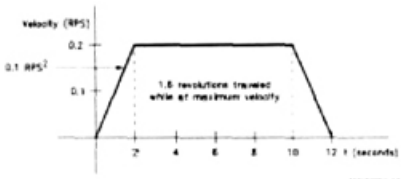


FIGURE 10. Velocity Profile for Simple Absolute Position Move Program

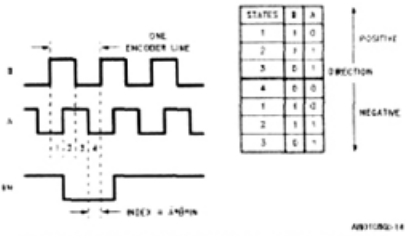


FIGURE 11. 3-Channel Quadrature Encoder Signals

The resolution of a quadrature incremental encoder is usually specified as a number of *lines*. This number indicates the number of cycles of the output signals for each complete shaft revolution. For example, an N-line encoder generates N cycles of its output signals during each complete shaft revolution.

By definition, two signals that are in quadrature are 90° out of phase. When considered together, channels A and B (*Figure 11*) traverse four distinct digital states during each full cycle of either channel. Each state transition represents one *count* of shaft motion. The leading channel indicates the direction of shaft rotation.

Each line, therefore, represents one cycle of the output signals, and each cycle represents four encoder counts.

$$\left(N\frac{\text{CYCLES}}{\text{REVOLUTION}}\right)\cdot\left(4\frac{\text{COUNTS}}{\text{CYCLE}}\right)=4N\frac{\text{COUNTS}}{\text{REVOLUTION}}$$

The reference system uses a one thousand line encoder.

$$\left(1000\frac{\text{CYCLES}}{\text{REVOLUTION}}\right)\cdot\left(4\frac{\text{COUNTS}}{\text{CYCLE}}\right)=4000\frac{\text{COUNTS}}{\text{REVOLUTION}}$$

Sample Period

Sampling of actual shaft position occurs at a fixed frequency, the reciprocal of which is the system sample period. The system sample period is the unit of time upon which shaft acceleration and velocity are based.

$$T_S=(2048)\cdot\left(\frac{1}{f_{CLOCK}}\right)\text{ System Sample Period}$$

The reference system uses an 8 MHz clock. The sample period of the reference system follows directly from the definition.

$$T_S=(2048)\cdot\left(\frac{1}{8\times10^6\text{ Hz}}\right)=256\cdot10^{-6}\frac{\text{SECONDS}}{\text{SAMPLE}}$$

Trajectory Parameters Calculations

The shaft will accelerate at 0.1 rev/sec² until it reaches a maximum velocity of 0.2 rev/sec, and then decelerate to a stop exactly two revolutions from the starting position.

Trajectory parameters calculations for this move are detailed in *Figure 12*.

Comments

After completing the move, the control system will attempt to hold the shaft at its current absolute position. The shaft will feel lightly "spring loaded". If forced away from its desired resting position and released, the shaft will move back to the desired position.

REFERENCIAS:

[1] Dispositivos Semiconductores
Jasprit Singh
Mc Graw-Hill

[2] Mecatrónica
Sistemas de Control Electrónico
En Ingeniería Mecánica y Eléctrica
2da. Edición, W. Bolton
Alfaomega

[3] Programación de Gráficos en Turbo C++
Benn Ezzell

[4] Programación en Turbo C
Herbert Schildt
Mc Graw-Hill

[5] Programación en C
Byron S Gottfried

[6] www.DanaherMotion.com

[7] http://localhost/ConClase/graficos/curso/para-pdf/index.php?cap=002