

UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE TECNOLOGÍA
CARRERA: ELECTRONICA Y TELECOMUNICACIONES



TRABAJO DE APLICACIÓN:

**“IMPLEMENTACION DE UN PROTOTIPO DE GRABADO DE PLACAS
IMPRESAS PCB CON SOFTWARE Y HARDWARE LIBRE ”**

**Examen de Grado presentado para obtener el Grado de Licenciado en Electrónica
y Telecomunicaciones**

POSTULANTE: Mario Aldo Sejas Mollinedo

La Paz - Bolivia
Noviembre, 2017

DEDICATORIA

A mis queridos padres Mario e Hilda, de los cuales nunca me faltó su apoyo para seguir adelante, tanto en los buenos y malos momentos, supieron inculcarme valores y principios para tomar el camino apropiado.

También agradecer a mis queridas hermanas Ana, Silvia y tía Graciela por su cariño y amor en todo momento me brindaron su apoyo, que fueron esenciales para lograr el propósito.

AGRADECIMIENTOS

Un agradecimiento a todos los docentes de la carrera, de quienes tuve el privilegio de recibir enseñanzas y conocimientos, un agradecimiento especial al Ing. Roger Guachalla quien me guió para llevar a cabo exitosamente el presente proyecto, muchas gracias.

INDICE DE CONTENIDO

RESUMEN.....	III
CAPÍTULO I.....	IV
1. ANTECEDENTES DEL PROYECTO.....	IV
1.1 PLANTEAMIENTO DEL PROBLEMA	IV
1.2 PLANTEAMIENTO DE OBJETIVOS	V
1.2.1 OBJETIVO PRINCIPAL.....	V
1.2.2 OBJETIVOS SECUNDARIOS	V
1.3 JUSTIFICACION.....	VI
1.3.1 JUSTIFICACION TECNOLÓGICA	VI
1.3.2 JUSTIFICACION SOCIAL.....	VI
1.3.3 JUSTIFICACION ACADEMICA	VI
1.4 DELIMITACIONES	VII
1.4.1 DELIMITACION TEMPORAL.....	VII
1.4.2 DELIMITACION TEMÁTICA.....	VII
1.5 METODOLOGÍA	VIII
CAPITULO II	1
2. MARCO TEÓRICO CONCEPTUAL.....	1
2.1 DEFINICION DE CIRCUITO IMPRESO PCB.....	1
2.1.1 FUNCIONES DE LOS CIRCUITOS IMPRESOS	2
2.1.2 ELEMENTOS BASICOS DE UN CIRCUITO IMPRESO	2
2.2 CONTROL NUMERICO COMPUTARIZADO (CNC).-.....	2
2.2.1 PRINCIPIO DE FUNCIONAMIENTO DE UNA CNC	3
2.3 CAD/CAM EN EL PROCESO DE DISEÑO.....	3
2.4 CODIGO-G.....	4
2.5 GRBL.....	5
2.6 MOTORES DE PASO A PASO (PAP).....	5
2.6.1 PRINCIPIO DE FUNCIONAMIENTO.....	6
2.6.2 TIPOS DE MOTORES PASO A PASO.....	6
2.6.3 MOTOR PASO A PASO DE REDUCTANCIA VARIABLE	6
2.6.4 MOTOR PASO A PASO DE IMAN PERMANENTE	7
2.6.4.1 MOTORES UNIPOLARES	7
2.6.4.2 MOTORES BIPOLARES.....	9
2.7 HARDWARE LIBRE	10
2.7.1 PLACA DE DESARROLLO ARDUINO UNO.....	10

2.7.1.1	ESPECIFICACIONES TECNICAS DE ARDUINO UNO	11
2.7.2	CNC SHIELD.....	11
2.7.2.1	ESPECIFICACIONES TECNICAS DE LA CNC SHIELD	12
2.7.3	DRIVERS A4988 POLOLU.....	13
2.8	SOFTWARE LIBRE.....	13
2.9	FRESADO CNC	14
CAPITULO III.....		15
3.	INGENIERIA DEL PROYECTO	15
3.1	Diagrama en bloques del funcionamiento del prototipo CNC	15
3.2	Funcionamiento del Software	16
3.2.1	CAD/CAM Diseño de pistas de circuito a implementar	16
3.2.1.1	Arquitectura de un bloque de código-g.....	18
3.2.2	Universal G-Code Sender	19
3.2.3	Firmware GRBL para Arduino UNO.....	20
3.2.3.1	Diagrama en bloques del GRBL	22
3.2.3.2	Ejemplo de interpretación de código-g en bloques del GRBL	22
3.3	Funcionamiento del Hardware.....	24
3.3.1	Arduino Uno.....	24
3.3.2	CNC Shield	25
3.3.3	Drivers A4988 Pololu.....	25
3.3.3.1	Configuración de corriente de Driver A4988.....	27
3.4	Diagrama de Conexiones del prototipo CNC.-	30
3.5	Estructura mecánica del prototipo	31
3.5.1	Rieles para los Ejes.-	31
3.5.2	Tornillo sin fin.-	32
3.5.3	Acoples Flexibles	32
3.5.4	Fresa para grabado en PCB	33
CAPITULO IV.....		34
4.	COSTOS	34
4.1	Costos fijos.....	34
5.	CONCLUSIONES	35
6.	RECOMENDACIONES	35
7.	BIBLIOGRAFIA.....	36
8.	ANEXOS	36

RESUMEN

En el presente proyecto se explicara detalladamente la implementación y puesta en funcionamiento de un prototipo que permita la automatización en el proceso de grabado de circuitos en PCB, con la característica de que será un proceso con menos riesgos y más ventajoso, con el fin de ser un procedimiento alternativo a los métodos químicos que actualmente utilizan la mayoría de los estudiantes de la carrera.

La problemática surgió porque se observo que actualmente, una gran mayoría de los estudiantes de la carrera de Electrónica y Telecomunicaciones, en el proceso de elaboración de circuitos, utilizan métodos químicos convencionales para el grabado de los circuitos PCB, siendo que los mismos incluyen una constante manipulación de elementos que pueden ser peligrosos para la salud y el medio ambiente.

Actualmente, las grandes industrias de electrónica han logrado alcanzar una alta automatización en la fabricación de circuitos impresos, usando diversas técnicas y maquinaria que si bien están a la venta, los precios son extremadamente prohibitivos, también considerando que los mismos son software y hardware propietario o privado al público. Es por lo mencionado anteriormente que se utilizara hardware y software libre, logrando así reducir drásticamente los costos y reducir la complejidad en la implementación, para así poder llegar a toda la población estudiantil de la carrera.

CAPÍTULO I

1. ANTECEDENTES DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA

Actualmente, la mayoría de los estudiantes de la carrera de Electrónica y Telecomunicaciones, se ven en la necesidad de realizar prácticas y proyectos con circuitos impresos, dicho proceso consiste en retirar ciertas regiones de cobre de las placas PCB, quedando así circuitos y componentes electrónicos conectados entre sí.

Para dicho proceso existen varios métodos, el más empleado es el que consiste en aplicar cloruro férrico para hacer el quemado del cobre, lo cual conlleva la manipulación directa de dichos elementos que son, y representan un gran riesgo para la salud de los estudiantes. Cabe mencionar que en varias universidades del exterior ya se prohibieron realizar estos procedimientos en laboratorios o ambientes cerrados, debido al potencial riesgo que pueda ocasionar ya sea a la integridad del estudiante, como al medio ambiente.

Por otro lado, hoy en día las grandes industrias de electrónica han logrado alcanzar una alta automatización en la fabricación de circuitos impresos, usando diversas técnicas y maquinaria que si bien están a la venta, los precios son extremadamente prohibitivos para la economía de los estudiantes, también es necesario señalar que los mismos funcionan con software y hardware propietario o privado al público.

Consecuentemente a lo mencionado, el estudiante se ve obligado a poner en riesgo su salud empleando métodos conservadores de quemado de placas. Es por todo eso que es necesario ofrecer un procedimiento alternativo, que sea menos riesgoso y esté al alcance de todos los estudiantes.

1.2 PLANTEAMIENTO DE OBJETIVOS

1.2.1 OBJETIVO PRINCIPAL

Implementar un prototipo de grabado de circuitos impresos PCB operable y ensamblado con software y hardware libre, aportando como una alternativa para comunidad estudiantil y principalmente haciendo énfasis en costos reducidos y fácil implementación.

1.2.2 OBJETIVOS SECUNDARIOS

- Aplicar conocimientos en la manipulación y programación de placas de desarrollo basadas en microcontroladores.
- Implementar un dispositivo con las características necesarias básicas para el grabado de circuitos PCB.
- Motivar a la comunidad estudiantil en general, para optar por un procedimiento no convencional en el grabado de circuitos impresos.
- Demostrar los beneficios de la utilización de software y hardware libre.

1.3 JUSTIFICACION

1.3.1 JUSTIFICACION TECNOLÓGICA

Viendo las necesidades planteadas para el proyecto y considerando que se deben reducir costos, se hará uso de software libre, utilizando Universal G-code Sender y firmware GRBL, ambos de código abierto. Para la etapa electrónica se utilizara la placa de desarrollo Arduino Uno, el cual es una plataforma de hardware libre, agregando módulos especiales como la CNC Shield y drivers de corriente A4988.

1.3.2 JUSTIFICACION SOCIAL

Tendrá una gran importancia en la población estudiantil, ya que éste proyecto ofrecerá una alternativa diferente, novedosa, menos riesgosa, mas económica, y más ecológica para la elaboración de circuitos impresos. Así también se hará la implementación haciendo énfasis en reducción de costos y fácil implementación, para poder llegar al mayor porcentaje posible de estudiantes, de primer hasta último semestre.

1.3.3 JUSTIFICACION ACADEMICA

Se aplico los conocimientos adquiridos a lo largo de la carrera de Electrónica y Telecomunicaciones, fundamentalmente en el Área Digital, como ser en las asignaturas de Laboratorio de Microprocesadores I y II, y Laboratorio de Procesamiento Digital de Señales.

1.4 DELIMITACIONES

1.4.1 DELIMITACION TEMPORAL

Tomando en cuenta que se implementara el prototipo contando con todos los materiales necesarios, y empezando tanto la etapa mecánica y electrónica desde cero, se prevé la conclusión en un tiempo estimado de 2 meses.

1.4.2 DELIMITACION TEMÁTICA

Este proyecto presenta una propuesta, para la implementación de un prototipo de CNC para el grabado de circuitos impresos.

Para el proceso informático se utilizara software libre, como ser Universal G-code Sender y firmware GRBL, y para la parte electrónica el hardware libre, Arduino Uno, CNC Shield, y drivers de corriente A4988. Todos los materiales serán seleccionados para poder reducir costos y facilitar la implementación.

1.5 METODOLOGÍA

El Método Descriptivo, el objeto de la investigación descriptiva consiste en evaluar ciertas características de una situación particular en uno o más puntos del tiempo. En esta investigación se analizan los datos reunidos para descubrir así, cuales variables están relacionadas entre sí. Este método es el que consiste en desarrollar una caracterización de las situaciones y eventos de cómo se manifiesta el objeto de investigación, ya que éste busca especificar las propiedades importantes del problema en cuestión, mide independientemente los conceptos y también puede ofrecer la posibilidad de predicciones aunque sean muy rudimentarios.

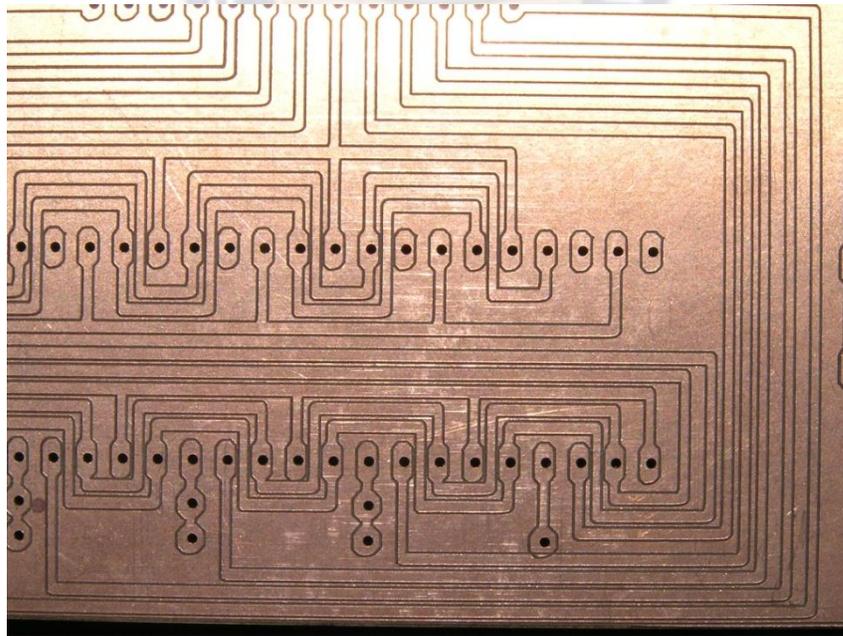
En este proyecto de Implementación de un prototipo de grabado (PCB) con hardware y software libre se utilizara el método descriptivo, porque se hará un estudio previo de la situación en la que se encuentran los estudiantes, así también se obtendrán datos y experiencias que utilizan más comúnmente para la elaboración de placas de circuito impreso, para así finalmente adecuar el prototipo de acuerdo a las posibilidades económicas, conocimientos prácticos y teóricos que tengan los estudiantes desde primer hasta el último semestre de la carrera.

CAPITULO II

2. MARCO TEÓRICO CONCEPTUAL

2.1 DEFINICION DE CIRCUITO IMPRESO PCB

Un circuito impreso está constituido de material aislante, tal como fibra de vidrio o fenólico con trayectorias conductoras. El propósito de todos los circuitos impresos es proporcionar trayectorias eléctricas para conectar todas las componentes de un circuito, estas trayectorias se colocan en uno o ambos lados del material aislante, es decir que se trata de una base no conductora sobre la cual se adhiere una capa de material conductor como el cobre, que posteriormente será tratada para formar vías de conexión entre componentes electrónicos; que en conjunto recibe el nombre de circuito electrónico.



*Figura N°1 Circuito impreso PCB
Fuente: www.robio.be*

2.1.1 FUNCIONES DE LOS CIRCUITOS IMPRESOS

La función que cumplen los circuitos impresos son:

- Proporcionar una base para alojar a los componentes electrónicos que conforman el circuito.
- Proporcionar interconexiones entre componentes.

En el diseño de los PCB se cumplen variadas reglas las cuales obedecen a las características de cada máquina o caso diferente en el que se quiera implementar un circuito electrónico. En el diseño de PCB también pueden influir factores tales como el tipo de material dieléctrico de la base, número de capas, densidad y muchos otros factores propios de cada caso.

2.1.2 ELEMENTOS BASICOS DE UN CIRCUITO IMPRESO

- (Dieléctrico) base para el montaje de los elementos.
- Agujeros para el montaje de los componentes electrónicos.
- (Pistas) Conexiones entre componentes electrónicos.

2.2 CONTROL NUMERICO COMPUTARIZADO (CNC).-

Se considera de Control Numérico por Computador, también llamado CNC (en inglés Computer Numerical Control) a todo dispositivo capaz de dirigir el posicionamiento de un órgano mecánico móvil mediante órdenes elaboradas de forma totalmente automática a partir de informaciones numérica en tiempo real. Para maquinar una pieza se utiliza un sistema de coordenadas que especificaran el movimiento de la herramienta de corte. Entre las operaciones de maquinado que se pueden realizar en una maquina CNC se encuentran las de torneado y de fresado. sobre la base de esta combinación es posible generar la mayoría de las piezas de industria.



Figura N°2 Maquina CNC
Fuente: <https://www.multicam.com>

2.2.1 PRINCIPIO DE FUNCIONAMIENTO DE UNA CNC

Los puntos más importantes del funcionamiento de una maquina CNC se describen a continuación:

- Para mecanizar una pieza se usa un sistema de coordenadas que especificaran el movimiento de la herramienta de corte.
- En una maquina CNC, a diferencia de una maquina convencional o manual, una computadora controla la posición y velocidad de los motores.
- Las maquinas CNC son capaces de mover la herramienta al mismo tiempo en los tres ejes para ejecutar trayectorias tridimensionales.

2.3 CAD/CAM EN EL PROCESO DE DISEÑO

EL diseño y la fabricación asistidos por ordenador (CAD/CAM) es una disciplina que estudia el uso de sistemas informáticos como herramienta de soporte en todos los procesos involucrados en el diseño y la fabricación de cualquier tipo de producto.

CAD es el acrónimo de ‘Computer Aided Design’ o diseño asistido por computador. Se trata de la tecnología implicada en el uso de ordenadores para realizar tareas de creación,

modificación, análisis y optimización de un diseño. De esta forma, cualquier aplicación que incluya una interfaz gráfica y realice alguna tarea de ingeniería se considera software de CAD.

El termino CAM se puede definir como el uso de sistemas informáticos para la planificación, gestión y control de las operaciones de una planta de fabricación mediante una interfaz directa o indirecta entre el sistema informático y los recursos de producción.

2.4 CODIGO-G

La programación nativa de la mayoría de las máquinas de Control Numérico Computarizado se efectúa mediante un lenguaje de bajo nivel llamado G & M.

Se trata de un lenguaje de programación vectorial mediante el que se describen acciones simples y entidades geométricas sencillas (básicamente segmentos de recta y arcos de circunferencia) junto con sus parámetros de maquinado (velocidades de husillo y de avance de herramienta).

El nombre G & M viene del hecho de que el programa está constituido por instrucciones Generales y Misceláneas.

<code>G90 G71</code>	(cotas absolutas referidas al punto 0,0; Programación en mm)
<code>G00 X0.0 Y0.0</code>	(posicionamiento rápido lineal al punto 0,0 del plano XY)
<code>G01 X10.0</code>	(movimiento lineal de 10mm en la dirección X positiva)
<code>G01 Y10.0</code>	(movimiento lineal de 10mm en la dirección Y positiva)
<code>G01 X0.0</code>	(movimiento lineal de 10mm en la dirección X negativa)
<code>G01 Y0.0</code>	(movimiento lineal de 10mm en la dirección Y negativa)

Figura N°3 Ejemplo de Código-G

Fuente: <http://wiki.ead.pucv.cl>

2.5 GRBL

GRBL es un controlador de fresadora CNC de alto rendimiento, libre y de código abierto, escrito en C optimizado y que se ejecuta en Arduino. Está escrito en C optimizado utilizando todas las características inteligentes de los chips ATmega328P del Arduino para lograr la sincronización exacta y la operación asincrónica. Es capaz de mantener una frecuencia de pasos de más de 30 kHz y ofrece un flujo limpio, libre de jitter (variabilidad en los cantos de subida y bajada) de impulsos de control.

2.6 MOTORES DE PASO A PASO (PAP)

Un motor paso a paso es un dispositivo electromecánico que convierte una serie de pulsos eléctricos en desplazamientos angulares, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.

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 de poder moverlos un paso a la vez por cada pulso que se le aplique. Este paso puede variar desde 90° hasta pequeños movimientos de 1.8°. Es por eso que ese tipo de motores son muy utilizados, ya que pueden moverse a deseo del usuario según la secuencia que se les indique a través de un microcontrolador.

Estos motores poseen la habilidad de quedar enclavados en una posición si una o más de sus bobinas está energizada o bien totalmente libres de corriente.



Figura N°4 Vista Interna de Motor de pasos PAP
Fuente: <http://www.ingmecafenix.com>

2.6.1 PRINCIPIO DE FUNCIONAMIENTO

El principio de funcionamiento está basado en un estator construido por varios bobinados en un material ferromagnético y un rotor que puede girar libremente en el estator.

Estos diferentes bobinados son alimentados uno a continuación del otro y causan un determinado desplazamiento angular que se denomina “paso angular” y es la principal característica del motor.

2.6.2 TIPOS DE MOTORES PASO A PASO

Existen tres tipos de motores paso a paso:

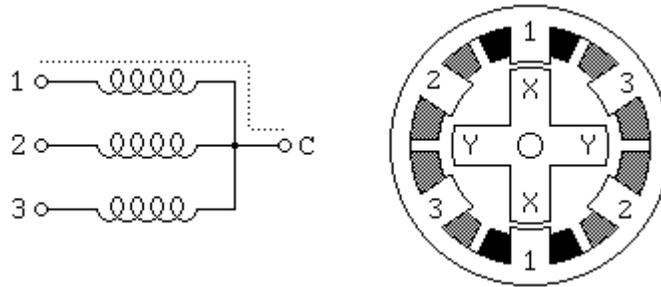
- De reluctancia variable
- De imán permanente
- Híbrido

2.6.3 MOTOR PASO A PASO DE REDUCTANCIA VARIABLE

Este motor no utiliza un campo magnético permanente, como resultado puede moverse sin limitaciones o sin un par de parada. Este tipo de montaje es el menos común y se usa, generalmente, en aplicaciones que no requieren un alto grado de par de fuerza, como puede ser el posicionamiento de un mando de desplazamiento.

Se desarrolló con objeto de poder conseguir unos desplazamientos angulares más reducidos que en el caso anterior, sin que por este motivo haya de aumentarse considerablemente el número de bobinados. El estator presentará la forma cilíndrica habitual conteniendo generalmente un total de tres devanados distribuidos de tal forma que existirá un ángulo de 120° aproximadamente entre dos de ellos.

Si el estator del motor tiene tres bobinas conectadas, con un terminal común, a todas las bobinas, será probablemente un motor de reluctancia variable. El conductor común se conecta habitualmente al borne positivo y las bobinas son alimentadas siguiendo una secuencia consecutiva.



*Figura N°5 Motor con reluctancia variable
Fuente: <http://www.ingmecafenix.com>*

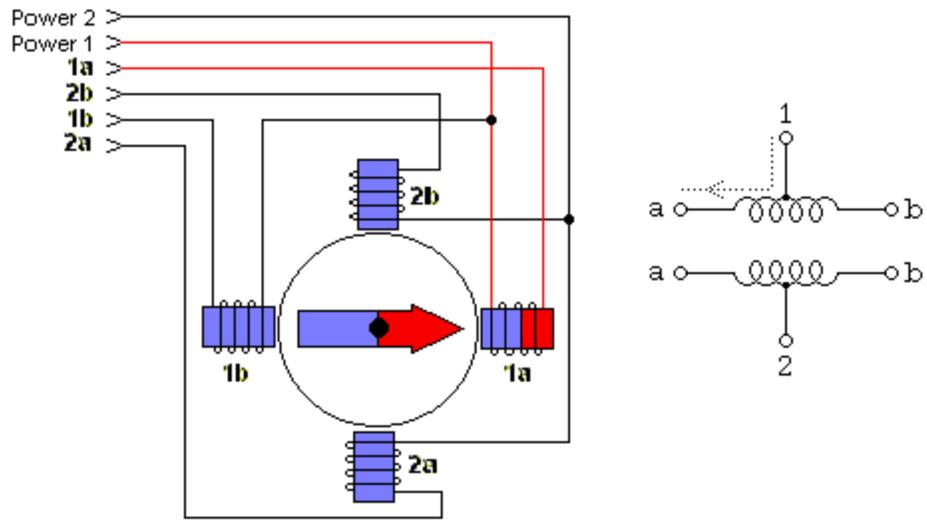
2.6.4 MOTOR PASO A PASO DE IMAN PERMANENTE

Existen dos tipos de motores de imán permanente que son los más utilizados en robótica:

- Unipolares
- Bipolares

2.6.4.1 MOTORES UNIPOLARES

Estos motores cuentan con dos bobinas con un punto medio de los cuales salen los cables hacia el exterior; estos cables se conectan a la fuente mientras que los extremos de las bobinas son aterrizadas para cerrar el circuito; dependiendo del tipo de motor, las líneas comunes pueden ser independientes o no. Esta configuración puede ser vista de las siguientes formas: que el motor tiene dos bobinas pequeñas conectadas a un punto en común, o que una bobina está dividida en dos por medio de un punto común. Ahora, y dependiendo de qué media bobina se energice, se puede tener un polo norte o un polo sur; si se energiza la otra mitad, se obtiene un polo opuesto al otro. En la Figura se muestra un esquema representativo del motor a pasos unipolar.



Conceptual Model of Unipolar Stepper Motor

Figura N°6 Motor de pasos PAP Unipolar

Fuente: <https://www.330ohms.com>

Un motor unipolar de 5 cables es así porque los cables intermedios están unidos en un sólo nodo mientras que el motor unipolar de 6 cables tiene un cable de alimentación para cada par de bobinas, ver las figuras siguientes.

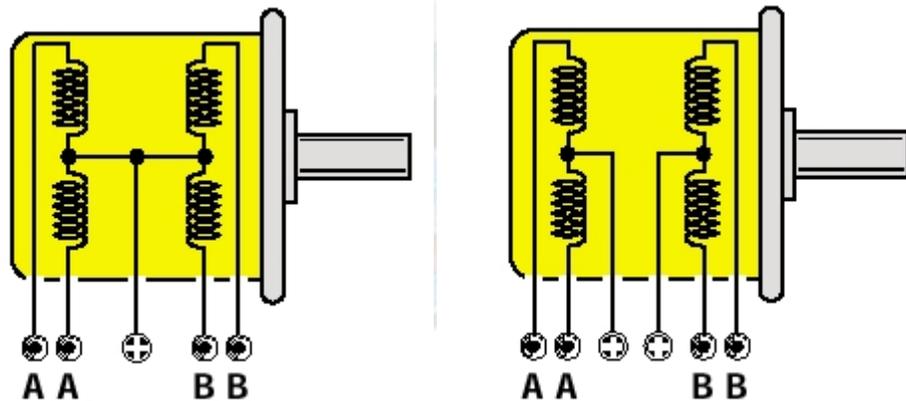
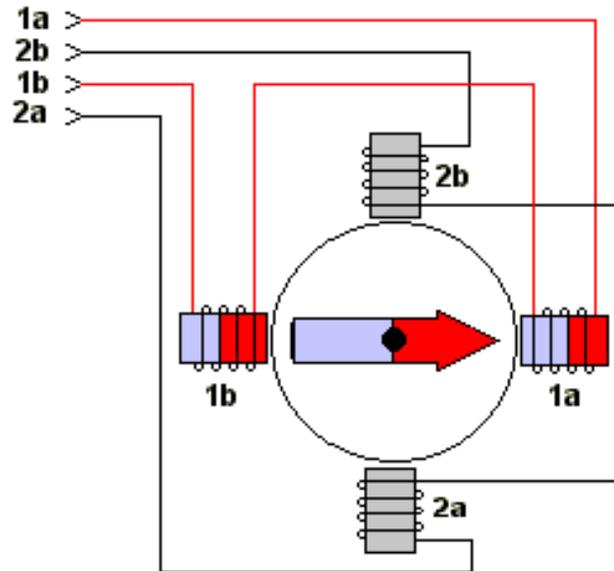


Figura N°7 Motor de pasos PAP Unipolar de 5y 6 hilos

Fuente: <https://www.330ohms.com>

2.6.4.2 MOTORES BIPOLARES

Cuentan con dos bobinas sin ningún punto medio donde salga un cable, por lo que se tienen cuatro cables y cada par corresponde a las terminales de una bobina, ver en la figura N° 9 y N°10. Dada la configuración de la bobina, la corriente puede fluir en dos direcciones, necesitando un control bidireccional o bipolar. En general, con respecto al sentido de giro de los motores a pasos bipolares, vale la pena recordar que el sentido de giro depende de la dirección del flujo de la corriente por las bobinas ya que ésta induce en el embobinado un campo magnético que genera un polo magnético norte y sur, de ahí que el rotor se mueva para que uno de los polos del rotor sea opuesto al de la bobina - localizado en el estator-, como se muestra en la figura N°9.



Conceptual Model of Bipolar Stepper Motor

*Figura N°8 Motor de pasos Bipolar
Fuente: <https://www.330ohms.com>*

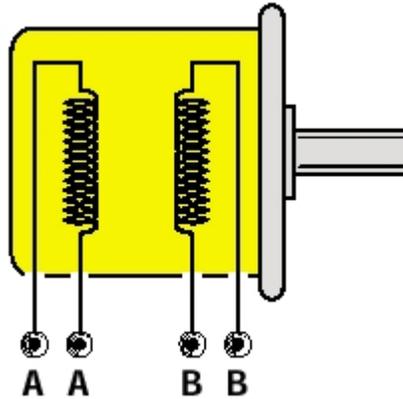


Figura N°9 Motor de pasos Bipolar de 4 hilos
Fuente: <https://www.330ohms.com>

2.7 HARDWARE LIBRE

2.7.1 PLACA DE DESARROLLO ARDUINO UNO

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Arduino Uno es una placa electrónica basada en el microcontrolador ATmega328. Cuenta con 14 entradas/salidas digitales, de las cuales 6 se pueden utilizar como salidas PWM (Modulación por ancho de pulsos) y otras 6 son entradas analógicas. Además, incluye un resonador cerámico de 16 MHz, un conector USB, un conector de alimentación, una cabecera ICSP y un botón de reseteado.



Figura N°10 Placa Arduino Uno R3
Fuente: <http://www.iescamp.es>

2.7.1.1 ESPECIFICACIONES TECNICAS DE ARDUINO UNO

- Microcontrolador ATmega328P
- Tensión de funcionamiento 5V
- Voltaje de entrada (recomendado) 7-12V
- Voltaje de entrada (límite) 6-20V
- Digital pines I/O 14 (de los cuales 6 proporcionan una salida PWM)
- PWM digital pines I/O 6
- Pines de entrada analógica 6
- Corriente DC por Pin I/O 20mA
- Corriente DC para Pin 3.3V 60mA
- Memoria flash 32KB ATmega328P de los que 0,5 KB son utilizados por el gestor de arranque.
- SRAM 2KB ATmega328P
- EEPROM 1KB ATmega328P
- Velocidad de reloj 16 MHz
- Longitud 68,6 mm
- Anchura 53,4 mm
- Peso 25 g

2.7.2 CNC SHIELD

La CNC Shield es una pequeña placa que permite controlar hasta 4 motores paso a paso fácilmente junto a Arduino gracias a su formato shield. Soporta 4 controladores de potencia **Pololu A4988** o **Pololu DRV8825** y dispone de todas las conexiones necesarias para conectar interruptores de final de carrera, salidas de relé y diversos sensores. Es totalmente compatible con el firmware de control GRBL y puede ser utilizada con cualquier modelo de Arduino, aunque se recomienda utilizar un modelo del tipo Arduino UNO o Arduino Leonardo.

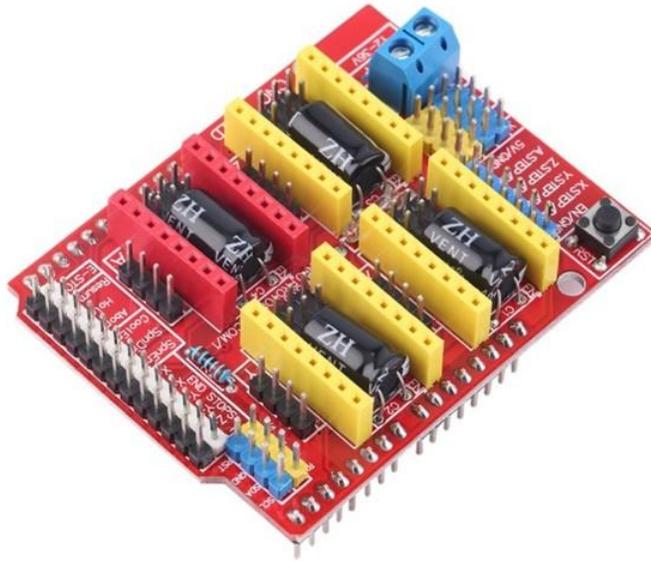


Figura N°11 CNC Shield
Fuente: <http://tienda.bricogeek.com>

2.7.2.1 ESPECIFICACIONES TECNICAS DE LA CNC SHIELD

- Compatible con GRBL 0.8c. (Fimware Open Source para Arduino que convierte G-code a instrucciones para motores PAP)
- Soporte para 4 ejes (X, Y, Z , A)
- 2 conexiones para finales de carrera para cada eje (6 en total)
- Salida "Spindle enable" y "direction"
- Salida "Coolant enable"
- Compatible con Pololu A4988 y DRV8825
- Jumpers para control de micro-stepping (Los controladores como el DRV8825 soportan hasta 1/32 para más precisión)
- Diseño compacto
- Los motores pueden ser conectados con bornes tipo Molex de 4 pines

- Alimentación: 12-36V DC. (Dependiendo de los controladores utilizados)

2.7.3 DRIVERS A4988 POLOLU

El A4988 es un controlador (driver) que simplifican el manejo de motores paso a paso desde un autómatas o procesador como Arduino. Para su control únicamente requieren dos salidas digitales, una para indicar el sentido de giro y otra para comunicar que queremos que el motor avance un paso. Además permiten realizar microstepping, una técnica para conseguir precisiones superiores al paso nominal del motor.

2.8 SOFTWARE LIBRE

El término software libre refiere el conjunto de software (programa informático) que por elección manifiesta de su autor, puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras.

En concreto, se determina que para que un software sea considerado libre es fundamental que le ofrezca al usuario cuatro grandes libertades como son estas:

- Libertad de poder ejecutar el programa en cuestión tal como desee y con el propósito que considere oportuno.
- Libertad de redistribuir las copias que considere útiles para poder “ofrecer” ayuda a las personas de su entorno.
- Libertad para estudiar a fondo el programa, averiguar cómo funciona e incluso llegar a cambiarlo si así lo considera oportuno.
- Libertad no sólo para modificar el software sino también para poder redistribuirlo una vez cambiado, para que así más personas puedan disfrutar del mismo.

2.9 FRESADO CNC

Una fresadora es una máquina que haciendo girar una herramienta de corte con el fin de eliminar material de una pieza de trabajo móvil.

- La herramienta de corte se monta en el husillo y gira a distintas velocidades, en función de las especificaciones de la herramienta y del material
- La pieza se fija a una mesa conocida como carro transversal.
- El carro transversal mueve la pieza de trabajo y la pone en contacto con la herramienta de corte. La herramienta elimina material de la pieza de trabajo en los puntos de contacto, creando piezas terminadas.

En años anteriores, todos los movimientos de la fresadora se realizaban manualmente. Con el desarrollo de la tecnología, las fresadoras más modernas son controladas por computadoras.



Figura N°12 Fresado CNC
Fuente: <http://cz2jncpics.com>

CAPITULO III

3. INGENIERIA DEL PROYECTO

En éste capítulo se describirá detalladamente el funcionamiento del prototipo CNC para circuitos impresos, siguiendo ordenadamente el diagrama de funcionamiento que se muestra a continuación.

3.1 Diagrama en bloques del funcionamiento del prototipo CNC

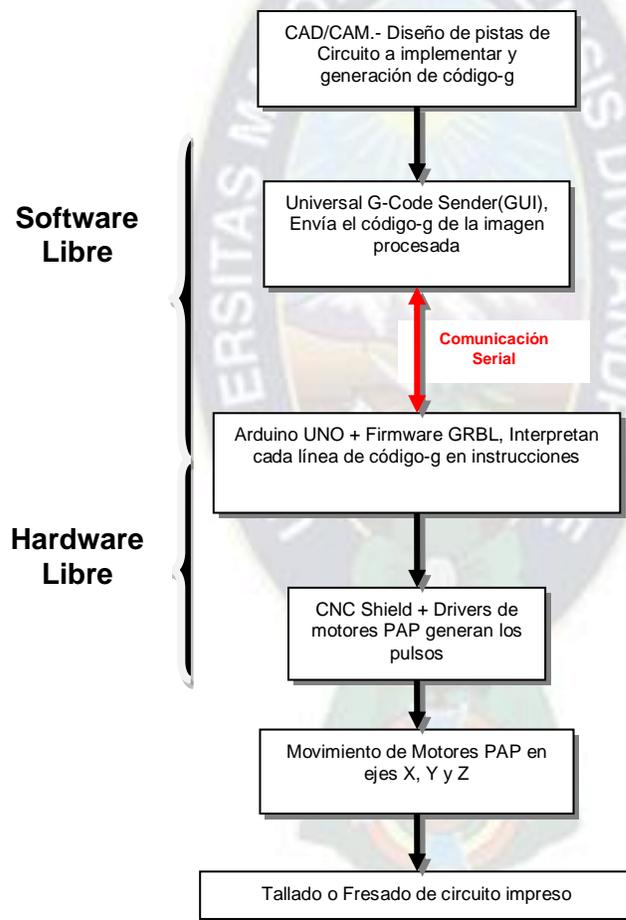


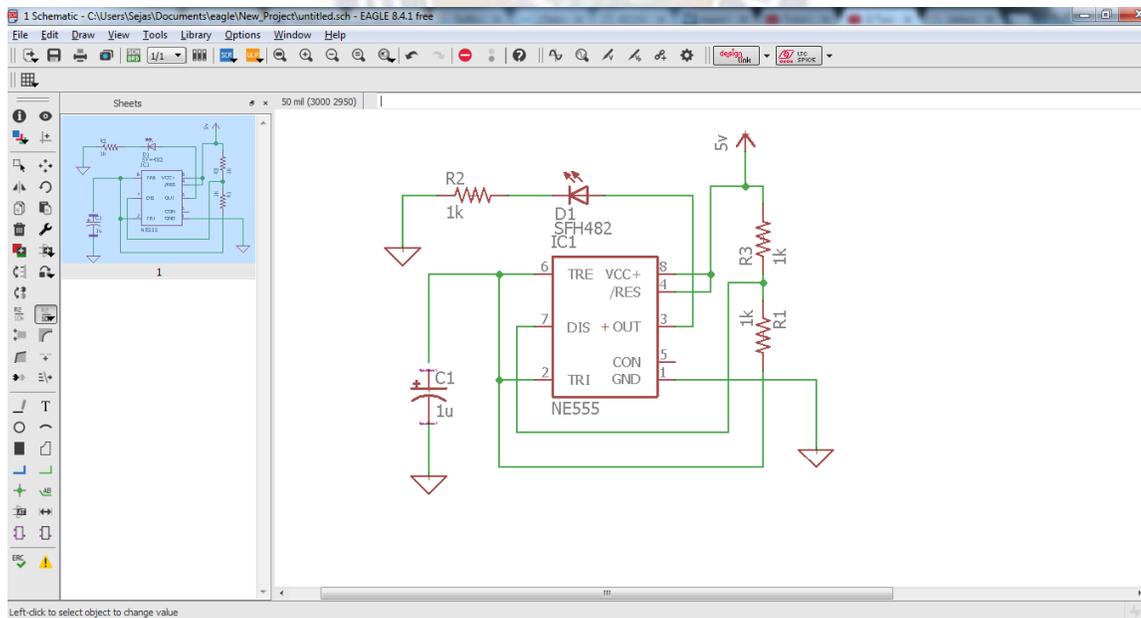
Figura N°13 Diagrama en bloques de prototipo CNC
Fuente: Diseño Propio

3.2 Funcionamiento del Software

En ésta sección se explicara el proceso que se lleva a cabo inicialmente desde un diseño de circuito impreso que se desea obtener, posteriormente procesando dicha imagen en un bloque de código-g, el cual será transmitido serialmente a Arduino UNO, donde se encuentra el firmware GRBL, y cuya función será finalmente traducir dicho código-g en pulsos para los motores PAP.

3.2.1 CAD/CAM Diseño de pistas de circuito a implementar

Partiendo de un diseño circuitual del cual se desea obtener su circuito impreso para ponerlo en funcionamiento, se disponen de muchos programas que realizan éste proceso CAD/CAM, convirtiendo dicho diseño en código-G, pero, para no salir del contexto, se utilizo un software de diseño de circuitos de licencia gratuita llamado Eagle, como se ve a continuación.



*Figura N°14 Diseño de circuito con Eagle
Fuente: Diseño Propio*

Una vez obtenida la imagen de las pistas, como se ve en la figura, se pasa al procesamiento de dicha imagen, para vectorizar las rutas que tomara la fresa, este proceso se realiza también con un Software de licencia gratuita como es *Inkscape*,

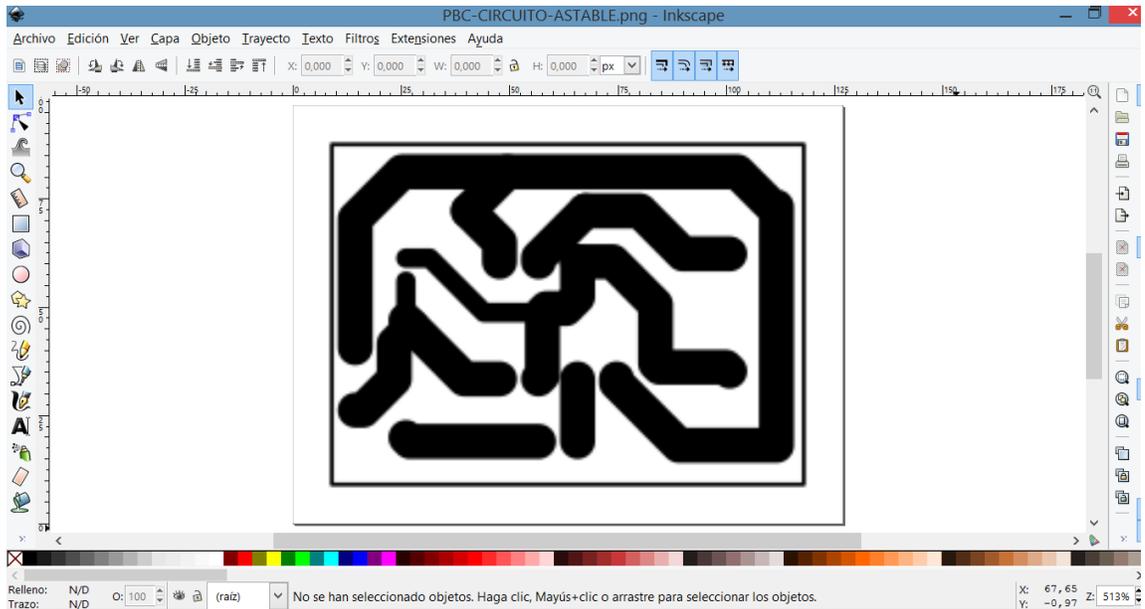


Figura N°15 Pistas para el circuito impreso
Fuente: Diseño Propio

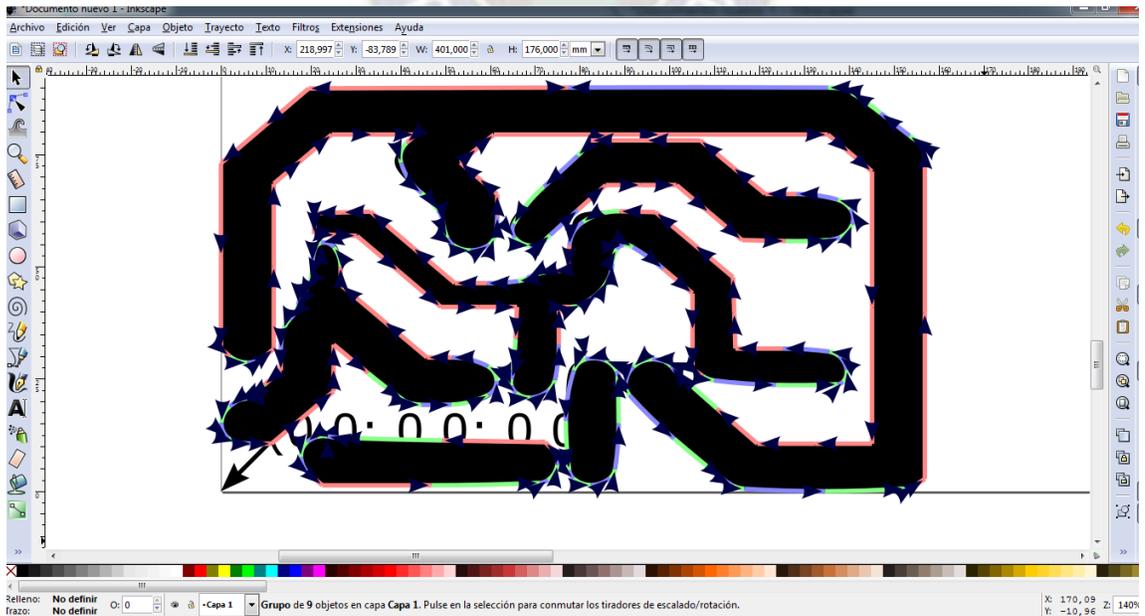


Figura N°16 Vectorización de pistas con Inkscape
Fuente: Diseño Propio

Finalmente como se ve en la figura nos genera un archivo de extensión ".ngc", el cual contiene todos los bloques de código-g para ser enviados a Arduino.

```
pista1_0001.ngc: Bloc de notas
Archivo Edición Formato Ver Ayuda
%
(Header)
(Generated by gcodetools from Inkscape.)
(Using default header. To add your own header create file "header" in the output dir.)
M3
(Header end.)
G21 (All units in mm)

(start cutting path id: path21)
(Change tool to cylindrical cutter)

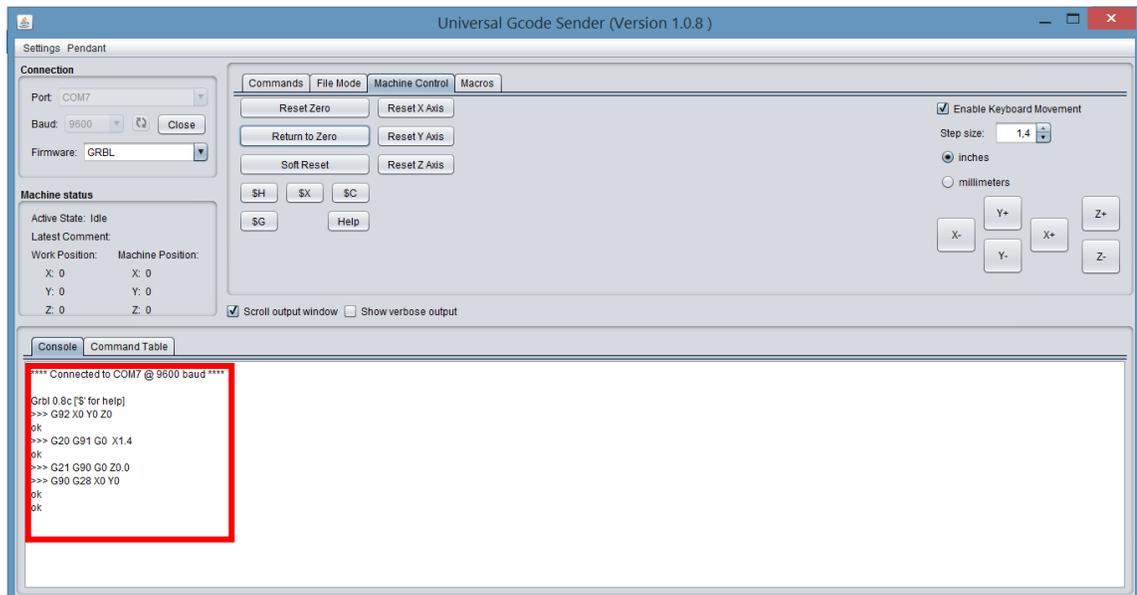
G00 Z5.000000
G00 X21.862029 Y25.389621

G01 Z-1.000000 F100.0(Penetrate)
G01 X5.524087 Y25.332772 Z-1.000000 F100.000000
G01 X2.762044 Y22.981016 Z-1.000000
G01 X0.000000 Y20.629261 Z-1.000000
G01 X0.000000 Y14.793137 Z-1.000000
G01 X0.000000 Y8.957026 Z-1.000000
G01 X0.544793 Y8.497063 Z-1.000000
G03 X1.662206 Y8.088431 Z-1.000000 I1.117412 J1.323478
G03 X2.779618 Y8.497063 Z-1.000000 I-0.000000 J1.732109
G01 X3.321482 Y8.957026 Z-1.000000
G01 X3.321482 Y14.150178 Z-1.000000
G01 X3.321482 Y19.343331 Z-1.000000
G01 X5.143318 Y20.888921 Z-1.000000
G01 X6.968083 Y22.434499 Z-1.000000
G01 X9.615895 Y22.434499 Z-1.000000
G01 X12.263707 Y22.434499 Z-1.000000
G01 X11.692554 Y21.920092 Z-1.000000
G03 X11.335481 Y20.654579 Z-1.000000 I0.826249 J-0.916263
G03 X12.623975 Y18.759691 Z-1.000000 I3.600247 J1.062589
G02 X13.682726 Y17.645883 Z-1.000000 I-3.224644 J-4.125357
G02 X14.006461 Y16.793703 Z-1.000000 I-1.508681 J-1.060716
G03 X14.255394 Y15.905500 Z-1.000000 I2.877584 J0.327503
G03 X14.612764 Y15.483058 Z-1.000000 I0.944988 J0.437041
G03 X15.724064 Y15.143927 Z-1.000000 I1.097142 J1.604854
```

*Figura N°17 Archivo contenedor de los codigos-g
Fuente: Diseño Propio*

3.2.1.1 Arquitectura de un bloque de código-g

Los bloques de código-g tienen 3 partes importantes, una es el número de bloque "NXX", otra es "GXX" el cual especifica qué tipo de movimiento se realizará de acuerdo al lenguaje, por ejemplo un arco "G2" o una recta, etc., y la tercera sección indica la distancia en milímetros que se va recorrer en cualquiera de los ejes, como se muestra en la figura siguiente:



*Figura N°19 Universal G-Code Sender
Fuente: Diseño Propio*

3.2.3 Firmware GRBL para Arduino UNO

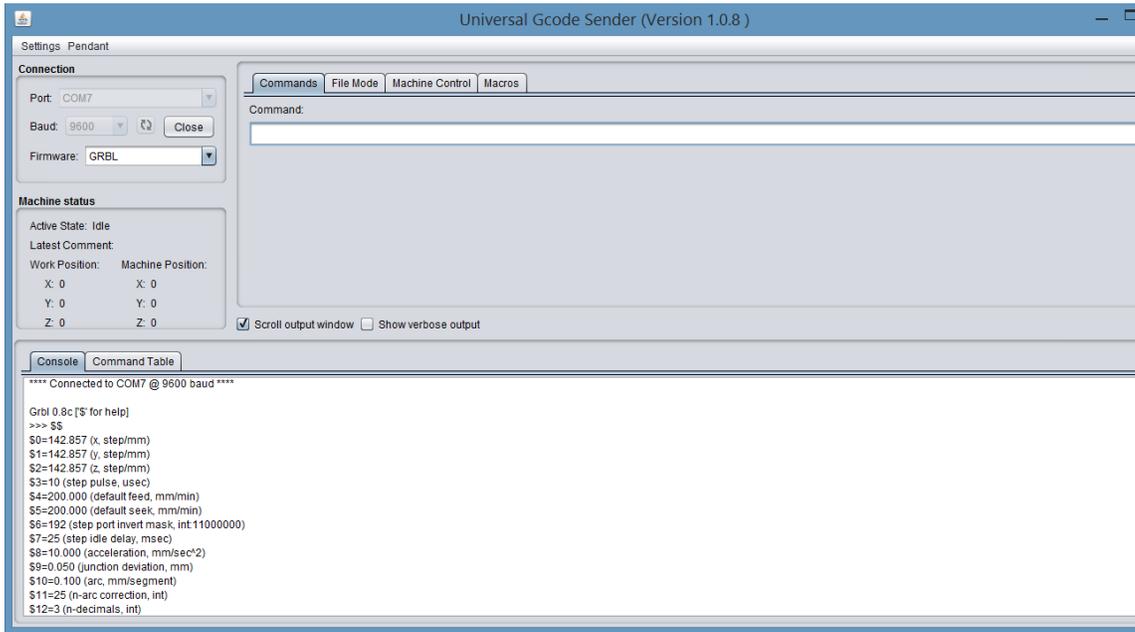
El funcionamiento de éste firmware es principalmente el de recibir los códigos-g mediante el puerto serial, para interpretarlos en instrucciones que se transmitirán hacia los motores paso a paso. Los códigos-g enviados por el Universal G-code Sender llegan en bloques, recibiendo un máximo 16 bloques por parte de Arduino Uno. Los bloques se van almacenando en un buffer para luego enviarlos a los motores.

El programa principal de GRBL se encarga de recibir dichos bloques y enviar un mensaje una vez procesado, ya sea un "ok" en caso de un procesamiento exitoso del bloque o "error" en caso de haberse suscitado una falla en el proceso.

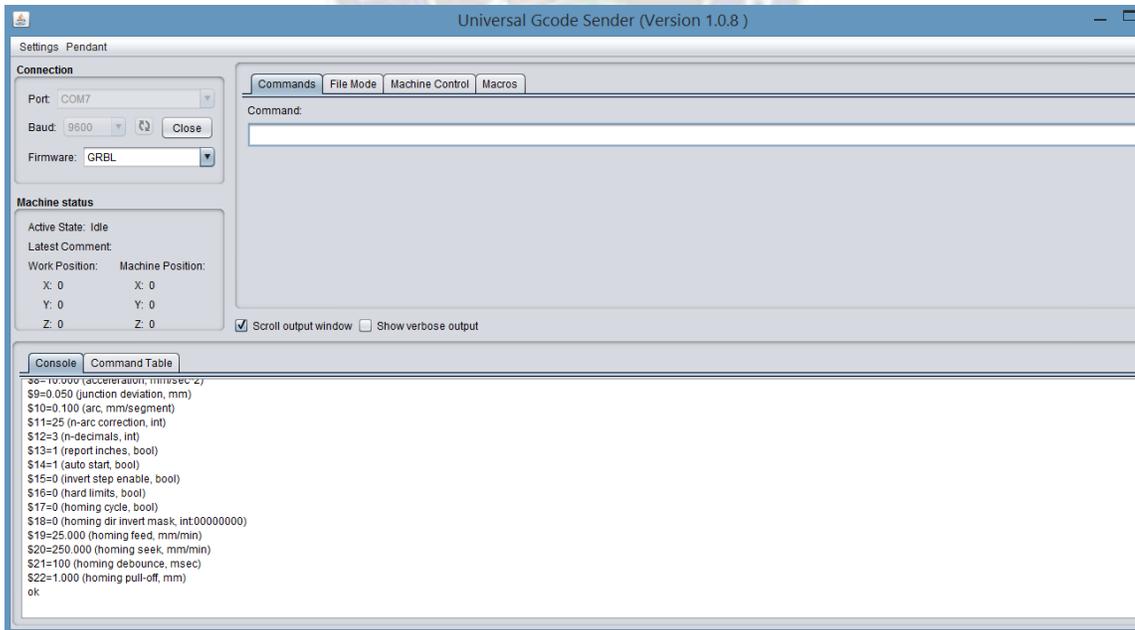
Por otra parte, también actúa en segundo plano un programa de interrupciones, el cual está encargado de controlar los motores, enviando impulsos de paso y bits de dirección a los drivers A4988.

Nota.- El código fuente de GRBL, se adjunta en anexos debido a su larga extensión.

También mencionar que, GRBL ofrece un menú de configuración de los motores, como se muestra en la figura siguiente:



*Figura N°20 Parámetros de GRBL mediante Universal G-Code Sender
Fuente: Diseño Propio*



*Figura N°21 Parámetros de GRBL mediante Universal G-Code Sender
Fuente: Diseño Propio*

3.2.3.1 Diagrama en bloques del GRBL

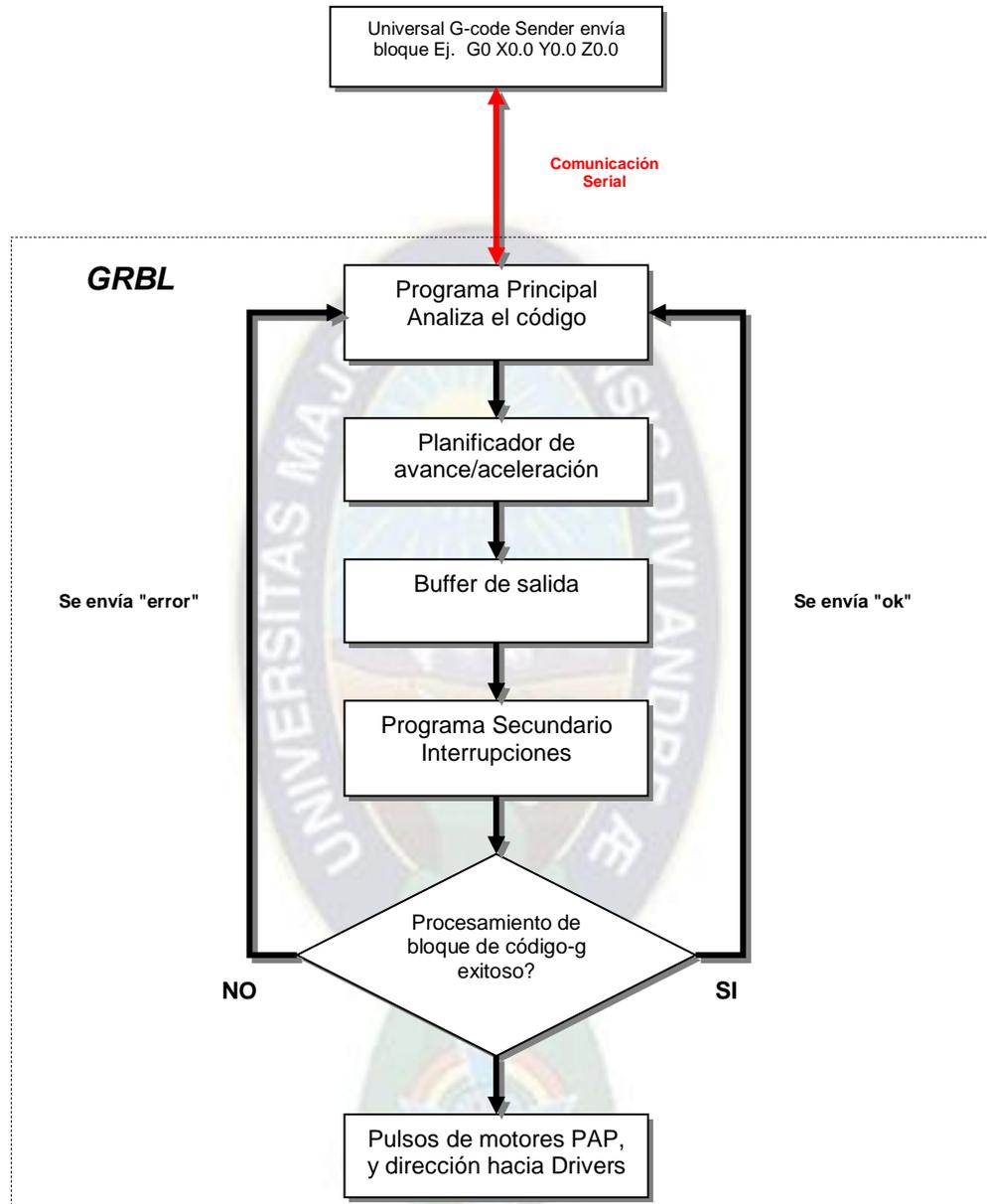


Figura N°22 Diagrama en bloques de GRBL
Fuente: Diseño Propio - <https://onehosshay.wordpress.com>

3.2.3.2 Ejemplo de interpretación de código-g en bloques del GRBL

Un ejemplo de la trayectoria que interpretaría el GRBL, realizando un movimiento que describa un cuadrado de 10mm, regresando a su origen, sería el siguiente código:

G90 G71	(cotas absolutas referidas al punto 0,0; Programación en mm)
G00 X0.0 Y0.0	(posicionamiento rápido lineal al punto 0,0 del plano XY)
G01 X10.0	(movimiento lineal de 10mm en la dirección X positiva)
G01 Y10.0	(movimiento lineal de 10mm en la dirección Y positiva)
G01 X0.0	(movimiento lineal de 10mm en la dirección X negativa)
G01 Y0.0	(movimiento lineal de 10mm en la dirección Y negativa)

Dando como resultado el siguiente movimiento:

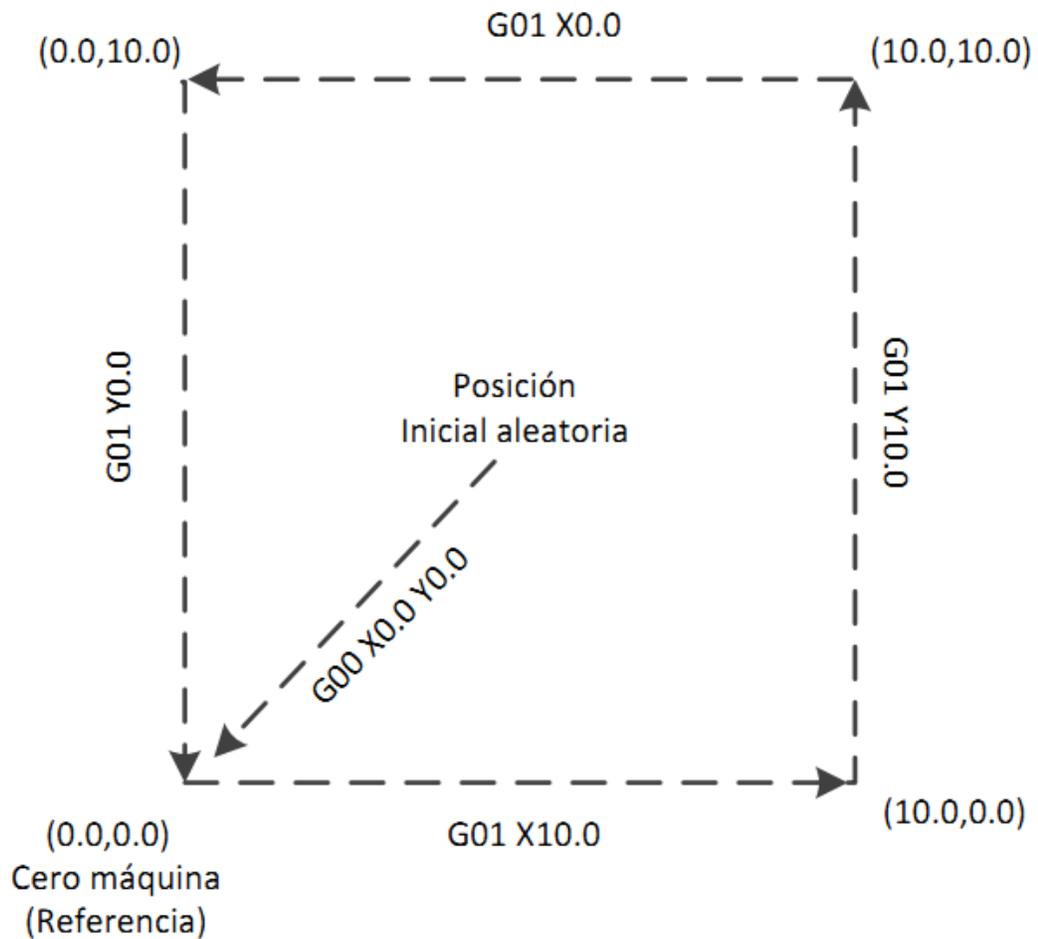


Figura N°23 Movimiento interpretado por GRBL
Fuente: <http://wiki.ead.pucv.cl>

3.3 Funcionamiento del Hardware

3.3.1 Arduino Uno

La placa Arduino Uno, cumple la función de recibir el código-g desde la computadora mediante comunicación serial, y enviar dichas instrucciones a los motores de paso PAP, enviando pulsos utilizando sus puertos de salida PWM en los pines 2,3,4,5,6 y 7, conectando dos pines por eje, uno para envío de dirección, y otro para envío del pulso de paso, ya sea para los ejes X , Y o Z.

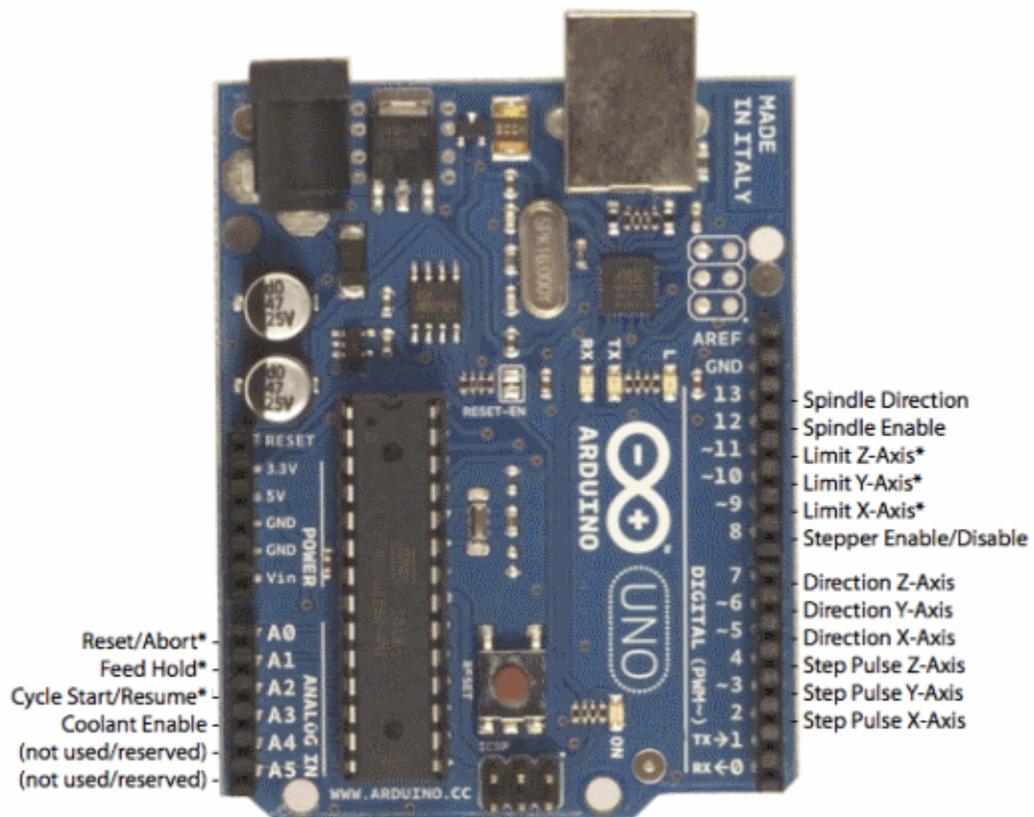


Figura N°24 Conexión pines de Ejes X,Y,Z hacia drivers

Fuente: <https://blog.protoneer.co.nz>

3.3.2 CNC Shield

La CNC Shield ofrece una inmediata conexión entre Arduino Uno y los Drivers A4988, simplificando y abaratando costos. También dando la opción de configurar los pasos de los motores mediante hardware.

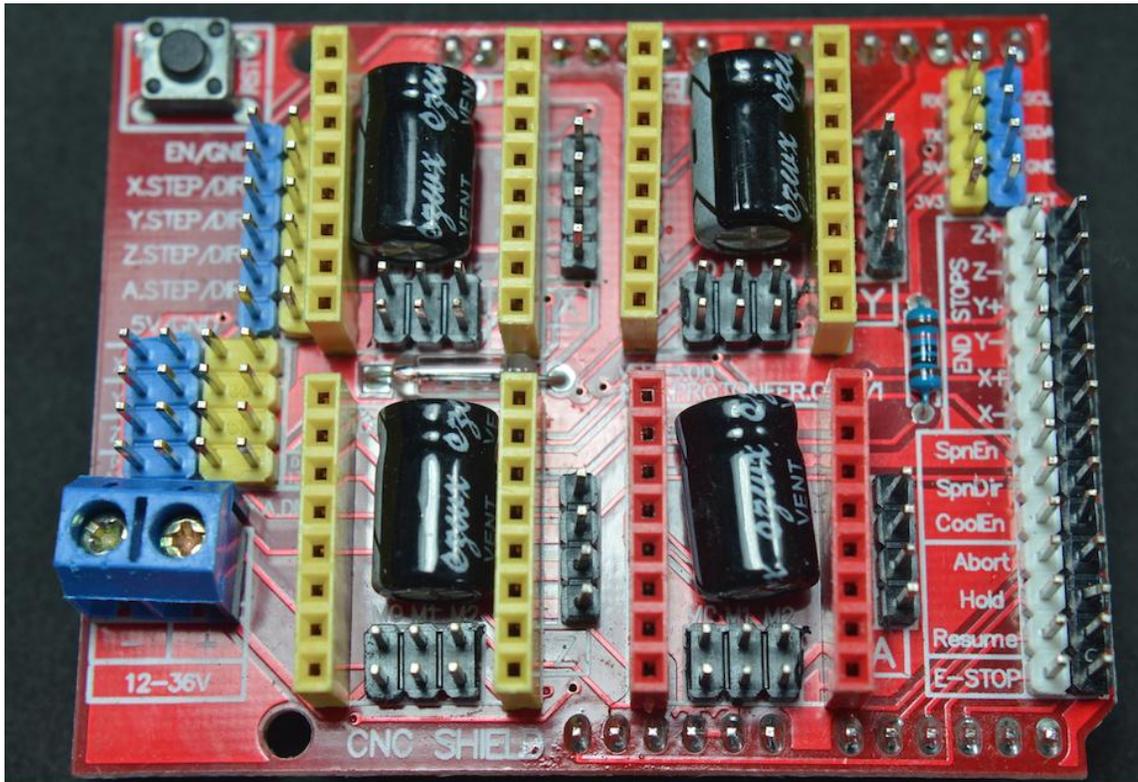


Figura N°25 CNC Shield
Fuente: <http://dinastiatecnologica.com>

3.3.3 Drivers A4988 Pololu

Estos controladores son los encargados de limitar la corriente que fluirá hacia los motores PAP, como también enviar los pulsos desde Arduino. Como se ve en la figura siguiente, los pines que van hacia el microcontrolador, y que se encargan de enviar los pulsos a los motores son:

Typical Application Diagram

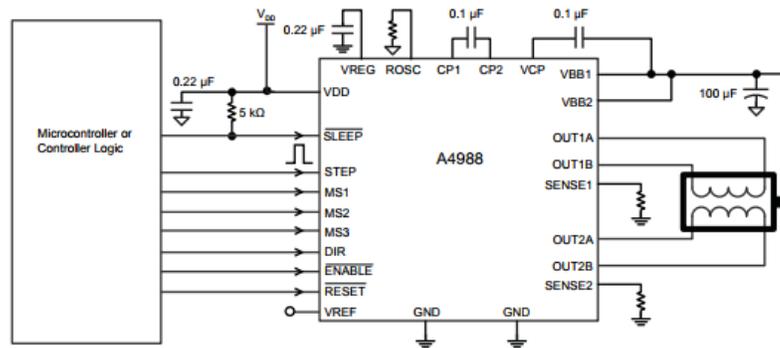


Figura N°26 Diagrama de pines Driver A4988

Fuente: www.alldatasheet.com

- **STEP.**- Es donde Arduino enviará los pulsos para los motores de paso, cabe mencionar que las transiciones se harán en flanco de subida, es decir cuando haya un cambio de estado de BAJO-ALTO.
- **MS1,MS2 y MS3.**- Son pines de resolución de MicroStep, micro-pasos, es decir en qué modo trabajaran, en éste caso para el funcionamiento optimo del prototipo CNC, evitando así perder fuerza, ni pulsos en el funcionamiento, se optó por la configuración en Full-Step, dejando éstos pines en BAJO.

Table 1. Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Figura N°27 Tabla de resolución de micro pasos

Fuente: www.alldatasheet.com

- **DIR.**- Configura el sentido del giro del motor, haciendo efecto el cambio cuando haya un flanco de subida en la entrada.

- **ENABLE**.- Para habilitar o deshabilitar las salidas de los FET, funciona en estado BAJO.
- **RESET**.- En estado BAJO, deshabilita todas las salidas de los FET, también todas las entradas de pasos STEP, se ignoran.

3.3.3.1 Configuración de corriente de Driver A4988

Como los motores a utilizar son de diferentes fabricantes, o diferentes características, es necesario calcular el voltaje de referencia V_{ref} para la corriente que el driver enviara para el funcionamiento del motor. Según el datasheet del fabricante, se debe hacer uso de una ecuación que se muestra a continuación:

The maximum value of current limiting is set by the selection of R_{Sx} and the voltage at the VREF pin. The transconductance function is approximated by the maximum value of current limiting, $I_{TripMAX}$ (A), which is set by

$$I_{TripMAX} = V_{REF} / (8 \times R_S)$$

where R_S is the resistance of the sense resistor (Ω) and V_{REF} is the input voltage on the REF pin (V).

Figura N°28 Calculo de corriente máxima en driver A4988

Fuente: www.alldatasheet.com

La corriente $I_{tripMAX}$ es aquella que soporta el motor paso a paso a configurar, por tanto es un dato que se debe conocer del fabricante. La resistencia de sensibilidad se obtiene observando el valor directamente del impreso como se ve en la figura:

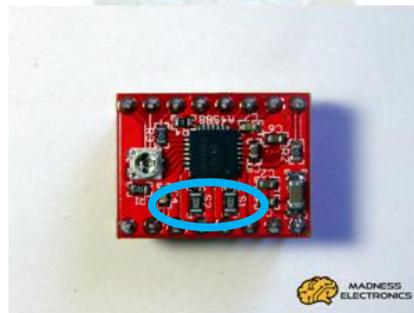


Figura N°29 Resistencia de sensibilidad R_s

Fuente: <http://www.madnesselectronics.com>

Otro parámetro a observar del datasheet es, el porcentaje de corriente que utilizara, de acuerdo a la configuración que se haya elegido , en éste caso para un optimo funcionamiento del prototipo, se optó por la configuración FULL-STEP, que ofrece un máximo de 70% como se ve a continuación, también se puede observar que en otras configuraciones, ya sea 1/2 de paso, 1/4, 1/8, 1/16 éste porcentaje de uso de corriente puede ir disminuir considerablemente, y haciendo el funcionamiento del motor más propenso a errores.

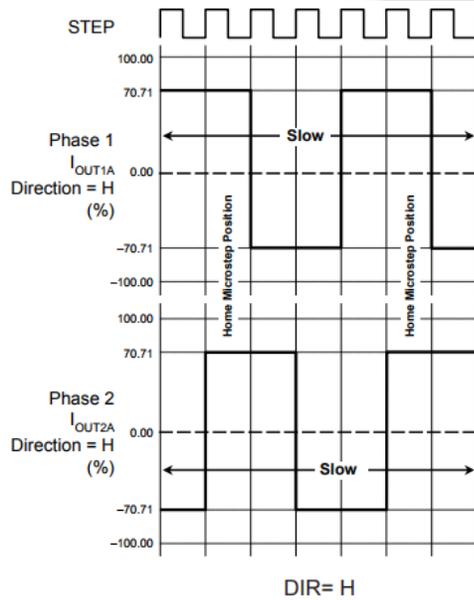


Figure 8. Decay Mode for Full-Step Increments

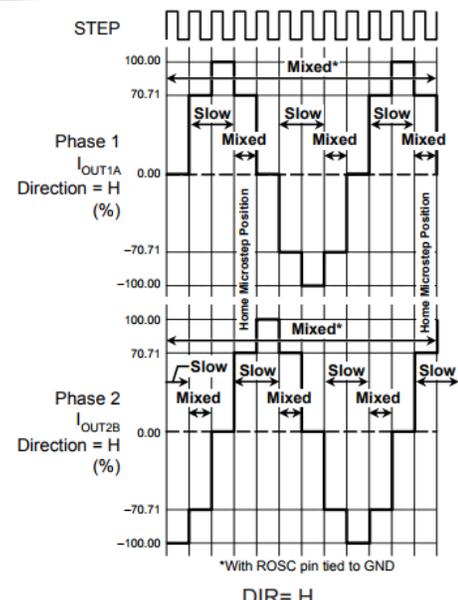


Figure 9. Decay Modes for Half-Step Increments

Figura N°30 Modo configuración pasos completos y medios pasos

Fuente: www.alldatasheet.com

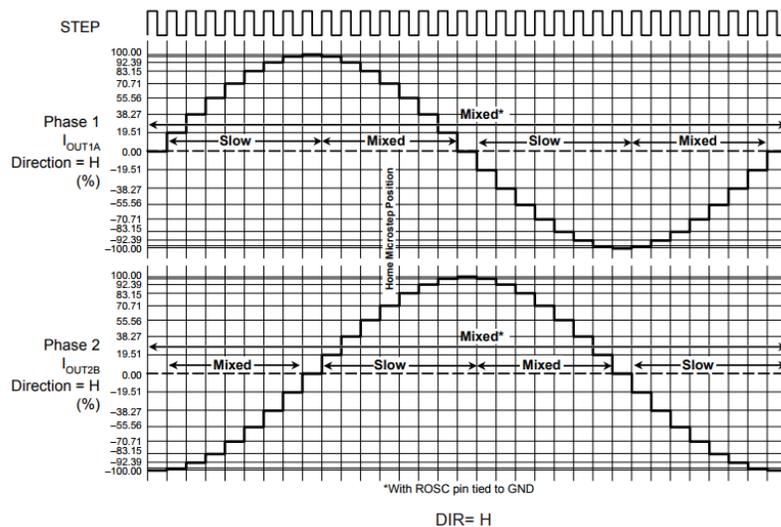


Figura N°31 Modo configuración 1/8 de paso

Fuente: www.alldatasheet.com

Por tanto, siguiendo estos parámetros anteriormente mencionados, se calcula el voltaje de referencia necesitado, que finalmente se configura directamente en el potenciómetro que se encuentra en el driver, por ejemplo para una $R_s=0.1\Omega$, $I_{trapMax}=0.4A$:

$$I_{trapMax} = \frac{V_{ref}}{8 \times R_s}$$

$$V_{ref} = I_{trapMax}(8 \times R_s)$$

$$V_{ref} = 0.4(8 \times 0.1)$$

$$V_{ref} = 0.4(8 \times 0.1)$$

$$V_{ref} = 0.32$$

Pero como se tiene una configuración FULL-STEP, la cual nos indica una utilización del 70%, se debe calcular ese porcentaje del V_{ref} .

$$V_{ref}(70\%) = 0.32 \times 0.7$$

$$V_{ref}(70\%) = 0.22$$

Dicho V_{ref} calculado al 70% se debe regular en el potenciómetro que se muestra a continuación:

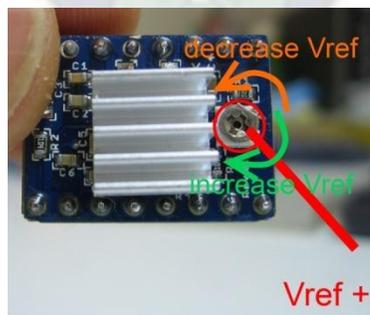


Figura N°32 Regulador Voltaje de Referencia del Driver

Fuente: <http://www.trustfm.net>

3.4 Diagrama de Conexiones del prototipo CNC.-

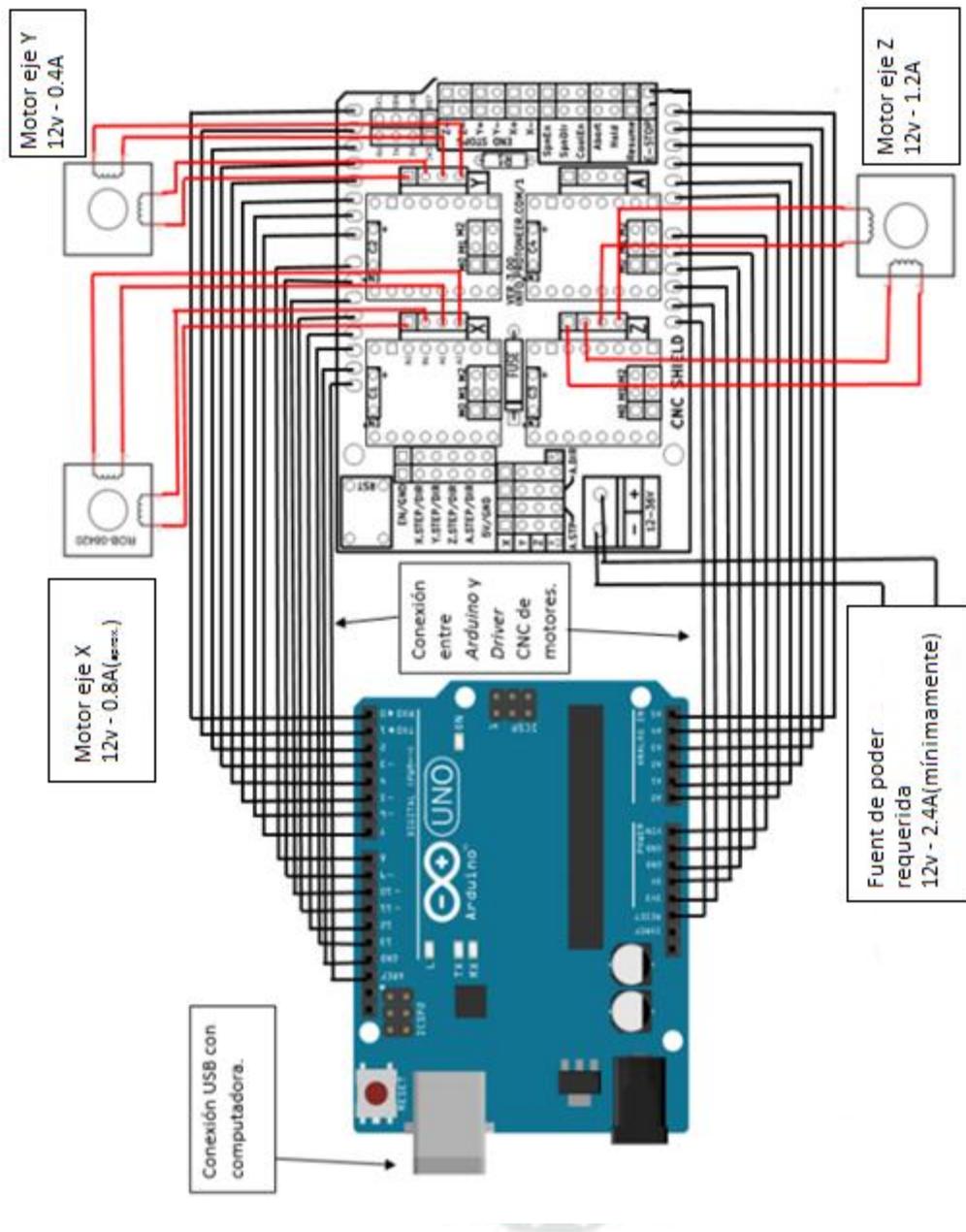
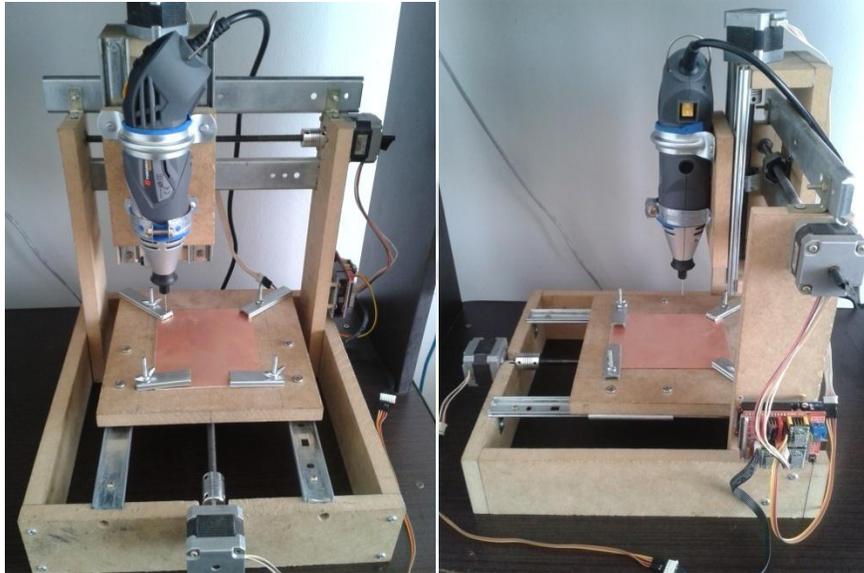


Figura N°33 Diagrama de conexiones del prototipo CNC
Fuente: Trabajo de grado Univ. Tecnológica de Pereira

3.5 Estructura mecánica del prototipo



*Figura N°34 Estructura mecánica
Fuente: Diseño propio*

3.5.1 Rieles para los Ejes.-

Éstas rieles sirven para el movimiento mecánico de los ejes X, Y y Z, impulsados por los motores PAP.



*Figura N°35 Rieles para el movimiento de los Ejes
Fuente: mercadolibre.com.mx*

3.5.2 Tornillo sin fin.-

Este tornillo cumple la función de impulsar los 3 ejes individualmente, es conectado un extremo al motor PAP mediante un acople flexible.



Figura N°36 Tornillo sin fin
Fuente: <http://www.directindustry.es>

3.5.3 Acoples Flexibles

Cumplen la función de unir el eje de los motores, con el tornillo sin fin, para así generar una rotación del tornillo, y consecuentemente movimiento en los ejes.



Figura N°37 Acople Flexible
Fuente: <http://www.naylampmechatronics.com>

3.5.4 Accesorio para grabado en PCB

La fresa a utilizar debe ser una con punta en V, al menos con un ángulo de 30 o 10° si es posible como se ve en la figura, para así poder realizar circuitos con acabado fino y totalmente funcional.

Por otro lado, también es posible utilizar una broca común, de un diametro adecuado para un tallado funcional.



Figura N°38 Fresa o broca para tallado
Fuente: <https://listado.mercadolibre.com.mx>

CAPITULO IV

4. COSTOS

A continuación, se detallan los costos que involucraron la puesta en funcionamiento del prototipo CNC.

4.1 Costos fijos

N°	Material	Costo(Bs)
1	Arduino Uno R3	40
2	CNC Shield	50
3	Drivers A4988 Pololu(x3)	60
4	Motores paso a paso PAP(x3)	30
5	Rieles para mecanizado(x5)	75
6	Madera MDF	30
7	Acoples Flexibles(x3)	60
8	Cable USB impresora	10
9	Cables de Protoboard	8
10	Drill Pequeño de 170w	150
11	Broca para tallado de PCB	5
12	Tornillo sin fin	7
13	Tuerca+Abrazadera(x3)	20
	TOTAL.-	545 Bs.-

5. CONCLUSIONES

- Al concluir en ensamblado y puesta en funcionamiento de la CNC, se logra obtener una alternativa, de fácil implementación, de bajo costo, mas practica, mucho menos riesgosa para la salud de los estudiantes y más ecológica.
- Se aplico conocimientos de electrónica digital para la configuración y manipulación de los drivers, como así también para el cargado de software en Arduino Uno.
- Se construyo el prototipo CNC, tratando de minimizar los costos, recurriendo a la adquisición de los materiales en Ferias de la ciudad.
- Este es un proyecto base, de mecanizado y movimiento en 3 ejes, el cual puede ser fácilmente útil para la implementación de una impresora 3D, con los conocimientos y experiencias obtenidas en el presente proyecto.

6. RECOMENDACIONES

- Para la reducción de costos, es muy recomendable la adquisición de los motores PAP reciclados en Ferias de la ciudad, donde existen un sin fin de los mismos, claro está que, la mayoría no tiene garantía, ni modelos, por ende es muy difícil encontrar información de los datasheet. Pero por otro lado, es posible hacer procedimientos de prueba-error para deducir las características de los motores, como grados/paso, corriente, etc.
- Para la estructura, se utilizo rieles recicladas, siendo las mismas muy económicas y estables para el mecanizado.

7. BIBLIOGRAFIA

- Michael Felipe Cifuentes Molano - Jeison Steven Jaramillo Blandón (2015), DISEÑO DE UN SISTEMA DE MANUFACTURA AUTOMÁTICO PARA CIRCUITOS IMPRESOS, Trabajo de Grado, UNIVERSIDAD TECNOLÓGICA DE PEREIRA.
- Ivan Camilo Garcia Mutis - Juan Gabriel Lagos Lopez - Luis Fernando Urrego Perez - Peter Yesid Delgado Parra (2009), Diseño e Implementación de un control CNC para crear modelos y esculturas en tercera dimensión a partir de un diseño CAD , Trabajo de Grado, UNIVERSIDAD DE SAN BUENAVENTURA.
- <http://elprofegarcia.com/?tag=cnc>
- [http://wiki.ead.pucv.cl/Introducci%C3%B3n_al_control_num%C3%A9rico_computarizado_\(CNC\)](http://wiki.ead.pucv.cl/Introducci%C3%B3n_al_control_num%C3%A9rico_computarizado_(CNC))
- <https://latinoamerica.autodesk.com/solutions/cad-cam>
- <https://lenguajedeingenieria.files.wordpress.com/2013/02/introduccc3b3n-al-cad-cam.pdf>
- [http://wiki.ead.pucv.cl/Introducci%C3%B3n_al_control_num%C3%A9rico_computarizado_\(CNC\)#Sistemas_CAD.2C_CAM_y_c.C3.B3digo_G](http://wiki.ead.pucv.cl/Introducci%C3%B3n_al_control_num%C3%A9rico_computarizado_(CNC)#Sistemas_CAD.2C_CAM_y_c.C3.B3digo_G)

8. ANEXOS

A continuación se adjuntara, las partes más importantes del código fuente del Firmware GRBL, debido a que es un código demasiado extenso, sin embargo se puede acceder a todo el código fuente del autor mediante la dirección web:

<https://github.com/Protoneer/GRBL-Arduino-Library>.

Programa Principal grblmail.cpp

```
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "config.h"
#include "planner.h"
#include "nuts_bolts.h"
#include "stepper.h"
#include "spindle_control.h"
#include "coolant_control.h"
#include "motion_control.h"
#include "gcode.h"
#include "protocol.h"
#include "limits.h"
#include "report.h"
#include "settings.h"
#include "serial.h"

// Declare system global variable structure
system_t sys;

int startGrbl(void)
{
    // Initialize system
    serial_init(); // Setup serial baud rate and interrupts
    settings_init(); // Load grbl settings from EEPROM
    st_init(); // Setup stepper pins and interrupt timers
    sei(); // Enable interrupts

    memset(&sys, 0, sizeof(sys)); // Clear all system variables
    sys.abort = true; // Set abort to complete initialization
    sys.state = STATE_INIT; // Set alarm state to indicate unknown initial position

    for(;;) {

        // Execute system reset upon a system abort, where the main program will return to this loop.
        // Once here, it is safe to re-initialize the system. At startup, the system will automatically
        // reset to finish the initialization process.
        if (sys.abort) {
            // Reset system.
            serial_reset_read_buffer(); // Clear serial read buffer
            plan_init(); // Clear block buffer and planner variables
            gc_init(); // Set g-code parser to default state
            protocol_init(); // Clear incoming line data and execute startup lines
            spindle_init();
            coolant_init();
            limits_init();
            st_reset(); // Clear stepper subsystem variables.

            // Sync cleared gcode and planner positions to current system position, which is only
            // cleared upon startup, not a reset/abort.
            sys_sync_current_position();

            // Reset system variables.
            sys.abort = false;
            sys.execute = 0;
            if (bit_istrue(settings.flags,BITFLAG_AUTO_START)) { sys.auto_start = true; }

            // Check for power-up and set system alarm if homing is enabled to force homing cycle
            // by setting Grbl's alarm state. Alarm locks out all g-code commands, including the
            // startup scripts, but allows access to settings and internal commands. Only a homing
            // cycle '$H' or kill alarm locks '$X' will disable the alarm.
            // NOTE: The startup script will run after successful completion of the homing cycle, but
            // not after disabling the alarm locks. Prevents motion startup blocks from crashing into
            // things uncontrollably. Very bad.
            #ifndef HOMING_INIT_LOCK
            if (sys.state == STATE_INIT && bit_istrue(settings.flags,BITFLAG_HOMING_ENABLE)) { sys.state = STATE_ALARM; }
            #endif

            // Check for and report alarm state after a reset, error, or an initial power up.
            if (sys.state == STATE_ALARM) {
                report_feedback_message(MESSAGE_ALARM_LOCK);
            } else {
                // All systems go. Set system to ready and execute startup script.
                sys.state = STATE_IDLE;
                protocol_execute_startup();
            }
        }
        protocol_execute_runtime();
        protocol_process(); // ... process the serial protocol
    }
    return 0; /* never reached */
}
```

```
}
```

Rutina gcode.cpp

```
#include "gcode.h"
#include <string.h>
#include "nuts_bolts.h"
#include <math.h>
#include "settings.h"
#include "motion_control.h"
#include "spindle_control.h"
#include "coolant_control.h"
#include "errno.h"
#include "protocol.h"
#include "report.h"

// Declare gc extern struct
parser_state_t gc;

#define FAIL(status) gc.status_code = status;

static int next_statement(char *letter, float *float_ptr, char *line, uint8_t *char_counter);

static void select_plane(uint8_t axis_0, uint8_t axis_1, uint8_t axis_2)
{
    gc.plane_axis_0 = axis_0;
    gc.plane_axis_1 = axis_1;
    gc.plane_axis_2 = axis_2;
}

void gc_init()
{
    memset(&gc, 0, sizeof(gc));
    gc.feed_rate = settings.default_feed_rate; // Should be zero at initialization.
    // gc.seek_rate = settings.default_seek_rate;
    select_plane(X_AXIS, Y_AXIS, Z_AXIS);
    gc.absolute_mode = true;

    // Load default G54 coordinate system.
    if (!(settings_read_coord_data(gc.coord_select, gc.coord_system))) {
        report_status_message(STATUS_SETTING_READ_FAIL);
    }
}

// Sets g-code parser position in mm. Input in steps. Called by the system abort and hard
// limit pull-off routines.
void gc_set_current_position(int32_t x, int32_t y, int32_t z)
{
    gc.position[X_AXIS] = x/settings.steps_per_mm[X_AXIS];
    gc.position[Y_AXIS] = y/settings.steps_per_mm[Y_AXIS];
    gc.position[Z_AXIS] = z/settings.steps_per_mm[Z_AXIS];
}

static float to_millimeters(float value)
{
    return(gc.inches_mode ? (value * MM_PER_INCH) : value);
}

// Executes one line of 0-terminated G-Code. The line is assumed to contain only uppercase
// characters and signed floating point values (no whitespace). Comments and block delete
// characters have been removed. All units and positions are converted and exported to grbl's
// internal functions in terms of (mm, mm/min) and absolute machine coordinates, respectively.
uint8_t gc_execute_line(char *line)
{
    // If in alarm state, don't process. Immediately return with error.
    // NOTE: Might not be right place for this, but also prevents $N storing during alarm.
    if (sys.state == STATE_ALARM) { return(STATUS_ALARM_LOCK); }

    uint8_t char_counter = 0;
    char letter;
    float value;
    int int_value;

    uint16_t modal_group_words = 0; // Bitflag variable to track and check modal group words in block
    uint8_t axis_words = 0; // Bitflag to track which XYZ(ABC) parameters exist in block

    float inverse_feed_rate = -1; // negative inverse_feed_rate means no inverse_feed_rate specified
    uint8_t absolute_override = false; // true(1) = absolute motion for this block only {G53}
    uint8_t non_modal_action = NON_MODAL_NONE; // Tracks the actions of modal group 0 (non-modal)

    float target[3], offset[3];
    clear_vector(target); // XYZ(ABC) axes parameters.
    clear_vector(offset); // IJK Arc offsets are incremental. Value of zero indicates no change.
```

```

gc.status_code = STATUS_OK;

/* Pass 1: Commands and set all modes. Check for modal group violations.
NOTE: Modal group numbers are defined in Table 4 of NIST RS274-NGC v3, pg.20 */
uint8_t group_number = MODAL_GROUP_NONE;
while(next_statement(&letter, &value, line, &char_counter)) {
    int_value = trunc(value);
    switch(letter) {
        case 'G':
            // Set modal group values
            switch(int_value) {
                case 4: case 10: case 28: case 30: case 53: case 92: group_number = MODAL_GROUP_0; break;
                case 0: case 1: case 2: case 3: case 80: group_number = MODAL_GROUP_1; break;
                case 17: case 18: case 19: group_number = MODAL_GROUP_2; break;
                case 90: case 91: group_number = MODAL_GROUP_3; break;
                case 93: case 94: group_number = MODAL_GROUP_5; break;
                case 20: case 21: group_number = MODAL_GROUP_6; break;
                case 54: case 55: case 56: case 57: case 58: case 59: group_number = MODAL_GROUP_12; break;
            }
            // Set 'G' commands
            switch(int_value) {
                case 0: gc.motion_mode = MOTION_MODE_SEEK; break;
                case 1: gc.motion_mode = MOTION_MODE_LINEAR; break;
                case 2: gc.motion_mode = MOTION_MODE_CW_ARC; break;
                case 3: gc.motion_mode = MOTION_MODE_CCW_ARC; break;
                case 4: non_modal_action = NON_MODAL_DWELL; break;
                case 10: non_modal_action = NON_MODAL_SET_COORDINATE_DATA; break;
                case 17: select_plane(X_AXIS, Y_AXIS, Z_AXIS); break;
                case 18: select_plane(X_AXIS, Z_AXIS, Y_AXIS); break;
                case 19: select_plane(Y_AXIS, Z_AXIS, X_AXIS); break;
                case 20: gc.inches_mode = true; break;
                case 21: gc.inches_mode = false; break;
                case 28: case 30:
                    int_value = trunc(10*value); // Multiply by 10 to pick up Gxx.1
                    switch(int_value) {
                        case 280: non_modal_action = NON_MODAL_GO_HOME_0; break;
                        case 281: non_modal_action = NON_MODAL_SET_HOME_0; break;
                        case 300: non_modal_action = NON_MODAL_GO_HOME_1; break;
                        case 301: non_modal_action = NON_MODAL_SET_HOME_1; break;
                        default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
                    }
                    break;
                case 53: absolute_override = true; break;
                case 54: case 55: case 56: case 57: case 58: case 59:
                    gc.coord_select = int_value-54;
                    break;
                case 80: gc.motion_mode = MOTION_MODE_CANCEL; break;
                case 90: gc.absolute_mode = true; break;
                case 91: gc.absolute_mode = false; break;
                case 92:
                    int_value = trunc(10*value); // Multiply by 10 to pick up G92.1
                    switch(int_value) {
                        case 920: non_modal_action = NON_MODAL_SET_COORDINATE_OFFSET; break;
                        case 921: non_modal_action = NON_MODAL_RESET_COORDINATE_OFFSET; break;
                        default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
                    }
                    break;
                case 93: gc.inverse_feed_rate_mode = true; break;
                case 94: gc.inverse_feed_rate_mode = false; break;
                default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
            }
            break;
        case 'M':
            // Set modal group values
            switch(int_value) {
                case 0: case 1: case 2: case 30: group_number = MODAL_GROUP_4; break;
                case 3: case 4: case 5: group_number = MODAL_GROUP_7; break;
            }
            // Set 'M' commands
            switch(int_value) {
                case 0: gc.program_flow = PROGRAM_FLOW_PAUSED; break; // Program pause
                case 1: break; // Optional stop not supported. Ignore.
                case 2: case 30: gc.program_flow = PROGRAM_FLOW_COMPLETED; break; // Program end and reset
                case 3: gc.spindle_direction = 1; break;
                case 4: gc.spindle_direction = -1; break;
                case 5: gc.spindle_direction = 0; break;
                #ifdef ENABLE_M7
                case 7: gc.coolant_mode = COOLANT_MIST_ENABLE; break;
                #endif
                case 8: gc.coolant_mode = COOLANT_FLOOD_ENABLE; break;
                case 9: gc.coolant_mode = COOLANT_DISABLE; break;
                default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
            }
    }
}

```

```

    break;
}
// Check for modal group multiple command violations in the current block
if (group_number) {
    if ( bit_istrue(modal_group_words,bit(group_number)) ) {
        FAIL(STATUS_MODAL_GROUP_VIOLATION);
    } else {
        bit_true(modal_group_words,bit(group_number));
    }
    group_number = MODAL_GROUP_NONE; // Reset for next command.
}
}

// If there were any errors parsing this line, we will return right away with the bad news
if (gc.status_code) { return(gc.status_code); }

/* Pass 2: Parameters. All units converted according to current block commands. Position
parameters are converted and flagged to indicate a change. These can have multiple connotations
for different commands. Each will be converted to their proper value upon execution. */
float p = 0, r = 0;
uint8_t l = 0;
char_counter = 0;
while(next_statement(&letter, &value, line, &char_counter)) {
    switch(letter) {
        case 'G': case 'M': case 'N': break; // Ignore command statements and line numbers
        case 'F':
            if (value <= 0) { FAIL(STATUS_INVALID_STATEMENT); } // Must be greater than zero
            if (gc.inverse_feed_rate_mode) {
                inverse_feed_rate = to_millimeters(value); // seconds per motion for this motion only
            } else {
                gc.feed_rate = to_millimeters(value); // millimeters per minute
            }
            break;
        case 'I': case 'J': case 'K': offset[letter-'I'] = to_millimeters(value); break;
        case 'L': l = trunc(value); break;
        case 'P': p = value; break;
        case 'R': r = to_millimeters(value); break;
        case 'S':
            if (value < 0) { FAIL(STATUS_INVALID_STATEMENT); } // Cannot be negative
            // TBD: Spindle speed not supported due to PWM issues, but may come back once resolved.
            // gc.spindle_speed = value;
            break;
        case 'T':
            if (value < 0) { FAIL(STATUS_INVALID_STATEMENT); } // Cannot be negative
            gc.tool = trunc(value);
            break;
        case 'X': target[X_AXIS] = to_millimeters(value); bit_true(axis_words,bit(X_AXIS)); break;
        case 'Y': target[Y_AXIS] = to_millimeters(value); bit_true(axis_words,bit(Y_AXIS)); break;
        case 'Z': target[Z_AXIS] = to_millimeters(value); bit_true(axis_words,bit(Z_AXIS)); break;
        default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
    }
}

// If there were any errors parsing this line, we will return right away with the bad news
if (gc.status_code) { return(gc.status_code); }

/* Execute Commands: Perform by order of execution defined in NIST RS274-NGC.v3, Table 8, pg.41.
NOTE: Independent non-motion/settings parameters are set out of this order for code efficiency
and simplicity purposes, but this should not affect proper g-code execution. */

// ([F]: Set feed and seek rates.)
// TODO: Seek rates can change depending on the direction and maximum speeds of each axes. When
// max axis speed is installed, the calculation can be performed here, or maybe in the planner.

if (sys.state != STATE_CHECK_MODE) {
    // ([M6]: Tool change should be executed here.)

    // [M3,M4,M5]: Update spindle state
    spindle_run(gc.spindle_direction);

    // [*M7,M8,M9]: Update coolant state
    coolant_run(gc.coolant_mode);
}

// [G54,G55,...,G59]: Coordinate system selection
if ( bit_istrue(modal_group_words,bit(MODAL_GROUP_12)) ) { // Check if called in block
    float coord_data[N_AXIS];
    if (!(settings_read_coord_data(gc.coord_select,coord_data))) { return(STATUS_SETTING_READ_FAIL); }
    memcpy(gc.coord_system,coord_data,sizeof(coord_data));
}

// [G4,G10,G28,G30,G92,G92.1]: Perform dwell, set coordinate system data, homing, or set axis offsets.
// NOTE: These commands are in the same modal group, hence are mutually exclusive. G53 is in this

```

```

// modal group and do not effect these actions.
uint8_t home_select;
switch (non_modal_action) {
  case NON_MODAL_DWELL:
    if (p < 0) { // Time cannot be negative.
      FAIL(STATUS_INVALID_STATEMENT);
    } else {
      // Ignore dwell in check gcode modes
      if (sys.state != STATE_CHECK_MODE) { mc_dwell(p); }
    }
    break;
  case NON_MODAL_SET_COORDINATE_DATA:
    int_value = trunc(p); // Convert p value to int.
    if ((l != 2 && l != 20) || (int_value < 1 || int_value > N_COORDINATE_SYSTEM)) { // L2 and L20. P1=G54, P2=G55, ...
      FAIL(STATUS_UNSUPPORTED_STATEMENT);
    } else if (!axis_words && l==2) { // No axis words.
      FAIL(STATUS_INVALID_STATEMENT);
    } else {
      int_value--; // Adjust P index to EEPROM coordinate data indexing.
      if (l == 20) {
        settings_write_coord_data(int_value,gc.position);
        // Update system coordinate system if currently active.
        if (gc.coord_select == int_value) { memcpy(gc.coord_system,gc.position,sizeof(gc.position)); }
      } else {
        float coord_data[N_AXIS];
        if (!settings_read_coord_data(int_value,coord_data)) { return(STATUS_SETTING_READ_FAIL); }
        // Update axes defined only in block. Always in machine coordinates. Can change non-active system.
        uint8_t i;
        for (i=0; i<N_AXIS; i++) { // Axes indices are consistent, so loop may be used.
          if ( bit_istrue(axis_words,bit(i)) ) { coord_data[i] = target[i]; }
        }
        settings_write_coord_data(int_value,coord_data);
        // Update system coordinate system if currently active.
        if (gc.coord_select == int_value) { memcpy(gc.coord_system,coord_data,sizeof(coord_data)); }
      }
    }
    axis_words = 0; // Axis words used. Lock out from motion modes by clearing flags.
    break;
  case NON_MODAL_GO_HOME_0: case NON_MODAL_GO_HOME_1:
    // Move to intermediate position before going home. Obeys current coordinate system and offsets
    // and absolute and incremental modes.
    if (axis_words) {
      // Apply absolute mode coordinate offsets or incremental mode offsets.
      uint8_t i;
      for (i=0; i<N_AXIS; i++) { // Axes indices are consistent, so loop may be used.
        if ( bit_istrue(axis_words,bit(i)) ) {
          if (gc.absolute_mode) {
            target[i] += gc.coord_system[i] + gc.coord_offset[i];
          } else {
            target[i] += gc.position[i];
          }
        } else {
          target[i] = gc.position[i];
        }
      }
    }
    mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS], settings.default_seek_rate, false);
  }
  // Retrieve G28/30 go-home position data (in machine coordinates) from EEPROM
  float coord_data[N_AXIS];
  home_select = SETTING_INDEX_G28;
  if (non_modal_action == NON_MODAL_GO_HOME_1) { home_select = SETTING_INDEX_G30; }
  if (!settings_read_coord_data(home_select,coord_data)) { return(STATUS_SETTING_READ_FAIL); }
  mc_line(coord_data[X_AXIS], coord_data[Y_AXIS], coord_data[Z_AXIS], settings.default_seek_rate, false);
  memcpy(gc.position, coord_data, sizeof(coord_data)); // gc.position[] = coord_data[];
  axis_words = 0; // Axis words used. Lock out from motion modes by clearing flags.
  break;
  case NON_MODAL_SET_HOME_0: case NON_MODAL_SET_HOME_1:
    home_select = SETTING_INDEX_G28;
    if (non_modal_action == NON_MODAL_SET_HOME_1) { home_select = SETTING_INDEX_G30; }
    settings_write_coord_data(home_select,gc.position);
    break;
  case NON_MODAL_SET_COORDINATE_OFFSET:
    if (!axis_words) { // No axis words
      FAIL(STATUS_INVALID_STATEMENT);
    } else {
      // Update axes defined only in block. Offsets current system to defined value. Does not update when
      // active coordinate system is selected, but is still active unless G92.1 disables it.
      uint8_t i;
      for (i=0; i<=2; i++) { // Axes indices are consistent, so loop may be used.
        if (bit_istrue(axis_words,bit(i)) ) {
          gc.coord_offset[i] = gc.position[i]-gc.coord_system[i]-target[i];
        }
      }
    }
  }
}

```

```

axis_words = 0; // Axis words used. Lock out from motion modes by clearing flags.
break;
case NON_MODAL_RESET_COORDINATE_OFFSET:
clear_vector(gc.coord_offset); // Disable G92 offsets by zeroing offset vector.
break;
}

// [G0,G1,G2,G3,G80]: Perform motion modes.
// NOTE: Commands G10,G28,G30,G92 lock out and prevent axis words from use in motion modes.
// Enter motion modes only if there are axis words or a motion mode command word in the block.
if ( bit_istrue(modal_group_words,bit(MODAL_GROUP_1)) || axis_words ) {

// G1,G2,G3 require F word in inverse time mode.
if ( gc.inverse_feed_rate_mode ) {
if (inverse_feed_rate < 0 && gc.motion_mode != MOTION_MODE_CANCEL) {
FAIL(STATUS_INVALID_STATEMENT);
}
}
// Absolute override G53 only valid with G0 and G1 active.
if ( absolute_override && !(gc.motion_mode == MOTION_MODE_SEEK || gc.motion_mode == MOTION_MODE_LINEAR)) {
FAIL(STATUS_INVALID_STATEMENT);
}
// Report any errors.
if (gc.status_code) { return(gc.status_code); }

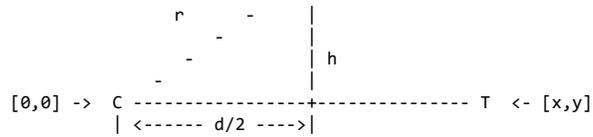
// Convert all target position data to machine coordinates for executing motion. Apply
// absolute mode coordinate offsets or incremental mode offsets.
// NOTE: Tool offsets may be appended to these conversions when/if this feature is added.
uint8_t i;
for (i=0; i<=2; i++) { // Axes indices are consistent, so loop may be used to save flash space.
if ( bit_istrue(axis_words,bit(i)) ) {
if (!absolute_override) { // Do not update target in absolute override mode
if (gc.absolute_mode) {
target[i] += gc.coord_system[i] + gc.coord_offset[i]; // Absolute mode
} else {
target[i] += gc.position[i]; // Incremental mode
}
}
} else {
target[i] = gc.position[i]; // No axis word in block. Keep same axis position.
}
}

switch (gc.motion_mode) {
case MOTION_MODE_CANCEL:
if (axis_words) { FAIL(STATUS_INVALID_STATEMENT); } // No axis words allowed while active.
break;
case MOTION_MODE_SEEK:
if (!axis_words) { FAIL(STATUS_INVALID_STATEMENT); }
else { mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS], settings.default_seek_rate, false); }
break;
case MOTION_MODE_LINEAR:
// TODO: Inverse time requires F-word with each statement. Need to do a check. Also need
// to check for initial F-word upon startup. Maybe just set to zero upon initialization
// and after an inverse time move and then check for non-zero feed rate each time. This
// should be efficient and effective.
if (!axis_words) { FAIL(STATUS_INVALID_STATEMENT); }
else { mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS],
(gc.inverse_feed_rate_mode) ? inverse_feed_rate : gc.feed_rate, gc.inverse_feed_rate_mode); }
break;
case MOTION_MODE_CW_ARC: case MOTION_MODE_CCW_ARC:
// Check if at least one of the axes of the selected plane has been specified. If in center
// format arc mode, also check for at least one of the IJK axes of the selected plane was sent.
if ( !( bit_false(axis_words,bit(gc.plane_axis_2)) ) ||
( !r && !offset[gc.plane_axis_0] && !offset[gc.plane_axis_1] ) ) {
FAIL(STATUS_INVALID_STATEMENT);
} else {
if (r != 0) { // Arc Radius Mode
/*
We need to calculate the center of the circle that has the designated radius and passes
through both the current position and the target position. This method calculates the following
set of equations where [x,y] is the vector from current to target position, d == magnitude of
that vector, h == hypotenuse of the triangle formed by the radius of the circle, the distance to
the center of the travel vector. A vector perpendicular to the travel vector [-y,x] is scaled to the
length of h [-y/d*h, x/d*h] and added to the center of the travel vector [x/2,y/2] to form the new point
[i,j] at [x/2-y/d*h, y/2+x/d*h] which will be the center of our arc.

d^2 == x^2 + y^2
h^2 == r^2 - (d/2)^2
i == x/2 - y/d*h
j == y/2 + x/d*h

0 <- [i,j]
- |

```



C - Current position
T - Target position
O - center of circle that pass through both C and T
d - distance from C to T
r - designated radius
h - distance from center of CT to O

Expanding the equations:

```
d -> sqrt(x^2 + y^2)
h -> sqrt(4 * r^2 - x^2 - y^2)/2
i -> (x - (y * sqrt(4 * r^2 - x^2 - y^2)) / sqrt(x^2 + y^2)) / 2
j -> (y + (x * sqrt(4 * r^2 - x^2 - y^2)) / sqrt(x^2 + y^2)) / 2
```

Which can be written:

```
i -> (x - (y * sqrt(4 * r^2 - x^2 - y^2))/sqrt(x^2 + y^2))/2
j -> (y + (x * sqrt(4 * r^2 - x^2 - y^2))/sqrt(x^2 + y^2))/2
```

Which we for size and speed reasons optimize to:

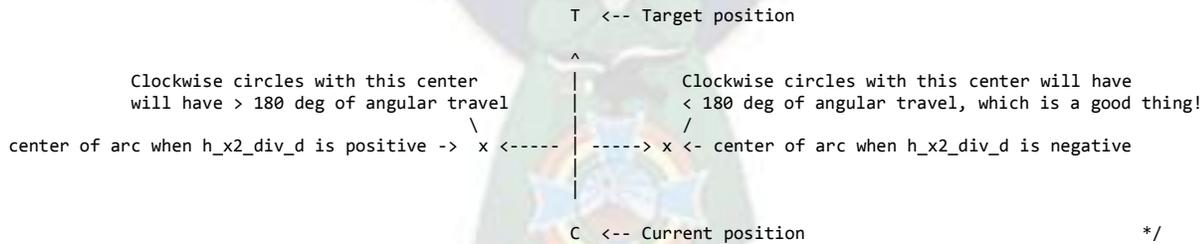
```
h_x2_div_d = sqrt(4 * r^2 - x^2 - y^2)/sqrt(x^2 + y^2)
i = (x - (y * h_x2_div_d))/2
j = (y + (x * h_x2_div_d))/2
```

*/

```
// Calculate the change in position along each selected axis
float x = target[gc.plane_axis_0]-gc.position[gc.plane_axis_0];
float y = target[gc.plane_axis_1]-gc.position[gc.plane_axis_1];
```

```
clear_vector(offset);
// First, use h_x2_div_d to compute 4*h^2 to check if it is negative or r is smaller
// than d. If so, the sqrt of a negative number is complex and error out.
float h_x2_div_d = 4 * r*r - x*x - y*y;
if (h_x2_div_d < 0) { FAIL(STATUS_ARC_RADIUS_ERROR); return(gc.status_code); }
// Finish computing h_x2_div_d.
h_x2_div_d = -sqrt(h_x2_div_d)/hypot(x,y); // == -(h * 2 / d)
// Invert the sign of h_x2_div_d if the circle is counter clockwise (see sketch below)
if (gc.motion_mode == MOTION_MODE_CCW_ARC) { h_x2_div_d = -h_x2_div_d; }
```

/* The counter clockwise circle lies to the left of the target direction. When offset is positive, the left hand circle will be generated - when it is negative the right hand circle is generated.



```
/* Negative R is g-code-alese for "I want a circle with more than 180 degrees of travel" (go figure!),
// even though it is advised against ever generating such circles in a single line of g-code. By
// inverting the sign of h_x2_div_d the center of the circles is placed on the opposite side of the line of
// travel and thus we get the unadvisably long arcs as prescribed.
```

```
if (r < 0) {
    h_x2_div_d = -h_x2_div_d;
    r = -r; // Finished with r. Set to positive for mc_arc
}
```

```
// Complete the operation by calculating the actual center of the arc
offset[gc.plane_axis_0] = 0.5*(x-(y*h_x2_div_d));
offset[gc.plane_axis_1] = 0.5*(y+(x*h_x2_div_d));
```

```
} else { // Arc Center Format Offset Mode
    r = hypot(offset[gc.plane_axis_0], offset[gc.plane_axis_1]); // Compute arc radius for mc_arc
}
```

```
// Set clockwise/counter-clockwise sign for mc_arc computations
uint8_t isclockwise = false;
if (gc.motion_mode == MOTION_MODE_CW_ARC) { isclockwise = true; }
```

```

    // Trace the arc
    mc_arc(gc.position, target, offset, gc.plane_axis_0, gc.plane_axis_1, gc.plane_axis_2,
          (gc.inverse_feed_rate_mode) ? inverse_feed_rate : gc.feed_rate, gc.inverse_feed_rate_mode,
          r, isclockwise);
    }
    break;
}

// Report any errors.
if (gc.status_code) { return(gc.status_code); }

// As far as the parser is concerned, the position is now == target. In reality the
// motion control system might still be processing the action and the real tool position
// in any intermediate location.
memcpy(gc.position, target, sizeof(target)); // gc.position[] = target[];
}

// M0,M1,M2,M30: Perform non-running program flow actions. During a program pause, the buffer may
// refill and can only be resumed by the cycle start run-time command.
if (gc.program_flow) {
    plan_synchronize(); // Finish all remaining buffered motions. Program paused when complete.
    sys.auto_start = false; // Disable auto cycle start. Forces pause until cycle start issued.

    // If complete, reset to reload defaults (G92.2,G54,G17,G90,G94,M48,G40,M5,M9). Otherwise,
    // re-enable program flow after pause complete, where cycle start will resume the program.
    if (gc.program_flow == PROGRAM_FLOW_COMPLETED) { mc_reset(); }
    else { gc.program_flow = PROGRAM_FLOW_RUNNING; }
}

return(gc.status_code);
}

// Parses the next statement and leaves the counter on the first character following
// the statement. Returns 1 if there was a statements, 0 if end of string was reached
// or there was an error (check state.status_code).
static int next_statement(char *letter, float *float_ptr, char *line, uint8_t *char_counter)
{
    if (line[*char_counter] == 0) {
        return(0); // No more statements
    }

    *letter = line[*char_counter];
    if((*letter < 'A') || (*letter > 'Z')) {
        FAIL(STATUS_EXPECTED_COMMAND_LETTER);
        return(0);
    }
    (*char_counter)++;
    if (!read_float(line, char_counter, float_ptr)) {
        FAIL(STATUS_BAD_NUMBER_FORMAT);
        return(0);
    };
    return(1);
}
}

```



```

    step_pulse_time = -(((settings.pulse_microseconds-2)*TICKS_PER_MICROSECOND) >> 3);
#endif
// Enable stepper driver interrupt
TIMSK1 |= (1<<OCIE1A);
}
}

// Stepper shutdown
void st_go_idle()
{
// Disable stepper driver interrupt
TIMSK1 &= ~(1<<OCIE1A);
// Disable steppers only upon system alarm activated or by user setting to not be kept enabled.
if ((settings.stepper_idle_lock_time != 0xff) || bit_istrue(sys.execute,EXEC_ALARM)) {
// Force stepper dwell to lock axes for a defined amount of time to ensure the axes come to a complete
// stop and not drift from residual inertial forces at the end of the last movement.
delay_ms(settings.stepper_idle_lock_time);
if (bit_istrue(settings.flags,BITFLAG_INVERT_ST_ENABLE)) {
STEPPERS_DISABLE_PORT &= ~(1<<STEPPERS_DISABLE_BIT);
} else {
STEPPERS_DISABLE_PORT |= (1<<STEPPERS_DISABLE_BIT);
}
}
}

// This function determines an acceleration velocity change every CYCLES_PER_ACCELERATION_TICK by
// keeping track of the number of elapsed cycles during a de/ac-celeration. The code assumes that
// step_events occur significantly more often than the acceleration velocity iterations.
inline static uint8_t iterate_trapezoid_cycle_counter()
{
st.trapezoid_tick_cycle_counter += st.cycles_per_step_event;
if(st.trapezoid_tick_cycle_counter > CYCLES_PER_ACCELERATION_TICK) {
st.trapezoid_tick_cycle_counter -= CYCLES_PER_ACCELERATION_TICK;
return(true);
} else {
return(false);
}
}

// "The Stepper Driver Interrupt" - This timer interrupt is the workhorse of Grbl. It is executed at the rate set with
// config_step_timer. It pops blocks from the block_buffer and executes them by pulsing the stepper pins appropriately.
// It is supported by The Stepper Port Reset Interrupt which it uses to reset the stepper port after each pulse.
// The bresenham line tracer algorithm controls all three stepper outputs simultaneously with these two interrupts.
ISR(TIMER1_COMPA_vect)
{
if (busy) { return; } // The busy-flag is used to avoid reentering this interrupt

// Set the direction pins a couple of nanoseconds before we step the steppers
STEPPING_PORT = (STEPPING_PORT & ~DIRECTION_MASK) | (out_bits & DIRECTION_MASK);
// Then pulse the stepping pins
#ifdef STEP_PULSE_DELAY
step_bits = (STEPPING_PORT & ~STEP_MASK) | out_bits; // Store out_bits to prevent overwriting.
#else // Normal operation
STEPPING_PORT = (STEPPING_PORT & ~STEP_MASK) | out_bits;
#endif
// Enable step pulse reset timer so that The Stepper Port Reset Interrupt can reset the signal after
// exactly settings.pulse_microseconds microseconds, independent of the main Timer1 prescaler.
TCNT2 = step_pulse_time; // Reload timer counter
TCCR2B = (1<<CS21); // Begin timer2. Full speed, 1/8 prescaler

busy = true;
// Re-enable interrupts to allow ISR_TIMER2_OVERFLOW to trigger on-time and allow serial communications
// regardless of time in this handler. The following code prepares the stepper driver for the next
// step interrupt compare and will always finish before returning to the main program.
sei();

// If there is no current block, attempt to pop one from the buffer
if (current_block == NULL) {
// Anything in the buffer? If so, initialize next motion.
current_block = plan_get_current_block();
if (current_block != NULL) {
if (sys.state == STATE_CYCLE) {
// During feed hold, do not update rate and trap counter. Keep decelerating.
st.trapezoid_adjusted_rate = current_block->initial_rate;
set_step_events_per_minute(st.trapezoid_adjusted_rate); // Initialize cycles_per_step_event
st.trapezoid_tick_cycle_counter = CYCLES_PER_ACCELERATION_TICK/2; // Start halfway for midpoint rule.
}
st.min_safe_rate = current_block->rate_delta + (current_block->rate_delta >> 1); // 1.5 x rate_delta
st.counter_x = -(current_block->step_event_count >> 1);
st.counter_y = st.counter_x;
st.counter_z = st.counter_x;
st.event_count = current_block->step_event_count;
st.step_events_completed = 0;
} else {

```

```

    st_go_idle();
    bit_true(sys.execute,EXEC_CYCLE_STOP); // Flag main program for cycle end
}
}

if (current_block != NULL) {
// Execute step displacement profile by bresenham line algorithm
out_bits = current_block->direction_bits;
st.counter_x += current_block->steps_x;
if (st.counter_x > 0) {
    out_bits |= (1<<X_STEP_BIT);
    st.counter_x -= st.event_count;
    if (out_bits & (1<<X_DIRECTION_BIT)) { sys.position[X_AXIS]--; }
    else { sys.position[X_AXIS]++; }
}
st.counter_y += current_block->steps_y;
if (st.counter_y > 0) {
    out_bits |= (1<<Y_STEP_BIT);
    st.counter_y -= st.event_count;
    if (out_bits & (1<<Y_DIRECTION_BIT)) { sys.position[Y_AXIS]--; }
    else { sys.position[Y_AXIS]++; }
}
st.counter_z += current_block->steps_z;
if (st.counter_z > 0) {
    out_bits |= (1<<Z_STEP_BIT);
    st.counter_z -= st.event_count;
    if (out_bits & (1<<Z_DIRECTION_BIT)) { sys.position[Z_AXIS]--; }
    else { sys.position[Z_AXIS]++; }
}

st.step_events_completed++; // Iterate step events

// While in block steps, check for de/ac-celeration events and execute them accordingly.
if (st.step_events_completed < current_block->step_event_count) {
    if (sys.state == STATE_HOLD) {
        // Check for and execute feed hold by enforcing a steady deceleration from the moment of
        // execution. The rate of deceleration is limited by rate_delta and will never decelerate
        // faster or slower than in normal operation. If the distance required for the feed hold
        // deceleration spans more than one block, the initial rate of the following blocks are not
        // updated and deceleration is continued according to their corresponding rate_delta.
        // NOTE: The trapezoid tick cycle counter is not updated intentionally. This ensures that
        // the deceleration is smooth regardless of where the feed hold is initiated and if the
        // deceleration distance spans multiple blocks.
        if ( iterate_trapezoid_cycle_counter() ) {
            // If deceleration complete, set system flags and shutdown steppers.
            if (st.trapezoid_adjusted_rate <= current_block->rate_delta) {
                // Just go idle. Do not NULL current block. The bresenham algorithm variables must
                // remain intact to ensure the stepper path is exactly the same. Feed hold is still
                // active and is released after the buffer has been reinitialized.
                st_go_idle();
                bit_true(sys.execute,EXEC_CYCLE_STOP); // Flag main program that feed hold is complete.
            } else {
                st.trapezoid_adjusted_rate -= current_block->rate_delta;
                set_step_events_per_minute(st.trapezoid_adjusted_rate);
            }
        }
    }
} else {
    // The trapezoid generator always checks step event location to ensure de/ac-celerations are
    // executed and terminated at exactly the right time. This helps prevent over/under-shooting
    // the target position and speed.
    // NOTE: By increasing the ACCELERATION_TICKS_PER_SECOND in config.h, the resolution of the
    // discrete velocity changes increase and accuracy can increase as well to a point. Numerical
    // round-off errors can effect this, if set too high. This is important to note if a user has
    // very high acceleration and/or feedrate requirements for their machine.
    if (st.step_events_completed < current_block->accelerate_until) {
        // Iterate cycle counter and check if speeds need to be increased.
        if ( iterate_trapezoid_cycle_counter() ) {
            st.trapezoid_adjusted_rate += current_block->rate_delta;
            if (st.trapezoid_adjusted_rate >= current_block->nominal_rate) {
                // Reached nominal rate a little early. Cruise at nominal rate until decelerate_after.
                st.trapezoid_adjusted_rate = current_block->nominal_rate;
            }
            set_step_events_per_minute(st.trapezoid_adjusted_rate);
        }
    } else if (st.step_events_completed >= current_block->decelerate_after) {
        // Reset trapezoid tick cycle counter to make sure that the deceleration is performed the
        // same every time. Reset to CYCLES_PER_ACCELERATION_TICK/2 to follow the midpoint rule for
        // an accurate approximation of the deceleration curve. For triangle profiles, down count
        // from current cycle counter to ensure exact deceleration curve.
        if (st.step_events_completed == current_block-> decelerate_after) {
            if (st.trapezoid_adjusted_rate == current_block->nominal_rate) {
                st.trapezoid_tick_cycle_counter = CYCLES_PER_ACCELERATION_TICK/2; // Trapezoid profile
            } else {

```

```

        st.trapezoid_tick_cycle_counter = CYCLES_PER_ACCELERATION_TICK-st.trapezoid_tick_cycle_counter; // Triangle
profile
    }
} else {
    // Iterate cycle counter and check if speeds need to be reduced.
    if ( iterate_trapezoid_cycle_counter() ) {
        // NOTE: We will only do a full speed reduction if the result is more than the minimum safe
        // rate, initialized in trapezoid reset as 1.5 x rate_delta. Otherwise, reduce the speed by
        // half increments until finished. The half increments are guaranteed not to exceed the
        // CNC acceleration limits, because they will never be greater than rate_delta. This catches
        // small errors that might leave steps hanging after the last trapezoid tick or a very slow
        // step rate at the end of a full stop deceleration in certain situations. The half rate
        // reductions should only be called once or twice per block and create a nice smooth
        // end deceleration.
        if (st.trapezoid_adjusted_rate > st.min_safe_rate) {
            st.trapezoid_adjusted_rate -= current_block->rate_delta;
        } else {
            st.trapezoid_adjusted_rate >>= 1; // Bit shift divide by 2
        }
        if (st.trapezoid_adjusted_rate < current_block->final_rate) {
            // Reached final rate a little early. Cruise to end of block at final rate.
            st.trapezoid_adjusted_rate = current_block->final_rate;
        }
        set_step_events_per_minute(st.trapezoid_adjusted_rate);
    }
} else {
    // No accelerations. Make sure we cruise exactly at the nominal rate.
    if (st.trapezoid_adjusted_rate != current_block->nominal_rate) {
        st.trapezoid_adjusted_rate = current_block->nominal_rate;
        set_step_events_per_minute(st.trapezoid_adjusted_rate);
    }
}
} else {
    // If current block is finished, reset pointer
    current_block = NULL;
    plan_discard_current_block();
}
}
out_bits ^= settings.invert_mask; // Apply step and direction invert mask
busy = false;
}

// This interrupt is set up by ISR_TIMER1_COMPAREA when it sets the motor port bits. It resets
// the motor port after a short period (settings.pulse_microseconds) completing one step cycle.
// NOTE: Interrupt collisions between the serial and stepper interrupts can cause delays by
// a few microseconds, if they execute right before one another. Not a big deal, but can
// cause issues at high step rates if another high frequency asynchronous interrupt is
// added to Grbl.
ISR(TIMER2_OVF_vect)
{
    // Reset stepping pins (leave the direction pins)
    STEPPING_PORT = (STEPPING_PORT & ~STEP_MASK) | (settings.invert_mask & STEP_MASK);
    TCCR2B = 0; // Disable Timer2 to prevent re-entering this interrupt when it's not needed.
}

#ifdef STEP_PULSE_DELAY
// This interrupt is used only when STEP_PULSE_DELAY is enabled. Here, the step pulse is
// initiated after the STEP_PULSE_DELAY time period has elapsed. The ISR TIMER2_OVF interrupt
// will then trigger after the appropriate settings.pulse_microseconds, as in normal operation.
// The new timing between direction, step pulse, and step complete events are setup in the
// st_wake_up() routine.
ISR(TIMER2_COMPA_vect)
{
    STEPPING_PORT = step_bits; // Begin step pulse.
}
#endif

// Reset and clear stepper subsystem variables
void st_reset()
{
    memset(&st, 0, sizeof(st));
    set_step_events_per_minute(MINIMUM_STEPS_PER_MINUTE);
    current_block = NULL;
    busy = false;
}

// Initialize and start the stepper motor subsystem
void st_init()
{
    // Configure directions of interface pins
    STEPPING_DDR |= STEPPING_MASK;
    STEPPING_PORT = (STEPPING_PORT & ~STEPPING_MASK) | settings.invert_mask;
}

```

```

STEPPERS_DISABLE_DDR |= 1<<STEPPERS_DISABLE_BIT;

// waveform generation = 0100 = CTC
TCCR1B &= ~(1<<WGM13);
TCCR1B |= (1<<WGM12);
TCCR1A &= ~(1<<WGM11);
TCCR1A &= ~(1<<WGM10);

// output mode = 00 (disconnected)
TCCR1A &= ~(3<<COM1A0);
TCCR1A &= ~(3<<COM1B0);

// Configure Timer 2
TCCR2A = 0; // Normal operation
TCCR2B = 0; // Disable timer until needed.
TIMSK2 |= (1<<TOIE2); // Enable Timer2 Overflow interrupt
#ifdef STEP_PULSE_DELAY
TIMSK2 |= (1<<OCIE2A); // Enable Timer2 Compare Match A interrupt
#endif

// Start in the idle state, but first wake up to check for keep steppers enabled option.
st_wake_up();
st_go_idle();
}

// Configures the prescaler and ceiling of timer 1 to produce the given rate as accurately as possible.
// Returns the actual number of cycles per interrupt
static uint32_t config_step_timer(uint32_t cycles)
{
    uint16_t ceiling;
    uint8_t prescaler;
    uint32_t actual_cycles;
    if (cycles <= 0xffffL) {
        ceiling = cycles;
        prescaler = 1; // prescaler: 0
        actual_cycles = ceiling;
    } else if (cycles <= 0x7ffffL) {
        ceiling = cycles >> 3;
        prescaler = 2; // prescaler: 8
        actual_cycles = ceiling * 8L;
    } else if (cycles <= 0x3ffffL) {
        ceiling = cycles >> 6;
        prescaler = 3; // prescaler: 64
        actual_cycles = ceiling * 64L;
    } else if (cycles <= 0xfffffL) {
        ceiling = (cycles >> 8);
        prescaler = 4; // prescaler: 256
        actual_cycles = ceiling * 256L;
    } else if (cycles <= 0x3fffffL) {
        ceiling = (cycles >> 10);
        prescaler = 5; // prescaler: 1024
        actual_cycles = ceiling * 1024L;
    } else {
        // Okay, that was slower than we actually go. Just set the slowest speed
        ceiling = 0xffff;
        prescaler = 5;
        actual_cycles = 0xffff * 1024;
    }
    // Set prescaler
    TCCR1B = (TCCR1B & ~(0x07<<CS10)) | (prescaler<<CS10);
    // Set ceiling
    OCR1A = ceiling;
    return(actual_cycles);
}

static void set_step_events_per_minute(uint32_t steps_per_minute)
{
    if (steps_per_minute < MINIMUM_STEPS_PER_MINUTE) { steps_per_minute = MINIMUM_STEPS_PER_MINUTE; }
    st.cycles_per_step_event = config_step_timer((TICKS_PER_MICROSECOND*1000000*60)/steps_per_minute);
}

// Planner external interface to start stepper interrupt and execute the blocks in queue. Called
// by the main program functions: planner auto-start and run-time command execution.
void st_cycle_start()
{
    if (sys.state == STATE_QUEUED) {
        sys.state = STATE_CYCLE;
        st_wake_up();
    }
}

// Execute a feed hold with deceleration, only during cycle. Called by main program.
void st_feed_hold()
{

```

```

if (sys.state == STATE_CYCLE) {
    sys.state = STATE_HOLD;
    sys.auto_start = false; // Disable planner auto start upon feed hold.
}
}

// Reinitializes the cycle plan and stepper system after a feed hold for a resume. Called by
// runtime command execution in the main program, ensuring that the planner re-plans safely.
// NOTE: Bresenham algorithm variables are still maintained through both the planner and stepper
// cycle reinitializations. The stepper path should continue exactly as if nothing has happened.
// Only the planner de/ac-celerations profiles and stepper rates have been updated.
void st_cycle_reinitialize()
{
    if (current_block != NULL) {
        // Replan buffer from the feed hold stop location.
        plan_cycle_reinitialize(current_block->step_event_count - st.step_events_completed);
        // Update initial rate and timers after feed hold.
        st.trapezoid_adjusted_rate = 0; // Resumes from rest
        set_step_events_per_minute(st.trapezoid_adjusted_rate);
        st.trapezoid_tick_cycle_counter = CYCLES_PER_ACCELERATION_TICK/2; // Start halfway for midpoint rule.
        st.step_events_completed = 0;
        sys.state = STATE_QUEUED;
    } else {
        sys.state = STATE_IDLE;
    }
}
}

```

