DISEÑO DE UN ROBOT CARTESIANO PARA UNA APLICACIÓN INDUSTRIAL DE PALETIZACIÓN Y DESARROLLO DE PRUEBAS PRELIMINARES

Escuela Técnica Superior del Diseño

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Ingeniería Eléctrica

Realizado por Ferran Bernad Aguilar
Tutorizado por Luis Gracia Calandin
2018/2019

Agradecimientos

Quiero dar las gracias a mi familia por estar siempre ahí y apoyarme en todas mis decisiones.

A todos mis compañeros y amigos de la carrera, que me han ayudado sin pedir nunca nada a cambio.

Mención especial a mi amigo y compañero Álvaro, por haberme ayudado tanto en las piezas en impresión 3D.

Por último, agradecer a mi tutor por haberme ayudado en todo lo que ha podido y contestar siempre muy rápido todos mis correos.

Índice

Índice de figuras

Índice de tablas

1. Resumen

Este proyecto se centra en el diseño y la construcción de un robot cartesiano tipo CNC, para representar el funcionamiento de un robot paletizador en una línea de recogida de azulejos.

El trabajo se divide en tres partes significativas donde detallo las herramientas y los métodos utilizados para la realización de una maqueta.

A demás de profundizar en el tema de la robótica, programación y la utilización de nuevas tecnologías, como la impresora 3D, para la construcción de nuestra maqueta.

Abstract

This project focuses on the design and construction of a cartesian robot type CNC, to represent the operation of a palletizing robot in a tile collection line.

The work is divided into three significant parts where I detail the tools and methods used to make a model.

In addition to deepening the subject of robotics, programming and the use of new technologies, such as the 3D printer, for the construction of our model.

2. Introducción

Es objeto de este proyecto el diseño y construcción de una maqueta de un robot cartesiano de paletizado para el final de línea de cerámica.

Estos tipos de robot son esenciales en las líneas de producción. Además de apilar los productos en pales para facilitar su transporte, tiene la capacidad de ordenarlos. La idea del proyecto surgió durante mi período de prácticas curriculares en la empresa TAU Cerámica del grupo PAMESA localizada en Castellón. En la nave industrial estaba repleta de diferentes tipos de robots: apiladores, ensambladores, brazos robóticos, etc.… Pero en general la que me llamó la atención fue el robot situado en final de línea, **el robot paletizador**.

Figura 1. Pórtico en una empresa de cerámica.

Por desgracia durante estas prácticas curriculares, no tuve la oportunidad de obtener más información sobre este robot ya que pertenecía a una subcontrata, este fue uno de los motivos por los cuales me propuse hacer este TFG, con el fin de mejorar mi base en un campo de la ingeniería que está en continuo desarrollo.

3. Objetivo del proyecto

El proyecto es totalmente original, aunque está inspirado en el robot paletizador que vi en mis prácticas. Con la construcción de este proyecto quiero poner en relieve las virtudes y defectos más comunes en el diseño de un robot cartesiano, con el objetivo de conseguir experiencia e información para un futuro.

Podríamos decir que este proyecto sirve tanto para formación personal, ya que tengo que poner en práctica todo lo aprendido durante el grado para desarrollar algo real, tangible cosa que hemos hecho muy poco durante la carrera.

En conclusión, pretendo construir un robot paletizador cartesiano que pueda efectuar sus movimientos a través de accionamientos únicamente eléctricos y mecánicos, además de tener un bajo coste estos.

Se trata de un robot con 3 guías y 4 motores, cada uno con su respectiva coordenada, fundamentalmente la maqueta sirve para ver como funcionaría en la realidad dicho robot.

4. Marco teórico del proyecto

La producción de artículos aumenta cada año y por tanto aumentan las necesidades logísticas que un robot cartesiano XYZ-R puede encargarse de soportar. Diseñado para cargas medias que precisan un apilado repetitivo, un robot cartesiano mejora la productividad gracias a que libera mano de obra y reduce los problemas causados por un apilado incorrecto. Además, los robots cartesianos ahorran espacio porque trabajan "sobre" el lugar de trabajo, al contrario que los robots articulados que sí ocupan espacio en el suelo.

El robot cartesiano es un tipo de robot industrial de tres ejes los cuales operan de forma lineal, es decir su movimiento siempre es recto, no pueden girar por lo que forman ángulos rectos. Son más simples, pues su programación y configuración trabaja con menos parámetros, y económicos ya que están más limitados en sus funciones que otros robots industriales.
Este tipo de robots se utilizan mucho para realizar dibujos (impresoras 3D incluso) y también normalmente en tareas de “pick and place”.

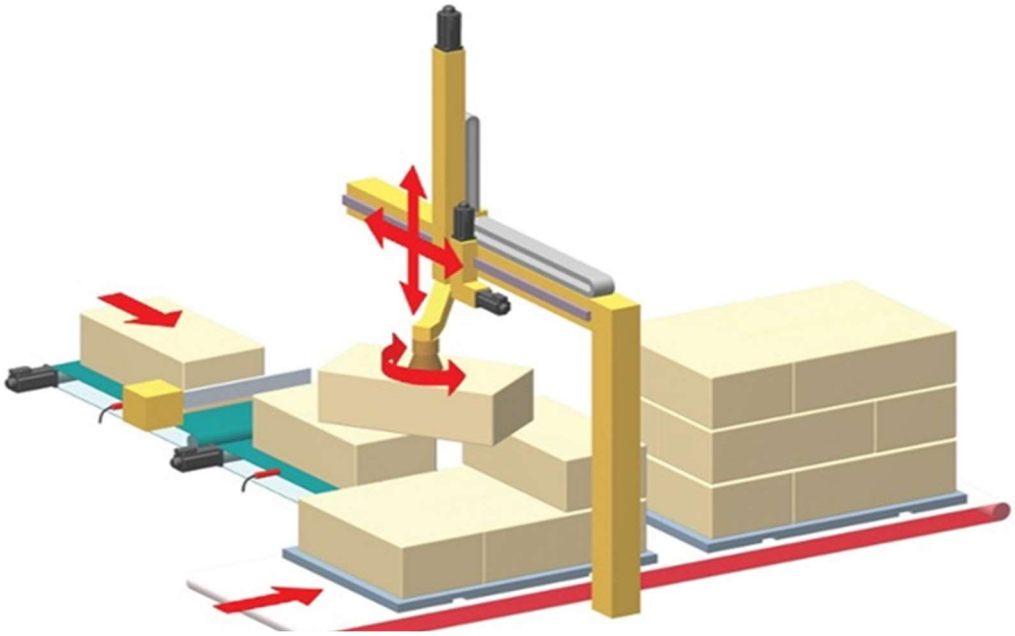


Figura 2. Robot paletizador tipo CNC

El diseño de este tipo de robot puede variar en función de la tarea que vaya a realizar. En todo robot cartesiano se pueden distinguir 3 zonas:

Zona de producción, dónde se produce el movimiento del manipulador.
Zona eléctrica, se trata de los accionamientos y el control.
Zona de supervisión, dónde se haya el control del operario.

5. Estructura del robot

La estructura de nuestro robot CNC se trata de tres partes fundamentales:

Parte eléctrica. Se trata de los motores, microprocesadores y cableado.

Figura 4. Motor PaP

Figura 3. Controlador A4988

Parte mecánica. Aquí entran todos los componentes que forman el chasis del robot, es decir, lo que la parte eléctrica (motores, cpu…) tendrá para sustentarse y mover.

En mi caso hemos utilizado varillas roscadas con husillos y las piezas que sustentan todo el sistema mediante el diseño de piezas en 3D.

Figura 6. Husillo con tuerca

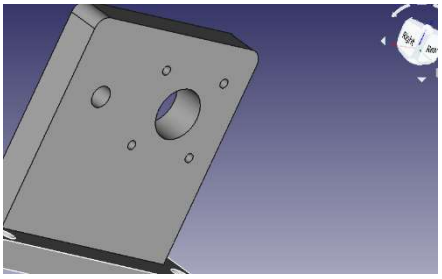


Figura 5. FreeCAD (2018)

Parte de control. Incluye los programas informáticos y la programación para la tarea a emplear el robot.

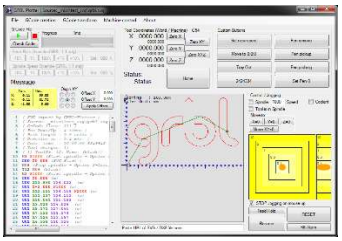


Figura 7. GRBL.

Todas estas partes entraremos en profundidad, explicando los procesos utilizados para la construcción total de la maqueta.

6. Parte mecánica

La parte mecánica de un robot CNC puede tener muchas configuraciones, cada cual más o menos conveniente dependiendo de la tarea. Tras investigar qué tipos eran los más frecuentes concluimos que debería realizar la maqueta de una de las dos formas más comunes, mediante varilla roscada con husillo o mediante correas y ruedas dentadas.

Figura 8. Correa GT2 y poleas

La primera idea fue intentar la opción de las correas. Adquirí un motor, correa y dos ruedas dentadas, para hacer unas pruebas preliminares. En seguida nos topamos con su principal problema, necesita de una unión fuerte y precisa en las correas. Una vez tienes las medidas de la correa y la cortas, volverlas a unir de una forma que las ruedas no pierdan fuerza, es haciendo que los dientes de la correa estén unidos a una distancia concreta además de que mantenga la tracción y no se rompa.

Para lograr esto necesitaba diseñar piezas complejas para la impresión 3D además de contar siempre con el peligro de que se quede algún tipo de holgura y la fuerza no pueda transmitirse bien. Aun así, hay muchos ejemplos de CNC con correas que este tipo de problemas las solucionan de maneras muy imaginativas.

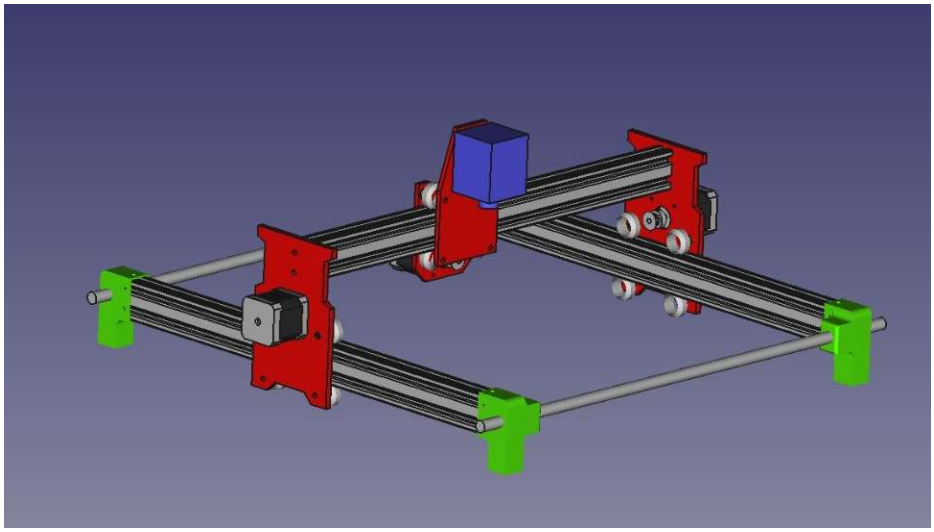


Figura 9. Ejemplo de un robot CNC por correas.

Como no disponía de las herramientas para hacer esto decidí investigar un poco más las maquinas CNC hechas mediante varilla roscada y husillo. Este tipo de configuración me aportaba una serie de ventajas que con las correas no me aportaban.

Una de las ventajas más destacables sería su facilidad de montaje, ya que queremos crear un sistema muy preciso donde no exista ningún tipo de oscilación o imprecisión, simplemente sería crear unos soportes y comprar unos acoples para el motor.

Otra ventaja es su estado en reposo, ya que al ser una varilla roscada puede mantener su posición con mucha precisión y estabilidad.

Quiero aclarar que se puede hacer de las dos formas y cada una tiene sus ventajas e inconvenientes, pero para el caso práctico que quiero llevar a cabo esta es la forma más eficiente y rentable que he visto.

Por último, será interesante como funciona de este método ya que los únicos robots CNC que he visto se movían mediante correas o cadenas y me gustaría hacerlo de otra forma para saber de primera mano cuáles son sus ventajas e inconvenientes reales.

A continuación, expongo todos los elementos utilizados para completar el proyecto, desde la parte mecánica.

Varilla roscada con husillo:

Se trata de una varilla de 500 mm de acero inoxidable, roscada para el movimiento lineal del husillo. He utilizado una varilla de 8 mm de diámetro porque son las medidas estándar de cualquier tienda especializada en este tipo de artículos. A demás el husillo viene incluido, con un avance de 2 mm por paso.

Figura 10. Husillo de 500 mm

Piezas en diseño 3D:

Estas piezas son muy importantes para el funcionamiento correcto, ya que es todo lo que va a unir las diferentes partes. Como no tenía conocimientos de diseño 3D asistido por pc, tuve que ser un poco autodidacta de la información que había por internet y di con un software libre, FreeCAD. Este software contaba con todo lo que necesitaba para elaborar las piezas necesarias.

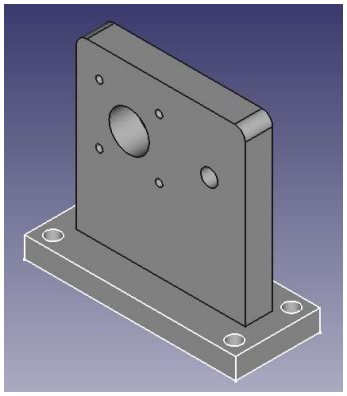


Figura 11. Pieza en diseño 3D

Básicamente son 4 piezas, 2 para el eje x (soporte del motor y soporte final) y 2 para el eje y (igual que el eje x)

Acople para ejes OD. Como sus ejes tienen diferentes diámetros necesitamos un acople de 5mm a 8 mm, que igualmente que la varilla roscada y el husillo, lo he encontrado todo en la misma tienda.

Figura 12. Acople de ejes

Cojinetes de acero de 8 mm. Se situarán en los orificios de las piezas por donde pasen las barras de soporte para que su fricción con la pieza sea el menor posible. Esto ayuda a que el sistema sea más eficiente ya que el motor tiene menos oposición y se evitan pérdidas mecánicas.

Figura 13. Cojinete de acero

6.1 Piezas en diseño 3D

Para la realización de las piezas que componen la infraestructura del robot he elegido un programa libre llamado **FreeCAD**. Esta aplicación de diseño asistido por computadora en tres dimensiones presenta un entorno de trabajo muy similar a otros más populares como puede ser el SolidWorks.

A demás cuenta con una comunidad enorme detrás de ella, con la cual he aprendido y me han ayudado para realizar todo este proyecto. Hemos diseñado 4 planos distintos para poder realizar un movimiento con 2 ejes, X e Y. Para empezar, tenemos el soporte del motor para el eje X:

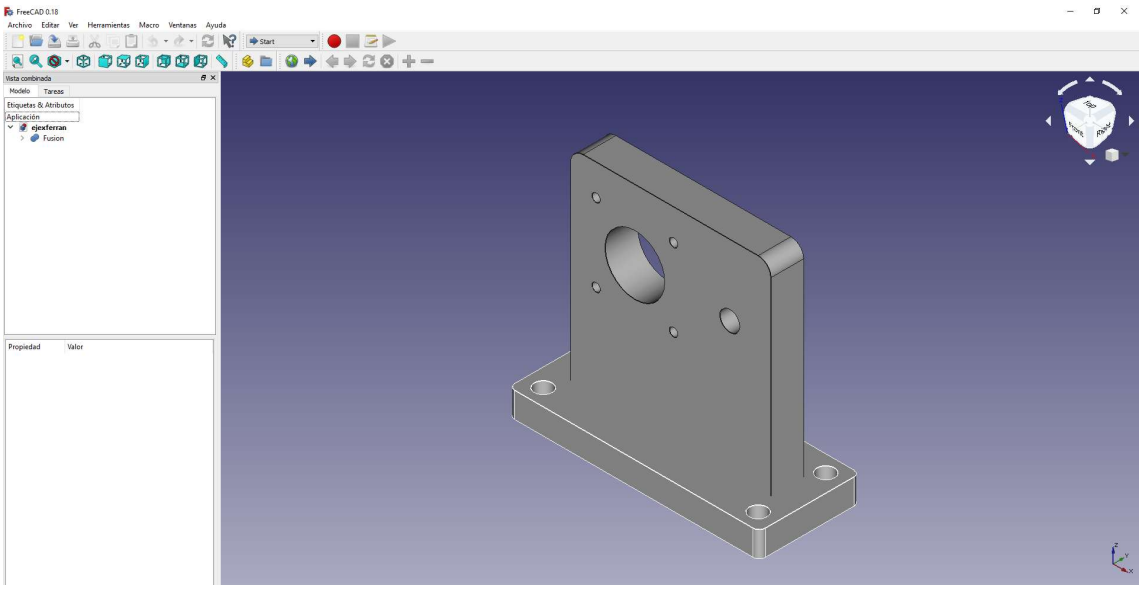


Figura 14. FreeCAD (2018) Eje X, soporte motor

Hemos dispuesto de 4 agujeros para tornillos m3 para la sujeción de nuestro motor Nema 17, y a una distancia apropiada para la sujeción del sistema, otra perforación para la barra metálica de acero inoxidable.

Todo esto está sujeto a una parte fija mediante unos tornillos m5 que se pueden apreciar en la base de la figura.

La siguiente pieza es la que utilizaremos para el apoyo de las dos barras (la del husillo y la de acero). Este elemento no necesita demasiados refuerzos ya que solo es un apoyo final:

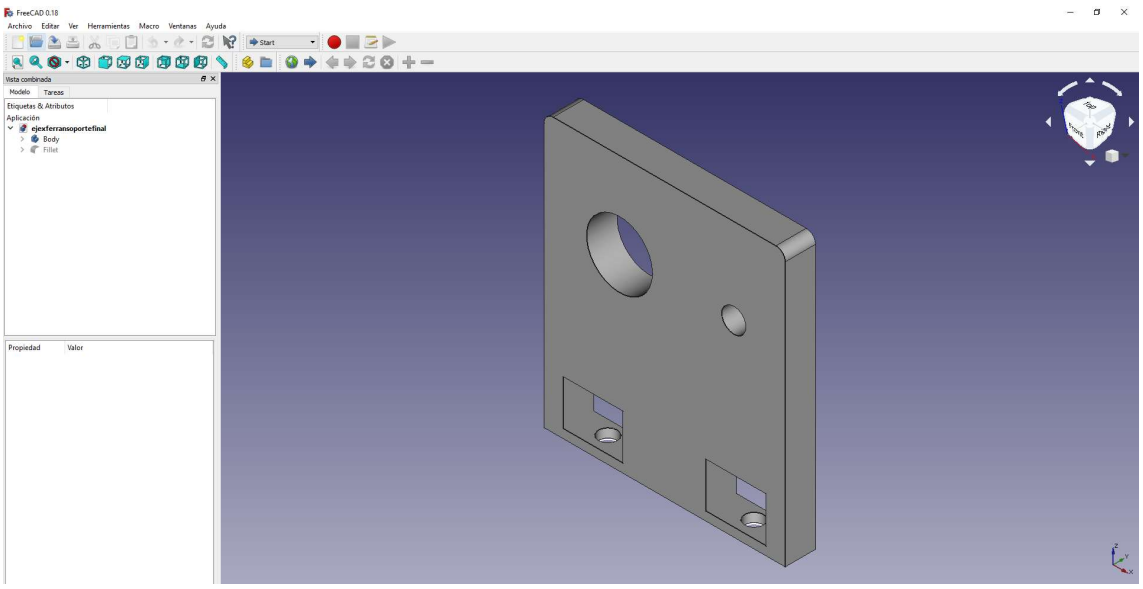


Figura 15. FreeCAD (2018), Eje X, soporte final

Como podemos apreciar en la imagen solo tenemos dos puntos de apoyo en la base, y dos agujeros, uno con el diámetro suficiente para que encaje un rodamiento para barras de 8mm y el otro para encajar la barra de acero.

Con estas dos piezas conseguimos un diseño estable y minimalista del eje X que prácticamente es el que soporta todo el sistema.

Por último, tenemos el carrete del eje y que recorrerá el eje X transversalmente:

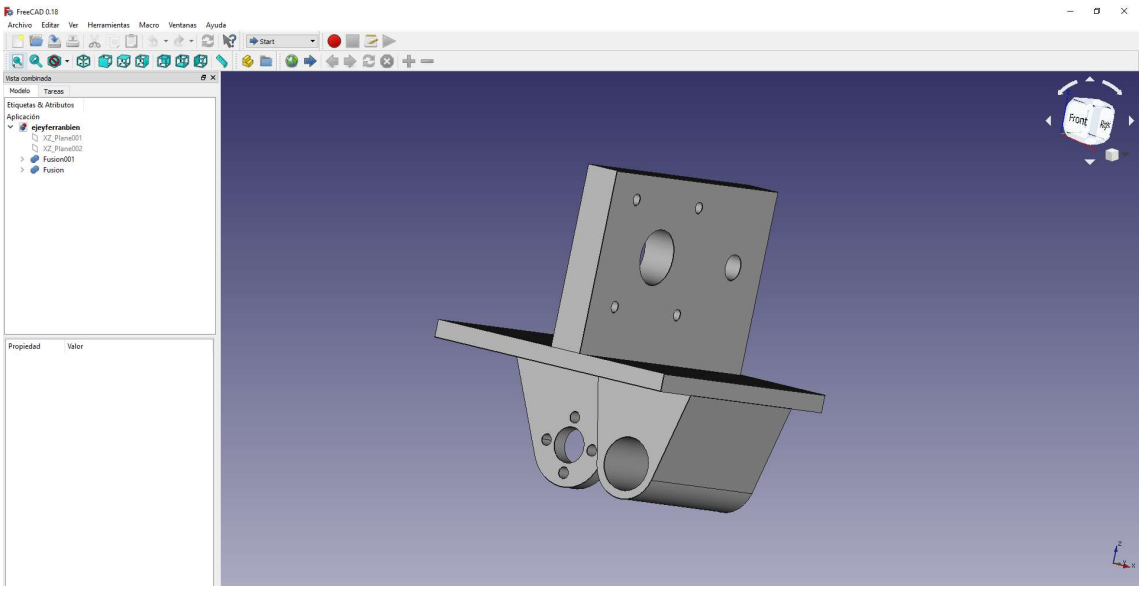


Figura 16. FreeCAD (2018) Eje Y

En esta pieza hemos cogido las medidas del soporte del motor del eje x ya que son las mismas medidas. La parte inferior está compuesta por dos elementos, el orificio para el cojinete de la barra de acero para evitar pérdidas por rozamiento, y los agujeros para la sujeción del husillo, pieza fundamental para el movimiento del robot:

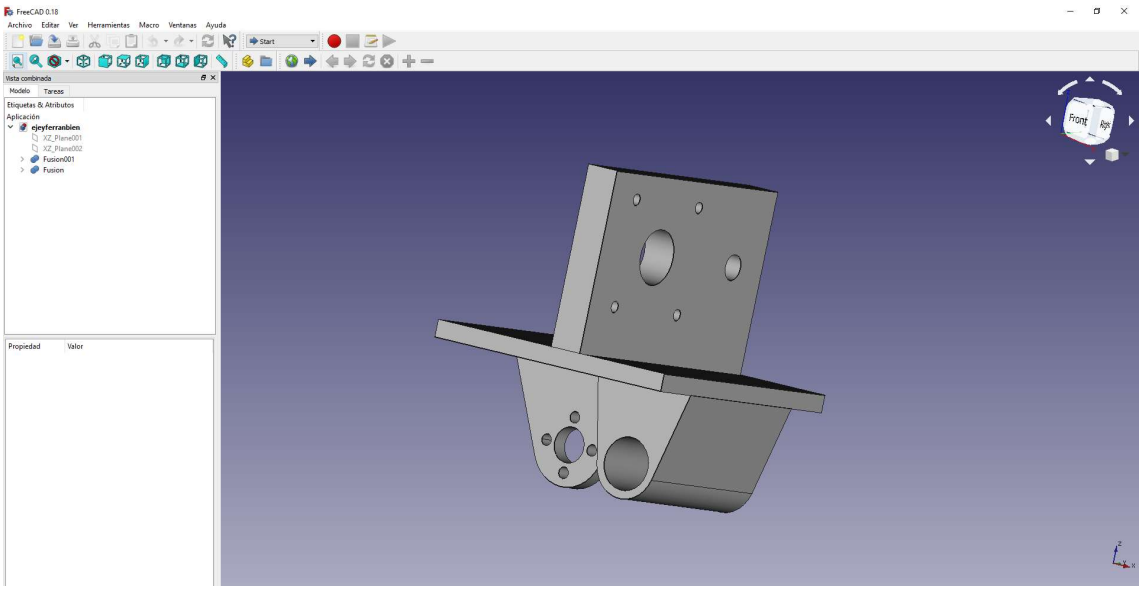


Figura 17. FreeCAD(2018) Eje Y

Todo esto lo he realizado mediante la función Sketch de FreeCAD. Esta función nos permite elaborar figuras en 2D para posteriormente darle volumen y convertirlas en figuras 3D:

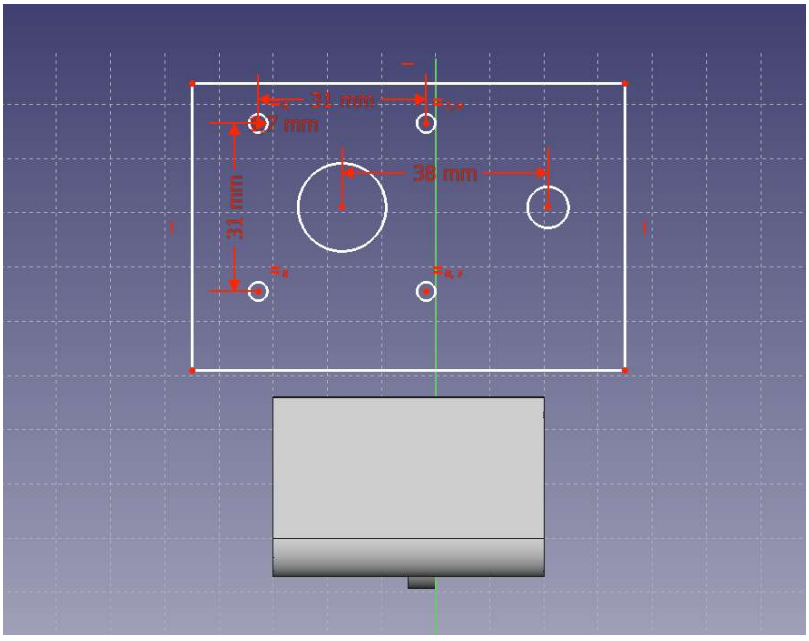


Figura 18. FreeCAD (2018), Modo Sketch

Por último, tenemos el eje z. Este eje es diferente al resto ya que nos hemos encontrado con muchos problemas para su construcción. Es debido a que la utilización de un husillo nos supondría un trabajo tal vez demasiado complejo y a la vez innecesario.

Por lo tanto, decidimos buscar una manera de transmitir este movimiento circular de nuestro motor en rectilíneo para su desplazamiento en el eje z. La solución más económica y sencilla fue el mecanismo piñón-cremallera, esto es, haciendo una guía en nuestra pieza para que la cremallera se desplace de arriba abajo:

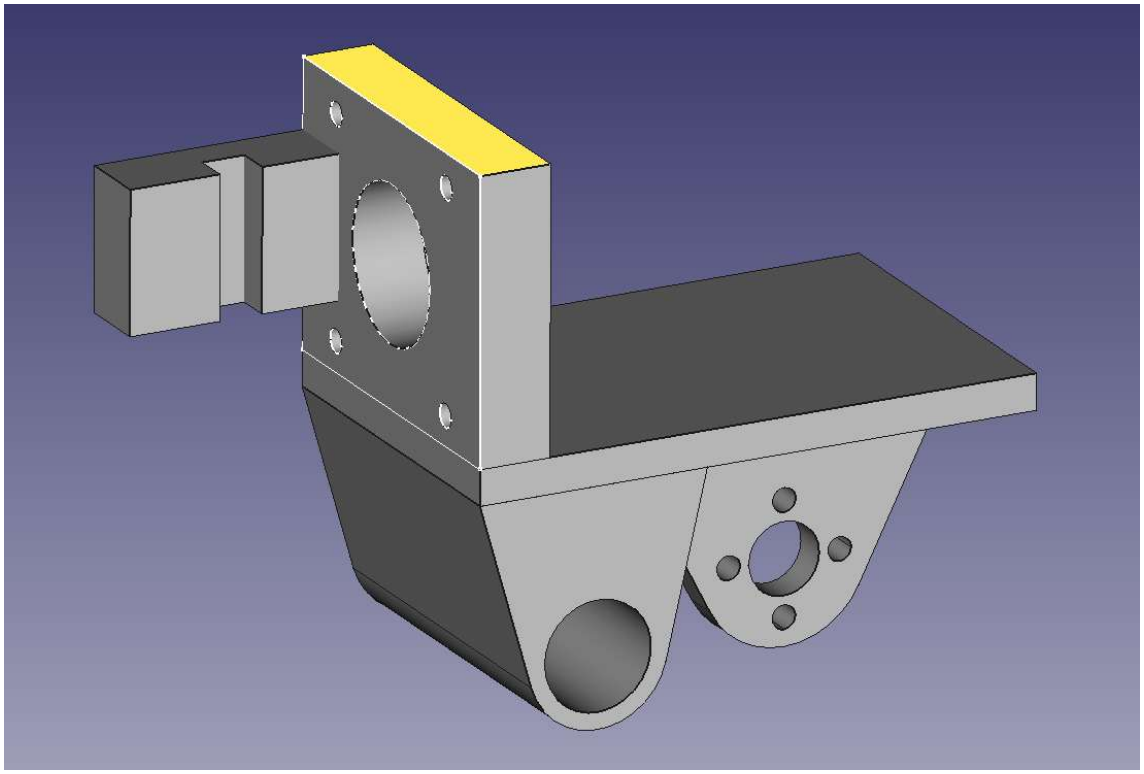


Figura 19. FreeCAD (2018) Eje Z, carrete

Con esta pieza y una guía unida en ella, mediante presión, se desplazará para recoger objetos. Este mecanismo no es el más preciso, pero si el más económico que podía optar ya que se realizara mediante la reutilización de otros elementos.

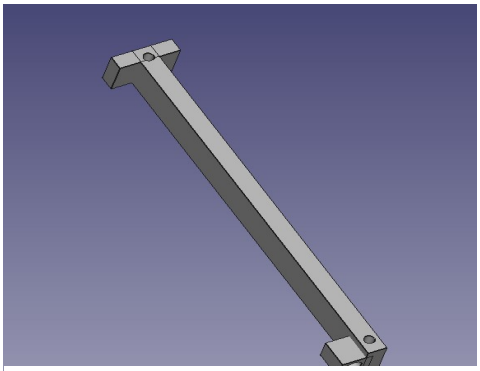


Figura 20. FreeCAD(2018) Eje Z, guía cremallera

6.2 Impresión 3D

Para la impresión de las piezas diseñadas anteriormente hemos utilizado una impresora 3D particular, la Anycubic i3 MEGA:

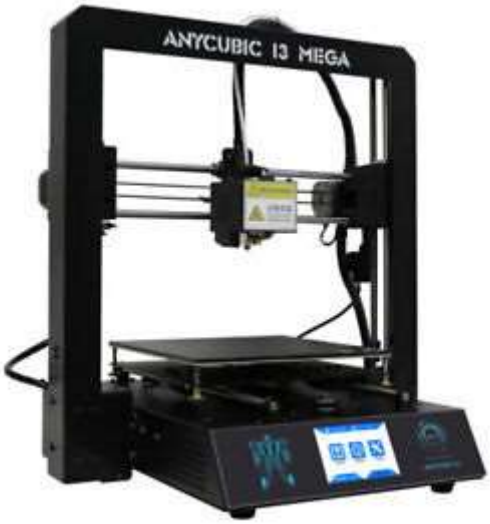


Figura 21. Impresora 3D Anycubic i3 Mega

Se trata de una impresora con una buena calidad/precio, que nos fabricará nuestras piezas en el material deseado. Uno de los aspectos positivos de esta impresora es la facilidad para su montaje y configuración, haciendo que en menos de una hora podamos estar realizando impresiones de calidad. A demás disponemos de una pantalla táctil para que nos resulte más sencillo movernos por la interfaz de la impresora.

Cuando hablamos de impresoras 3D necesitamos algún tipo de software capaz de transmitir nuestra pieza creada con algún tipo de programa en 3D (en nuestro caso es FreeCAD) hasta la impresora. Para esta tarea hemos utilizado el Simplify3D una herramienta de previsualización y configuración para las piezas a imprimir.

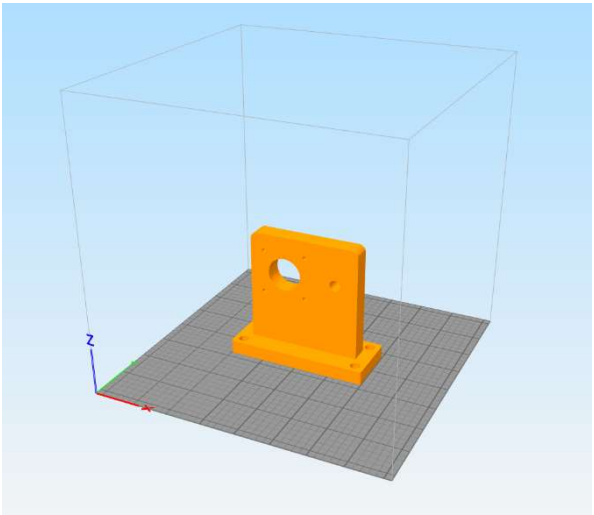


Figura 22. Previsualización Simplify3D

Este programa utiliza una interfaz muy sencilla y visual.

Sencilla porque para configurar los parámetros de impresión solamente hay que indicar con que impresora vas a realizar las piezas y este automáticamente cambia los parámetros para que se realice en el menor tiempo posible y que su calidad no se vea afectada. Estos son algunos de los parámetros que hay que modificar:

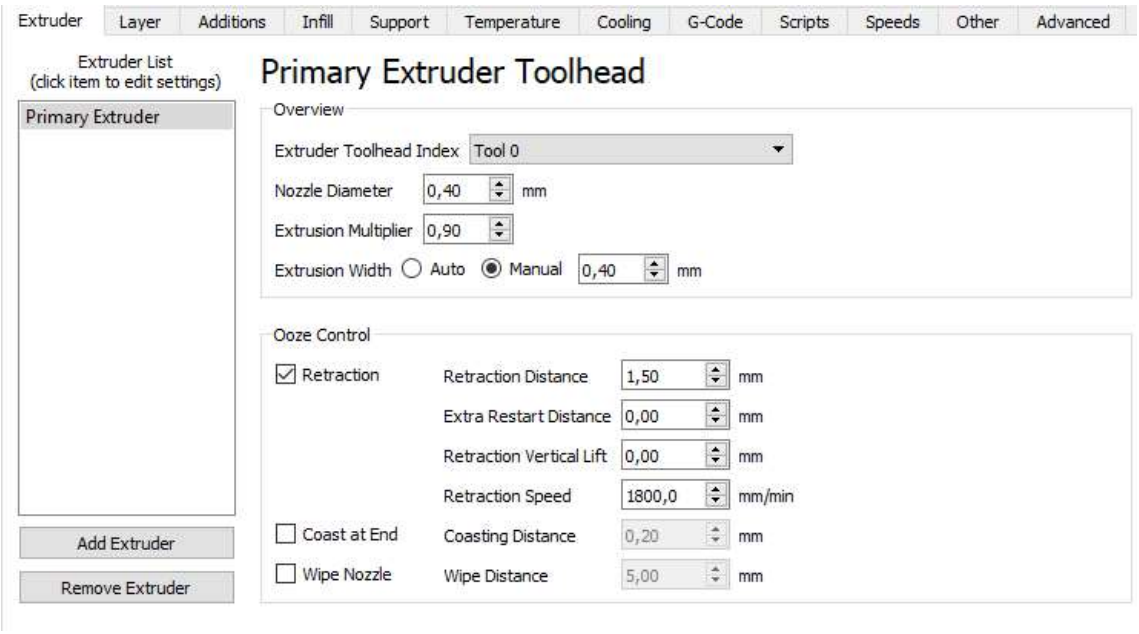


Figura 23. Parámetros avanzados Simplify3D.

Podemos observar cómo hay varias pestañas, para el extrusor, las capas, temperaturas, G-CODE… No vamos a entrar en materia para identificar que es cada cosa y para que sirve, ya que no entra en nuestras competencias. Pero hay que tener en cuenta que todas estas opciones variables afectan directamente a la calidad y a su velocidad de impresión, esto es, podemos investigar e intentar reducir estos tiempos con una configuración mucho más personalizada.

Este software también cuenta con un sistema de previsualización, dónde nos indica todos los datos útiles para que sepamos cuánto tardará la impresión, cuanto material consumirá, la longitud de material necesario e incluso su peso.

Por último, puede conectarse directamente a la impresora 3D desde un USB, o la opción de introducirlo en una tarjeta de memoria tipo SD. Guarda toda la información de la pieza además de indicar cómo debe de fabricar esta pieza, es una solución muy útil ya que rara vez disponemos de una impresora 3D al lado de nuestro ordenador personal.

7. Parte eléctrica

Cuando hablamos de esta parte nos referimos a los motores y la parte electrónica que hace posible el movimiento (motores) y su control. Estos son los componentes que he utilizado para la realización del robot CNC.

7.1 Motores

Para decidir los motores utilizados para nuestra aplicación debemos tener claro cuáles son las características más necesarias para nuestro robot.

Nuestro objetivo es crear un sistema en el cual el robot pueda moverse con sobre un sistema de coordenadas XYZ, por lo tanto, requerimos de motores que nos aporten mucha precisión en su movimiento, que su control en la medida de lo posible sea sencillo y al emplearlo en una tarea que no requiere mucha fuerza para llevarla a cabo, no necesitaremos motores de elevada potencia.

Entonces procedemos a realizar un pequeño estudio sobre los motores que hay en el mercado y los compararemos para saber cuál es el más adecuado.

Servomotor. También comúnmente llamados servos, son dispositivos de accionamiento para el control de par motor, posición y velocidad. El servomotor se compone de un motor eléctrico, un sistema de control, otro de regulación y un potenciómetro que es el que nos ayuda a saber la posición real.

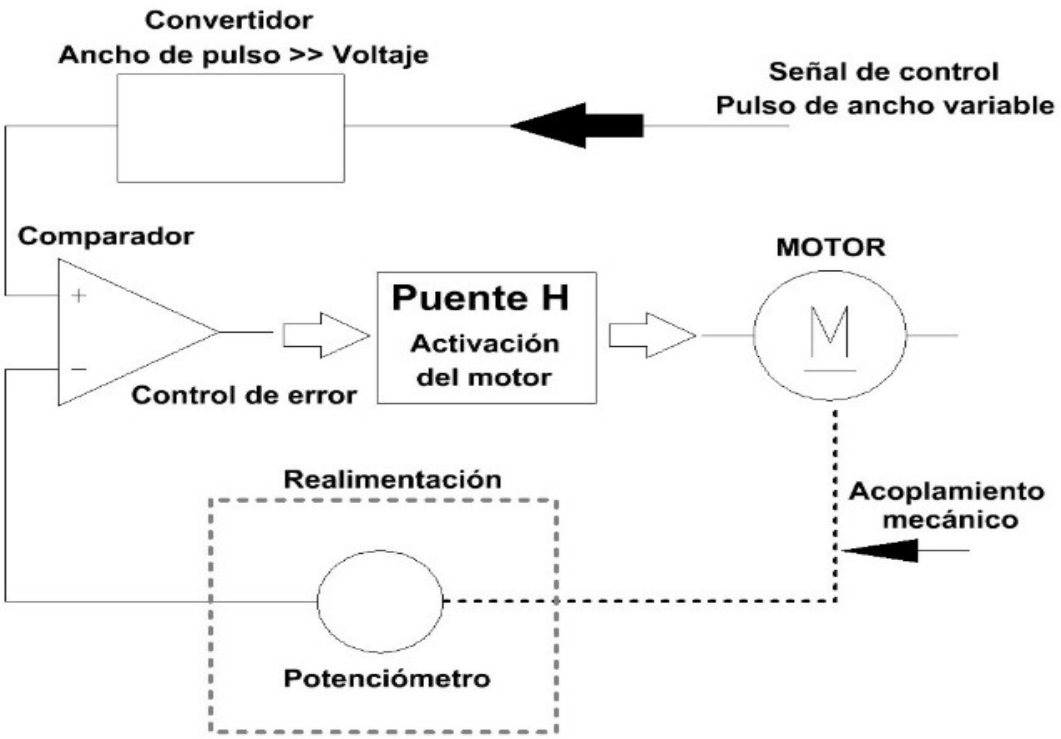


Figura 24. Diagrama de bloques servomotor

Gracias a la electrónica de control estos tipos de motores han aumentado enormemente su precisión y el rendimiento. A demás de todo esto el microcontrolador que lleva incorporado hace posible la configuración de algunos parámetros que en otro tipo de motor no encontramos.

Este tipo de motor sería una opción muy válida para proyectos a mayor escala por todas sus características, pero al tratarse de una maqueta es una opción demasiado cara y que tampoco podremos utilizar todas sus funciones y por lo tanto sería un sobrecoste innecesario.

7.1.1 Motor de corriente continua (DC). Este tipo de motores tiene un gran desgaste, pero por otro lado permite el que el control sobre la velocidad sea mucho más sencillo, además para cambiar de sentido de giro simplemente invertimos la polaridad. Dentro de estos motores nos encontramos con los motores DC brushless, que no cuenta con escobillas mejorando su rendimiento.

En nuestra aplicación estos motores no serían una elección muy adecuada ya que carecen de precisión en el arranque y parada del motor.

Figura 25. Motor DC

7.1.2 Motores paso a paso. Este tipo de motores como indica su nombre no giran, sino que transforman pulsos digitales en movimiento mecánico, por tanto, no gira de manera continua. Este motor, controlado en posición y velocidad, uno por el número de pulsos generados y el otro por la frecuencia de estos pulsos, respectivamente.

Estas características lo hacen idóneo para nuestro robot, ya que no queremos que los motores estén funcionando siempre, simplemente queremos que avance hasta una posición y se quede estacionado hasta nueva orden. A demás como no tenemos mucha inercia por los componentes del sistema el motor paso a paso no se verá afectado en su aceleración.

Dicho esto, para elegir un motor paso a paso adecuado tenemos que saber que hay dos tipos:

7.1.2.1 Unipolar. Estos motores poseen internamente dos pares de bobinas, son los motores paso a paso más sencillos de controlar. El controlador más usado es el ULN2803, muy sencillo ya que solo se encarga de encender y apagar cada bobina del motor, eso significa que el driver no sabe dónde está norte y sur. Para nuestra aplicación este tipo de motor no nos sirve ya que su potencia no es suficiente para nuestro robot.

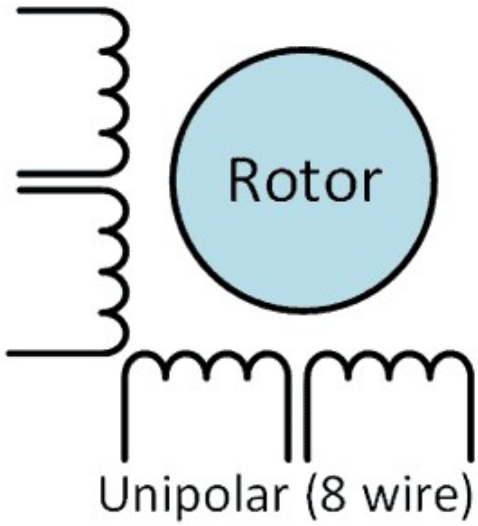


Figura 26. Conexionado motor PaP unipolar.

7.1.2.2 Bipolar. A diferencia de los motores PaP unipolares estos cuentan únicamente con dos bobinas. La complejidad de estos motores reside en su controlador, ya que tiene la tarea de suministrar corriente a la bobina y además de invertir la polaridad de esta.

He decidido emplear este tipo de motores porque son los más empleados en los robots tipo CNC, evidentemente vamos a realizarlo con motores de baja potencia controlados por un driver A4988 polulu que tiene unas prestaciones muy adecuadas que luego explicaré con más detalle.

Concretamente el motor elegido es un Nema 17, un motor muy utilizado en este tipo de robots.

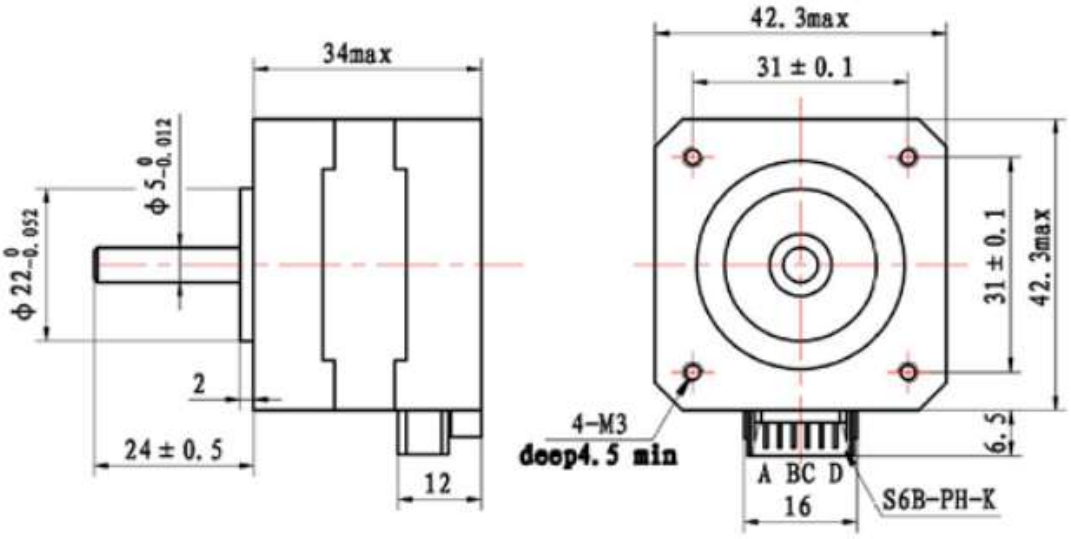


Figura 27. Motor PaP bipolar Nema 17

Especificaciones generales		Especificaciones eléctricas	
Angulo de paso	1,8	**Intensidad máxima por fase**	0,84 A
Numero de fases	2	**Voltaje por fase**	4,83V
Peso	0,2kg	**Resistencia por fase**	5,75 ohm
Par	2,8 kgcm	**Inductancia por fase**	9,3 mH

Tabla 1. Especificaciones Nema 17.

7.2 Controladores

Hay otro driver muy utilizado en casos que se requiera más potencia (recordemos que nuestro motor nema 17 soporta hasta 35 V) se trata del controlador DRV8825. Para reforzar la idea de utilizar en mi proyecto el A4988 haré un contraste entre estos dos controladores y la razón por la cual he elegido este.

Para empezar, necesitamos preguntarnos qué características debe de tener nuestro driver para el sistema diseñado. Los puntos más importantes son el precio y su corriente máxima para asegurarnos de que suministrará la corriente adecuada a nuestro motor. Tenemos una tabla comparativa:

Modelo	**A4988**	**DRV8825**
Color	Verde o rojo	Morado
Intensidad máxima	2A	2.5A
Tensión máxima	35V	45V
Microsteps	16	32
Rs típico	0.05, 0.1 o 0.2	0.1

Tabla 2. Comparación A4988 vs DRV8825.

Como podemos observar los dos controladores poseen las características necesarias para realizar nuestro sistema, ambos alcanzan altas temperaturas, disponen de sistemas de protección para cortocircuitos y sobretensión.

En cuanto al precio son componentes electrónicos muy baratos siendo el precio del A4988 de unos 0.8 € y el DRV8825 en unos 1.15 €. La diferencia del precio no es grande y técnicamente el DRV8825 es superior, pero al necesitar 4 de este tipo de controlador y al no necesitar de más potencia, optamos por el A4988.

En resumen, como no necesitamos de mucha potencia para mover nuestro motor y los dos son válidos, nos decantamos por el más barato, el A4988.

Ahora pasaremos a explicar la elección del controlador A4988 polulu, pero antes que eso explicare brevemente el porqué de la necesidad de un driver para un motor paso a paso.

Para nuestro sistema necesitamos un Arduino para controlar los movimientos del motor, pero este no tiene suficiente potencia para alimentarlos, así que requerimos de algo intermedio que sirva de puente entre el motor y el Arduino.

Ahí es donde entran los controladores o drivers, estos permiten la alimentación a un diferente voltaje al Arduino pudiendo hacer funcionar los motores que normalmente funcionan a 12 V (en nuestro caso claro está). Por eso mismo también nace la necesidad de contar con una fuente de alimentación adecuada.

Para empezar los controladores para un motor PaP bipolar son mucho más complejos que para motores unipolares. Se encargan de decidir la dirección de la corriente que circulará por el motor para así cambiar su polaridad.

Este dibujo de nuestro driver representa los pines que posee y sus conexiones:

Figura 28. Resumen pines A4988.

El elemento más común para controlar esto es el puente H. Se trata de un circuito que permite cambiar el sentido de giro del motor, su funcionamiento consiste en 4 interruptores (S1, S2, S3 Y S4), siendo normalmente transistores por su tiempo de vida útil y su elevada frecuencia de conmutación. A continuación, explicaré cómo funciona este circuito con un esquema:

Tenemos un circuito con 4 interruptores:

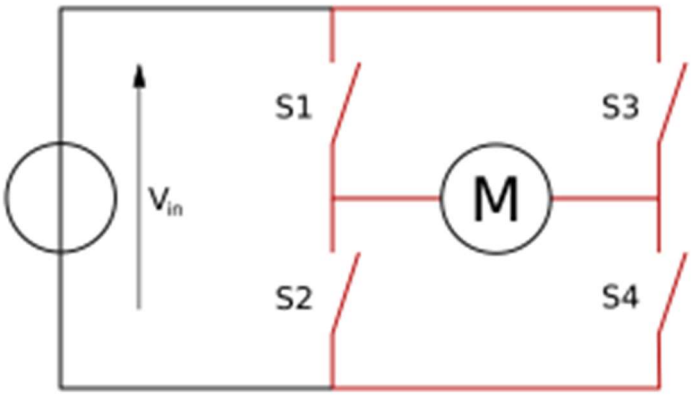


Figura 29. Puente H.

Para que gire en una dirección deseada debemos de activar los transistores S1 y S4

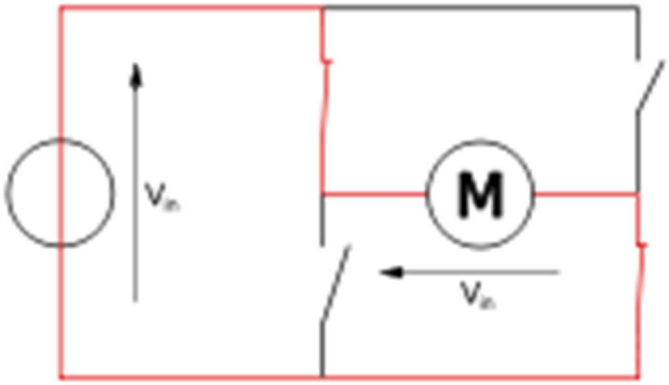


Figura 30. Puente H configuración 1.

Para que gire en la dirección contraria simplemente activaremos los transistores S2 y S3

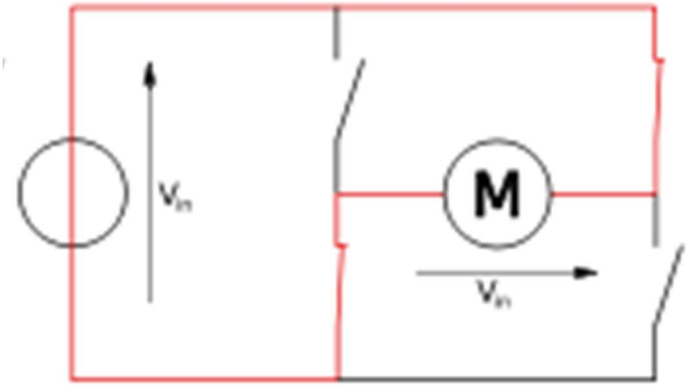


Figura 31. Puente H configuración 2.

Evidentemente si activamos los 4 transistores esto se traducirá en un cortocircuito y su consiguiente destrucción.

Esto es una explicación sencilla sobre cómo funciona el puente h que posee nuestro A4988 Polulu. A través de la página oficial de Polulu conseguimos el datasheet de nuestro controlador que nos muestra cómo funciona a nivel eléctrico con un esquema también llamado diagrama de bloques funcional:

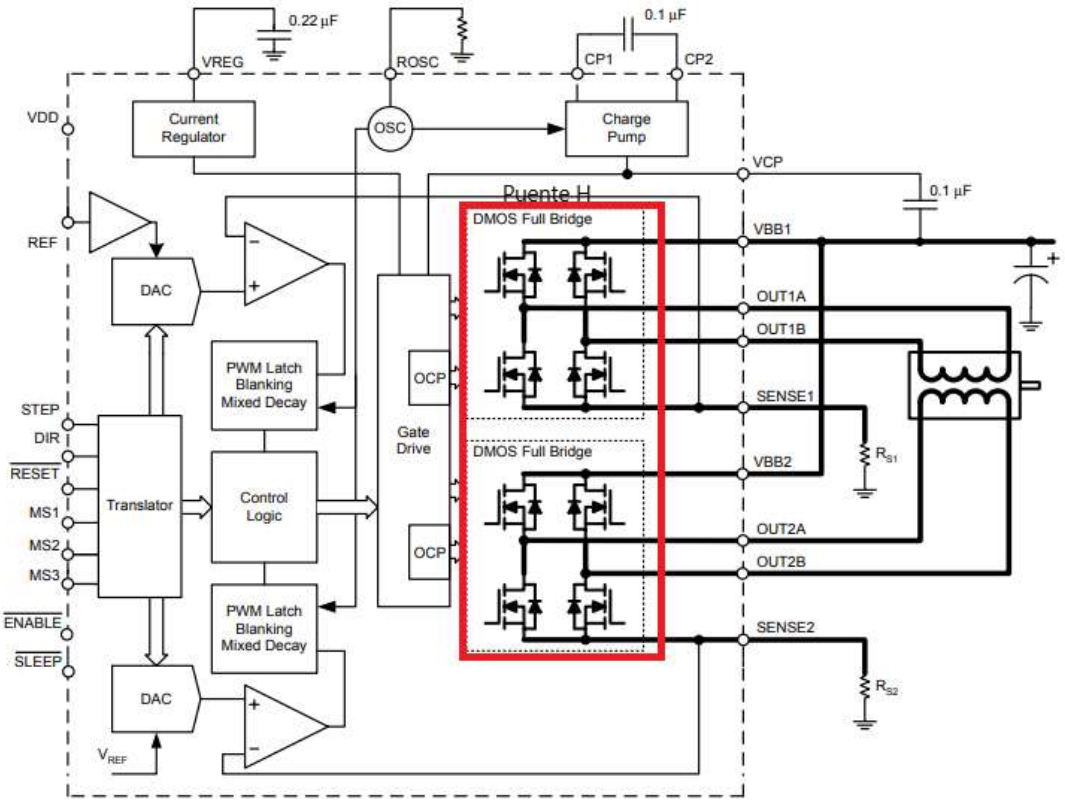


Figura 32. Diagrama de bloques del controlador A4988.

Con esto sabemos como se alimenta el motor entonces procederemos al control de este, normalmente este tipo de drivers se conecta al arduino mediante 3 cables.

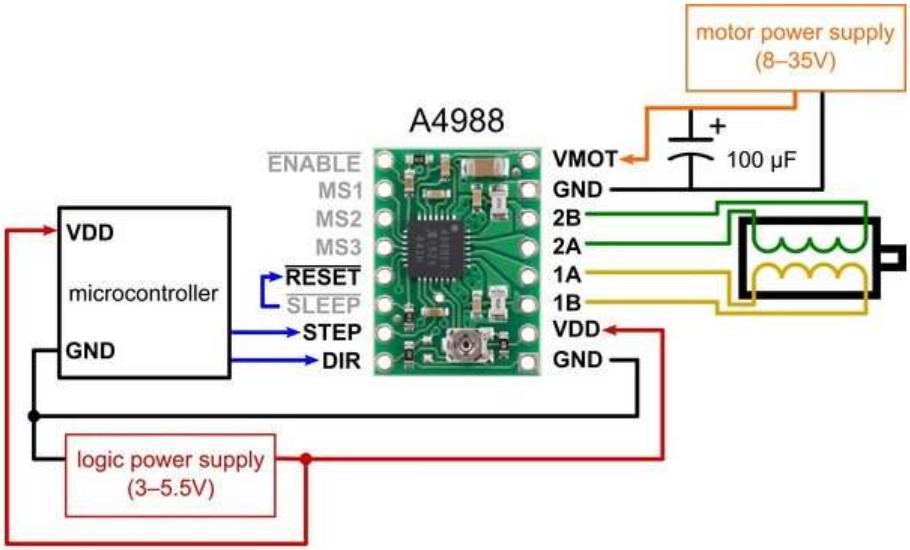


Figura 33. A4988 (E/S)

Como vemos en el dibujo el Arduino se conecta a los pines STEP, DIR y GND.

El pin STEP se encarga de recibir los pulsos que recibe del Arduino y así efectuar unos micro pulsos que hacen que el motor gire y avance un micropulso.

El pin DIR (dirección) es el que marca la dirección del motor.

El pin GND simplemente es la tierra común que es necesaria para conectar las señales del Arduino.

Entonces el proceso de control se basa en enviar pulsos rápidamente al pin step, que a su vez va comprobando todo el rato el nivel del pin DIR, para así alimentar las bobinas en la dirección que deseemos.

Los pasos que enviamos al motor son muy importantes ya que en el momento que se descontrole bloquea el motor siendo imposible su funcionamiento.

Nuestro driver cuenta con un controlador de pulsos o PWM que se encarga de regular la corriente que entra al motor introduciendo unos micro pasos para que el motor complete un paso.

7.2.1 PWM

Esto lo logra variando la corriente que pasa desde el 0% hasta el 100%:

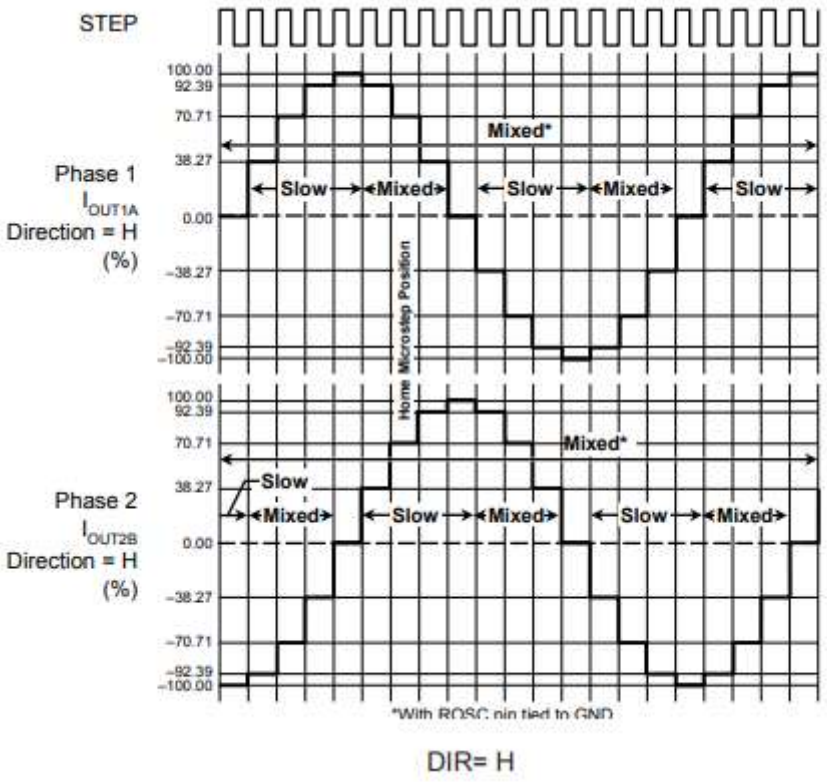


Figura 34. Gráfica modulación mediante PWM.

En esta gráfica podemos apreciar cómo va dando saltos desde el 0% a 38.27%, de ahí a un 70.71%... y así sucesivamente imitando una onda sinusoidal. Entonces los pasos se convierten en una serie de micro pasos que consiguen el movimiento sea controlado y suave.

En el diagrama de bloques de nuestro controlador aparece indicado donde se encuentra esta función y sus conexiones para que pueda desarrollarse.

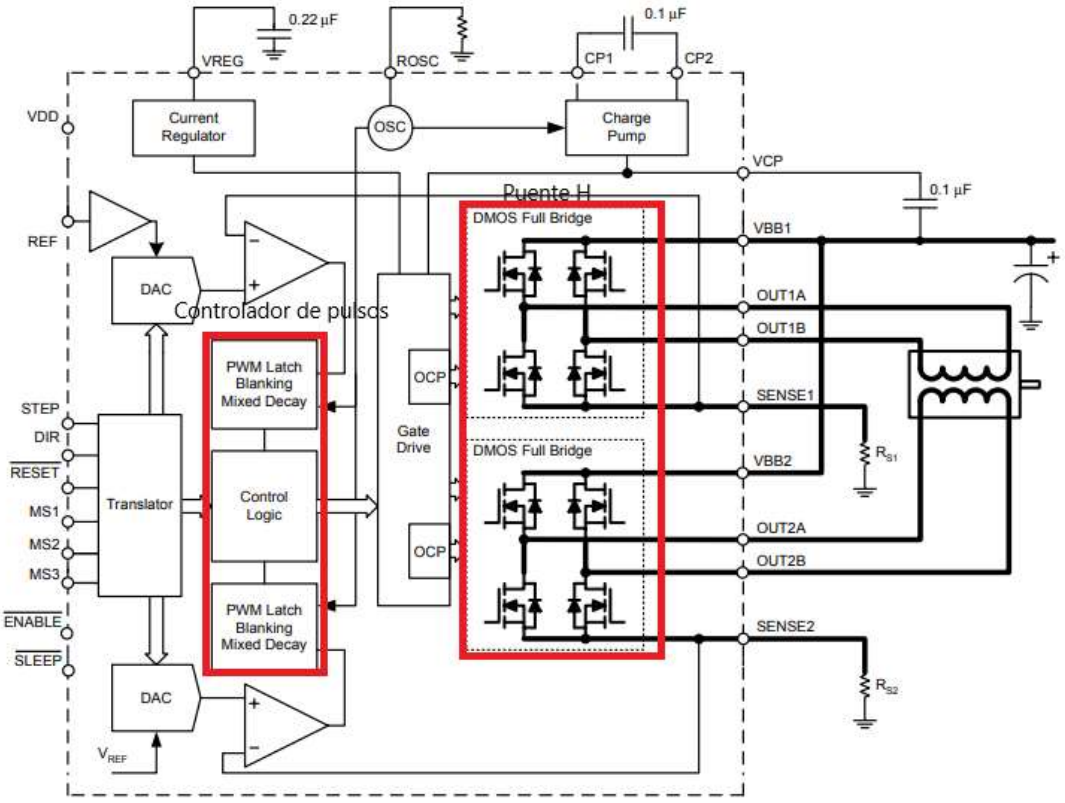


Figura 35. Diagrama de bloques A4988.

Poco a poco vamos entendiendo sus partes y como funciona.

Ya tenemos las partes del controlador de pulsos y el puente H, elementos destacados debido de su importancia para el funcionamiento adecuado del control de nuestro motor.

Con esto prácticamente podemos dar por terminado la electrónica del A4988. Sin embargo, debemos de hacer un último ajuste a estos controladores para que, en caso de sobreintensidad el controlador deje de alimentar el motor y así evitar algún tipo de averías en el motor. Esto lo detallaremos en las siguientes páginas ya que es una información necesaria para asegurarnos de que los motores no se romperán en el caso mencionado anteriormente.

7.2.2 Ajuste intensidad límite

Un ajuste muy importante de nuestro driver es la corriente máxima que puede circular por el motor ya que así evitamos que las bobinas se sobrecalienten y por lo tanto se rompan.

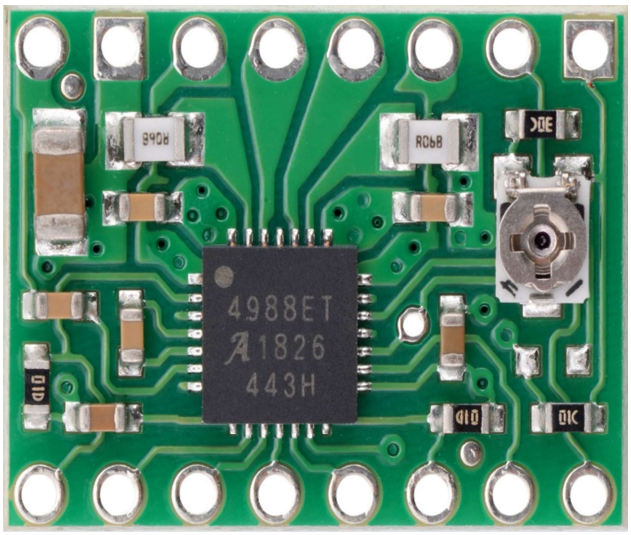


Figura 36. A4988

Como podemos apreciar, el A4988 cuenta con un pequeño potenciómetro que se encarga de regular la intensidad que recorrerá el motor para, en caso de superar la intensidad máxima que soporta el motor este automáticamente corte el suministro.

Para regular esta corriente necesitamos un multímetro, destornillador, nuestro Arduino y un capacitor. En nuestro caso me hice con un Keyes Stepper Motor, que viene incorporado un condensador y su cableado se simplifica ya que viene con un circuito impreso que hace que prescindamos de una placa board para su conexionado.

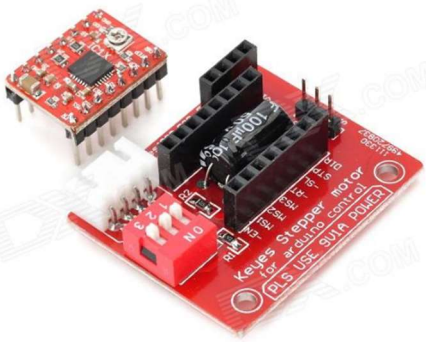


Figura 37. Keyes Stepper

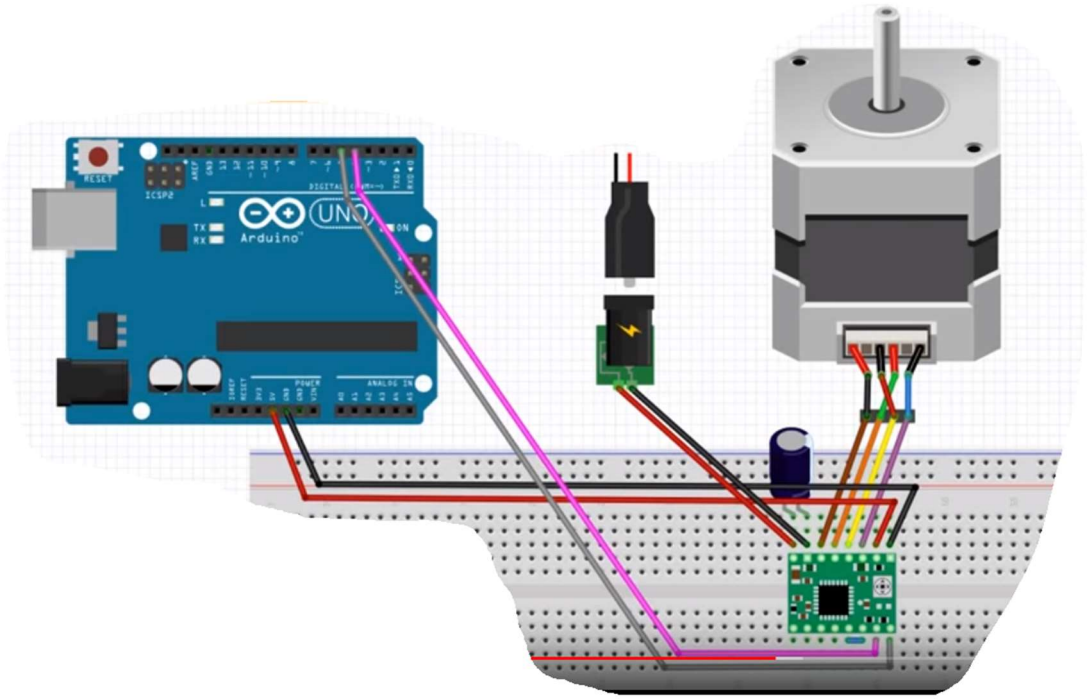


Figura 38. Conexionado para su limitación de corriente.

Podemos apreciar en esta imagen como sería el conexionado para comprobar la intensidad que circula por nuestro controlador, como ya he dicho antes en este caso no dispone del key stepper por lo tanto hace uso de un condensador y una placa board a parte. Entonces mediante la medida del voltaje que pasa por el driver aplicamos la fórmula proporcionada por el fabricante que es la siguiente:

$$I_{max} = Vref/(8 * Rs)$$

$$Vref = I_{max} * 8 * Rs$$

Dónde la Imáx es 0.84A y el Rs 0.068 Ω, dándonos lugar a una tensión de referencia de 0.457 voltios.

7.2.3 Microstepping

El microstepping es una técnica que permite controlar nuestro motor paso a paso mediante pasos inferiores al paso nominal.

Esta técnica lo que haces es emular una señal analógica para crear un campo magnético rotatorio en el motor.

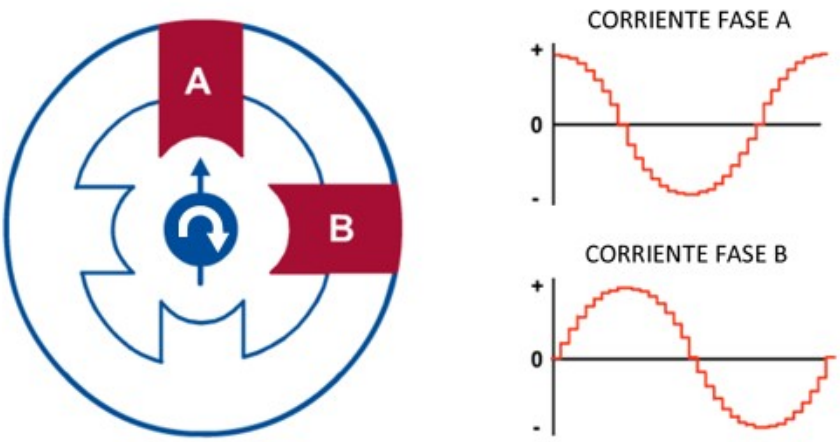


Figura 39. Señales digitales desfasadas 90º

La señal 'analógica' viene dada por el controlador digital siendo una señal analógica discretizada, dando valores discretos. Todo esto lo veremos con profundidad más adelante.

En nuestro controlador disponemos de una tabla para determinar el tipo de step que queremos:

Resolución	**MODE0**	**MODE1**	**MODE2**
Full step	Low	Low	Low
½ step	High	Low	Low
¼ step	Low	High	Low
1/8 step	High	High	Low
1/16 step	High	High	High

Tabla 3. Resolución microstepping

Esto lo controlará nuestro Arduino que a su vez recibirá la información de nuestro programa de control para el usuario u operador.

7.3 Arduino

Este microprocesador ya lo hemos mencionado anteriormente cuando hablábamos de los controladores, este será el encargado de comunicar la información que deseemos al robot. Antes de empezar a explicar la tarea que desarrollará nuestro Arduino haré una introducción a este hardware y a la vez software.

Arduino es una empresa de **hardware abierto** y **software libre**, diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan detectar y controlar objetos del mundo real.

Hardware y software libre

Cuando nos referimos a hardware libre nos referimos a que las especificaciones, datos y esquemas son de acceso libre y que cualquiera puede fabricarlos.

De la misma manera el software libre, se trata de un código accesible por cualquiera para que pueda utilizarlo y modificarlo.

Esta empresa se crea allá por el 2003 cuando por necesidades educativas en el ámbito de la electrónica se necesitaba comprar una placa que por aquel entonces no estaban al alcance de la mayoría de las personas, por esto crearon unas placas que permitieran de forma sencilla y económica tanto para estudiantes cómo para profesionales.

En nuestro caso se trata de una réplica del modelo **Arduino UNO** original, ya que no necesitamos más potencia para realizar nuestra tarea además es el microcontrolador base de todos los modelos que hay en el mercado.

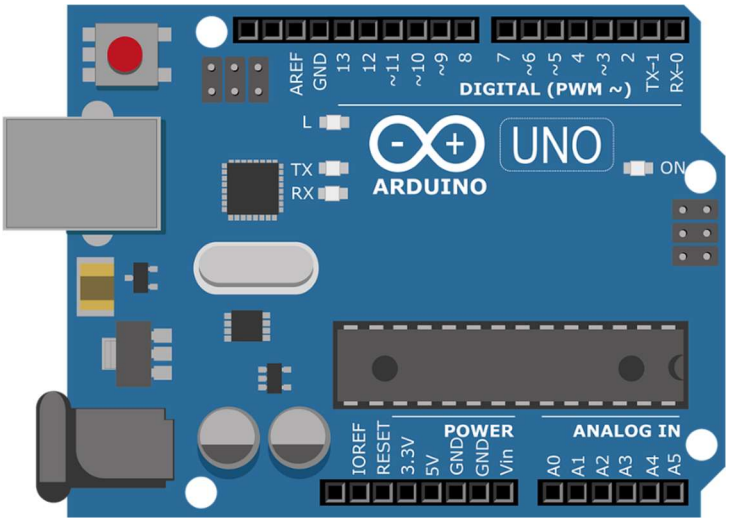


Figura 40. Arduino UNO

La imagen anterior se trata del Arduino UNO original, pero en mi caso voy a utilizar la siguiente réplica:

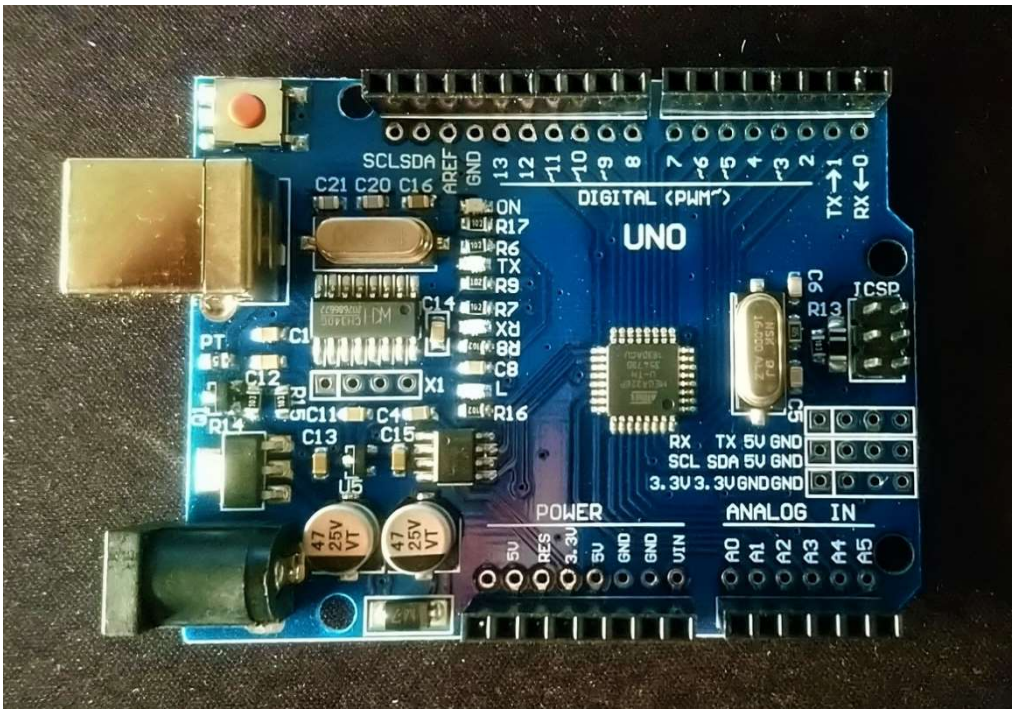


Figura 41. Arduino UNO R3

La electrónica que compone esta placa es la siguiente:

Es un microcontrolador basado en ATMEGA328P a 16MHz. Incluye 14 pines de entrada salida digital, 6 de los cuales pueden usarse como salidas PWM.

Características	
Microcontrolador	ATMEGA328P
Alimentación	5V
Alimentación recomendada	7-12V
Límite de alimentación	6-20V
Pines digitales (entrada/salida)	14 (6 con PWM)
Entradas analógicas	6
Corriente máxima por pin	20mA
Memoria flash	32KB
SRAM	2KB
Reloj	16MHz

Tabla 4. Características técnicas Arduino UNO.

7.3.1 Conversión analógico-digital

Para entender el funcionamiento de la electrónica básica del Arduino tenemos que saber que es una señal eléctrica analógica.

Una señal analógica es un valor de voltaje que puede variar y tomar cualquier valor.

Pero nuestra placa no puede trabajar con señales analógicas así que debe de tener algún componente que sea capaz de transformar estas señales analógicas en señales digitales, para que pueda entender la información adecuadamente. Este elemento es el **conversor analógico-digital**.

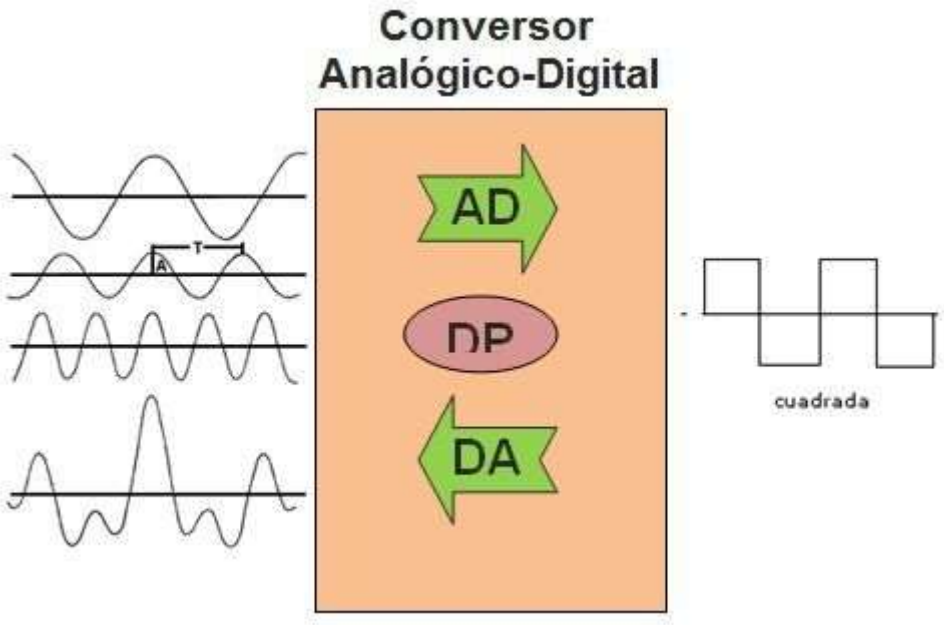


Figura 42. Conversor Analógico-Digital

Nuestro Arduino UNO tiene una resolución de 10 bits, tiene unos valores entre 0 y 1023. Para conseguir esta digitalización recurrimos al conversor analógico-digital que se compone de tres elementos:

1. Muestreo.
2. Cuantificación
3. Codificación

7.3.1.1 Muestreo

El convertidor realiza una serie de aproximaciones sucesivas, este método consiste en hacer comparaciones, en intervalos regulares una detrás de otra, para encontrar un valor digital hasta que iguale la tensión entregada por el conversor y la tensión de entrada. Recordemos que para cada bit de nuestro Arduino representa 4,833 mV.

Durante el muestreo la señal sigue siendo analógica, ya que aún puede tomar cualquier valor.

Aquí tenemos un ejemplo sobre cómo se produciría un muestreo simple, dónde cogemos 10 muestras a través de 1 periodo:

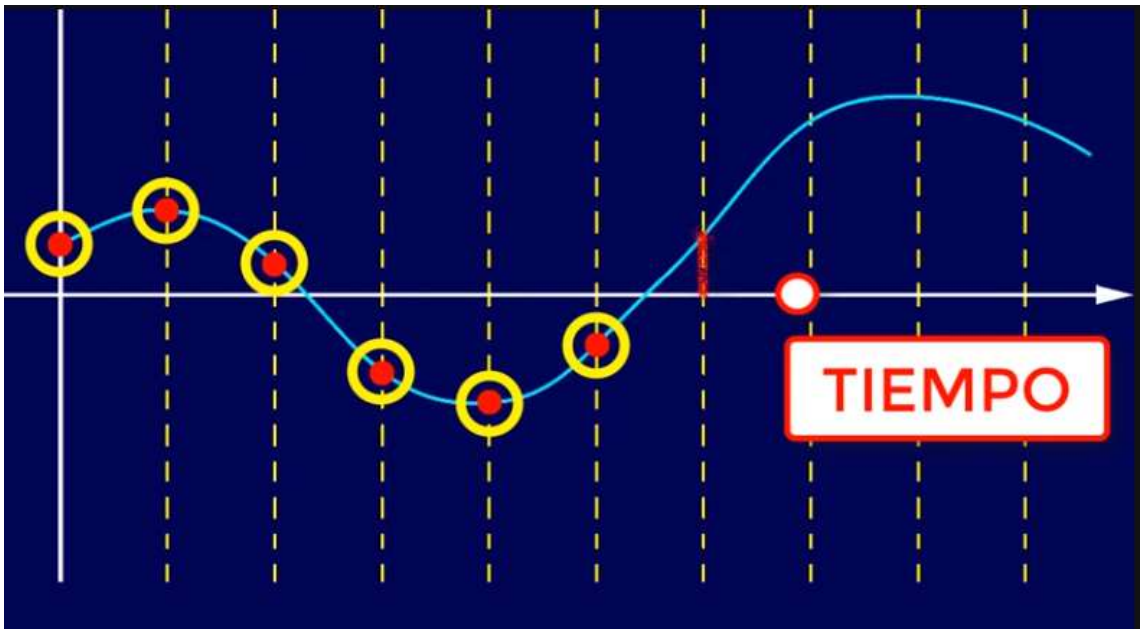


Figura 43. Muestreo.

El muestreo tiene una serie de características que harán variar el funcionamiento de este:

- Tensión máxima de entrada
- Numero de bits
- Resolución
- Tiempo de conversión
- Error de conversión

7.3.1.2 Cuantificación

El objetivo de este proceso es situar con bits la señal analógica mediante una asignación de niveles, o en nuestro caso, franjas.

Por lo tanto, cogemos los valores anteriores y los situamos dentro de unas franjas determinadas anteriormente con distintos valores. Podemos verlo reflejado de una manera gráfica:

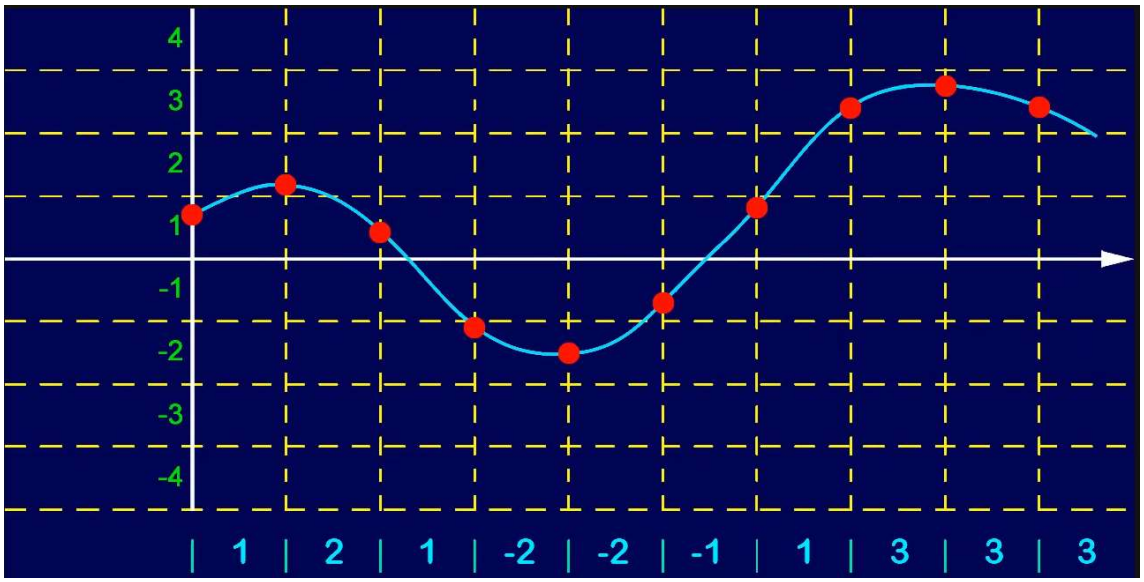


Figura 44. Cuantificación.

Estas franjas se utilizan porque tenemos que recibir un valor discreto, esto es un valor redondeado. En nuestro ejemplo solo tenemos 8 franjas, 4 para positivo y 4 para negativo, la cantidad de franjas afecta directamente a la precisión de nuestro convertidor.

Esto se puede desarrollar de diferentes maneras, algunos de los más conocidos son:

- Cuantificación uniforme. Este tipo de cuantificación simplemente tratan todas las señales igual, sin importar su procedencia.
- Cuantificación no uniforme. Al contrario que la anterior este sigue un proceso no lineal, esto es, sabiendo de un principio que no se reciben señales de una misma naturaleza, se procede a analizar las señales sabiendo que cada una es más sensible a una determinada banda de frecuencia.

Para finalizar este apartado cabe destacar que cómo es lógico, este proceso tiene un error conocido como “Error de cuantificación”. Esto es debido a que normalmente nuestro conversor no puede dar su valor exacto para la señal analógica y recurre a una estimación.

7.3.1.2 Codificación

Como tenemos 8 datos necesitamos una palabra de 3 bits para que nos dé un valor para cada franja. Esto es:

TABLA DE VERDAD	
4	111
3	110
2	101
1	100
-1	011
-2	010
-3	001
-4	000

Tabla de verdad para la codificación. Tabla 5

Cada uno de estos valores obtienen una numeración binaria siendo, por ejemplo, el 2 (010), el 3 (011) y el 0 (000). Por tanto, ya tenemos los valores de nuestra señal analógica en unos valores binarios que el microcontrolador Arduino puede procesar.

Estos 3 bits son lo que llamamos palabra en codificación, este sistema de codificación tiene el nombre de modulación mediante codificación de pulsos o en en inglés pulse code modulation (PCM).

Para finalizar en este apartado un pequeño resumen del procedimiento que realiza el conversor analógico-digital:

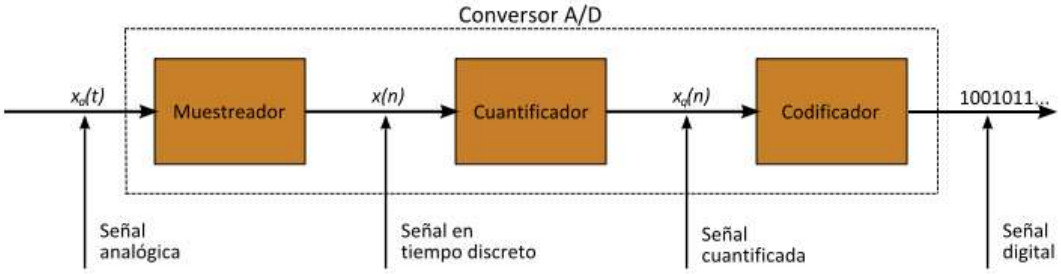


Figura 45. Esquema Conversor A/D.

Ahora ya sabemos cómo el Arduino reconoce una señal analógica, pero ¿cómo enviamos una señal analógica desde nuestro microcontrolador?

7.3.2 Señal digital a señal analógica (PWM)

En seguida nos damos cuenta de que las conexiones donde van conectados los motores están en los pines 3, 5, 6, 9, 10, y 11:

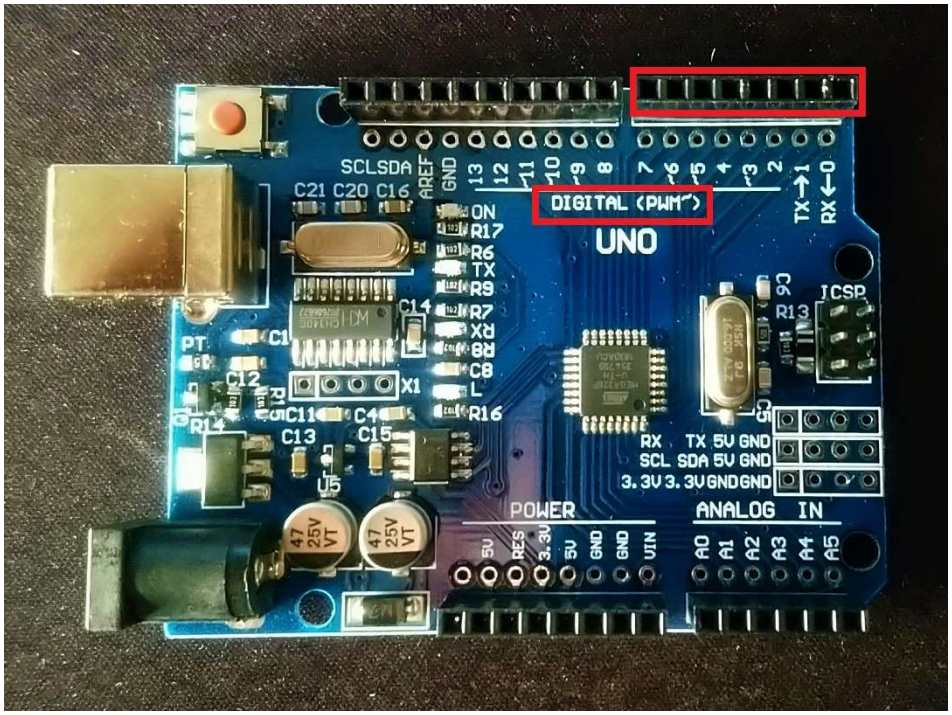


Figura 46. Arduino UNO R3.

Esto son las salidas digitales moduladas por PWM (pulse width modulation). Cabe destacar que estas señales no son analógicas, sino que se emplea un 'truco' para que mediante señales digitales imite a una analógica. Esto es debido a que en nuestro Arduino UNO no existen salidas analógicas puras por lo que deberemos de utilizar estas salidas.

La **modulación por ancho de pulso** es una técnica para simular señales analógicas usando pulsos digitales. En PWM se genera una señal cuadrada encendiendo y apagando un pin. Este patrón de encendido y apagado puede simular voltajes entre siempre encendido (5V) y apagado (0V), cambiando el porcentaje de tiempo durante el que la señal está encendida respecto al que pasa apagada.

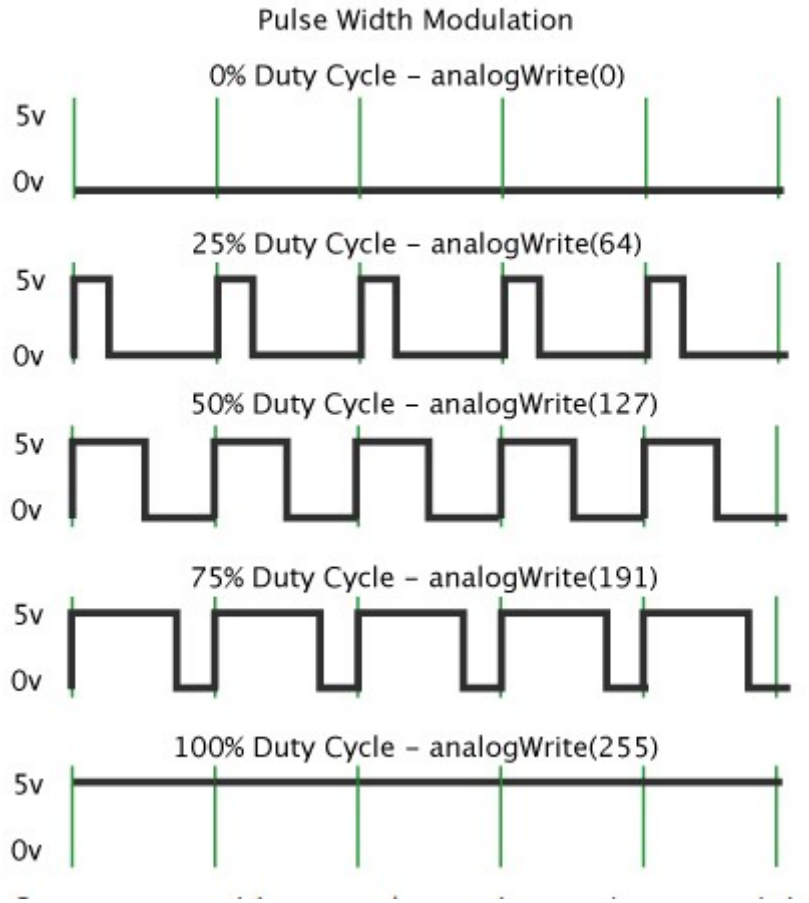


Figura 47. Gráfica variación Duty Cycle

El tiempo cuando la señal esta encendida se llama ancho de pulso. Este ancho de pulso es el valor que tenemos que cambiar para que nos salga una señal analógica. El porcentaje de tiempo que la señal se encuentra encendida respecto al período de la señal se llama ciclo de trabajo o duty cycle en inglés.

Entonces si repetimos este proceso el resultado será una señal analógica que su valor dependerá del ciclo de trabajo.

¿Entonces cuál es la tarea que necesitará empeñar nuestro microcontrolador?

El programa que deberá ejecutar será uno capaz de enviar la información hacia los controladores de los motores para situarnos en una posición exacta en el espacio.

Para lograrlo debemos de explicar que contamos con ¨mochilas¨ también llamadas shields que se encargan de proporcionar a la placa infinidad de funciones.

7.4 CNC Shield

Esta extensión nos permite el control de nuestros motores paso a paso de una manera sencilla para un proyecto tipo CNC. Esta placa cuenta con 4 zócalos para controlar 4 motores paso a paso, para insertar 4 controladores tipo polulu A4988 o DRV8825 cómo máximo.

Su sencillez reside a que simplemente es un Plug'n Play, esto es, conectarlo a nuestro Arduino y funcionar.

A continuación, nos vamos a encargar de estudiar el circuito electrónico para saber cómo funciona internamente esta extensión. Pero antes de esto debemos de hacer una descripción breve de sus características:

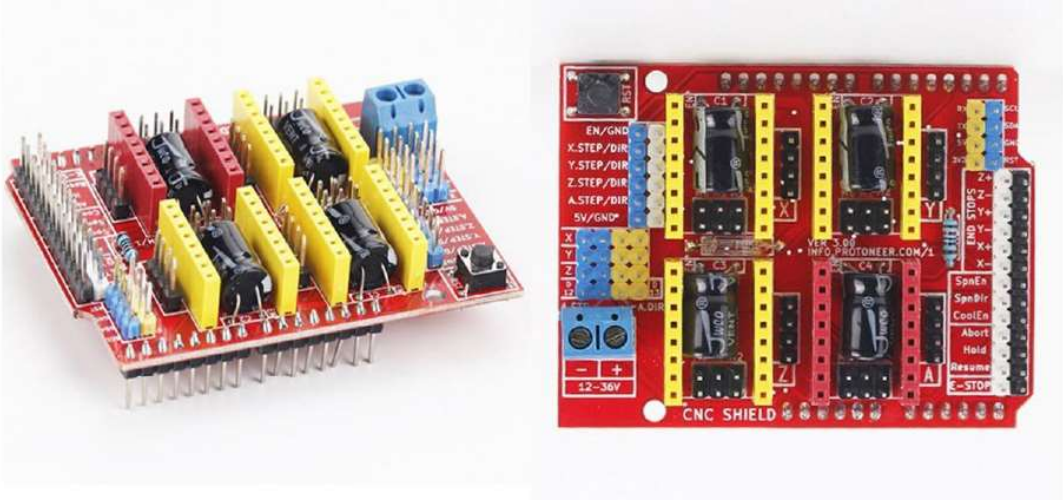


Figura 48. CNC Shield

Características generales:

- Totalmente compatible con la última versión del firmware de código abierto GRBL (0.9j). Este firmware es el que permite la comunicación usuario-máquina a través de una interfaz gráfica.

- Cuenta con 4 ejes de soporte X, Y, Z, A. El ultimo eje, el A, se puede utilizar para duplicar cualquier otro eje utilizando los pines D12 y D13. Esto lo utilizaremos más adelante para duplicar el eje X para aportarle mayor estabilidad.

- Disponemos de conexiones para hasta 6 finales de carrera, 2 por cada eje, facilitando mucho el trabajo del conexionado de estos.

- Puentes para el ajuste de los micro pasos haciéndolo más preciso si lo necesitamos. El A4988 tiene una precisión máxima de 1/16.

- Una regleta para soportar un voltaje entre 12-36 VDC. Nuestros controladores A4988 polulu soportan entre 8-35 VDC así que trabajaremos en 12 VDC-

- Molex de 4 pines, para conectar tus motores PAP.

En cuanto a cómo están conectados cada uno de sus pines a nuestra placa de Arduino lo detallamos a continuación:

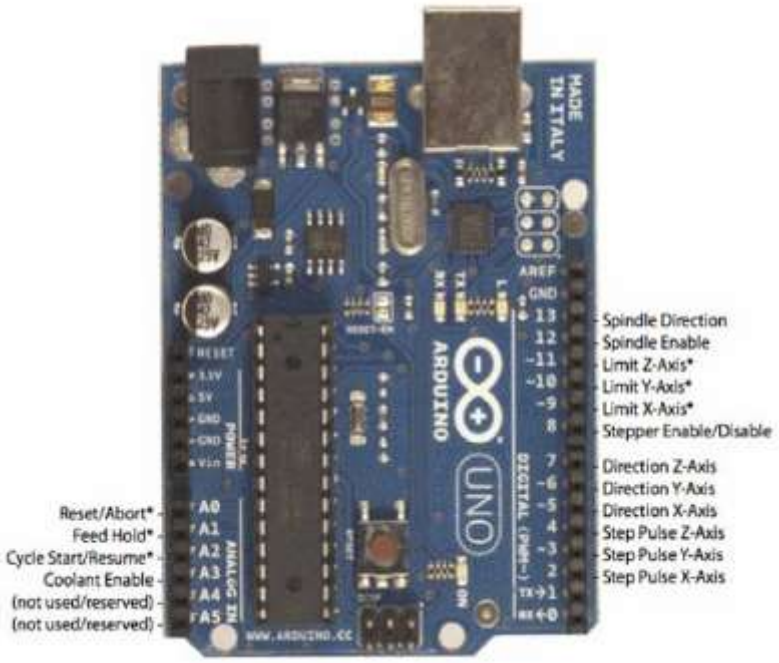


Figura 49. Configuración pines Arduino con CNC Shield.

Como podemos apreciar en la imagen proporcionada por el fabricante, casi todos los pines del Arduino están ocupados. Empezando por la derecha desde arriba hacia abajo nos encontramos con:

- Pin 13. Se utiliza para controlar la dirección del husillo.

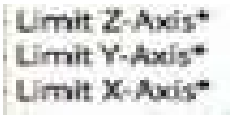
Spindle Direction

- Pin 12. Para activar el husillo.

Spindle Enable

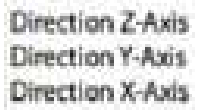
Cabe aclarar que este husillo es el que tiene que activar el láser, taladro… El elemento cortador de una máquina CNC, como en nuestro caso su utilidad no es cortar si no transportar (Pick n' Place) no lo utilizaremos como tal.

- Pin 11, 10, y 9. Estas son las conexiones para los finales de carrera para cada uno de nuestros ejes (X, Y, Z).

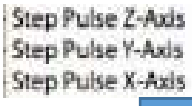


- Pin 8. Para activar/desactivar los motores recurriremos a este pin.

- Pin 7, 6 y 5. Se trata de la dirección de giro de cada uno de los ejes.



- Pin 4, 3 y 2. Estos pines sirven para la entrada y salida de los pulsos que se dan a cada eje, respectivamente.



Con los últimos 6 pines mencionados anteriormente se logra controlar nuestro sistema en velocidad, posición y giro.

- Pin A0. Entrada para la opción de abortar.

- Pin A3. Una opción de la última versión del CNC shield, contamos con una opción de refrigeración.

Movimiento OpenSource

En español, movimiento del software de código abierto, defiende el software en código abierto que surge como alternativa al código libre centrándose más en una cuestión pragmática que en cuestiones éticas. Esto es debido a que a veces el concepto "libre" se puede hacer referencia a adquirir software de una manera gratuita.

OPEN SOURCE INITIATIVE

En total tenemos 4 pines libres que los podremos usar en caso de querer hacer algún tipo de extensión.

Explicado esto podemos ver mediante algunas imágenes de la placa impresa sus conexiones entre el Arduino y el CNC shield. Los datos de esta placa no se encuentran en la red ya que sus autores se molestaron que los productores chinos hicieran copias de esto sin proporcionar alguna novedad, siendo totalmente contrario al movimiento del OpenSource.

Por tanto, buscando por internet nos encontramos con una placa prácticamente idéntica y que cuenta con toda la documentación que detallaremos en la siguiente página.

Esta imagen representa todos los cableados, desde los pines exteriores donde se conecta el Arduino hasta los pines dónde están situados cada zócalo para cada controlador.

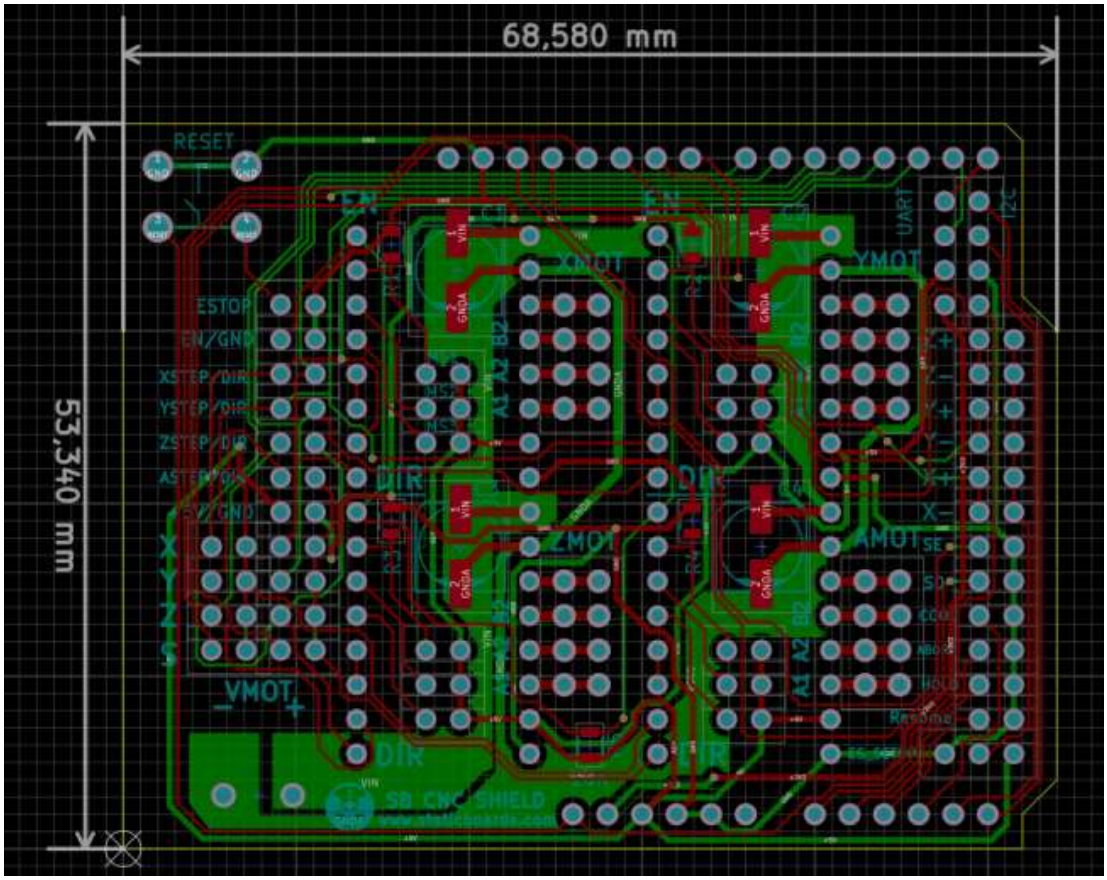


Figura 50. PCB de CNC Shield

Observando este esquema entendemos la cantidad de cableado que nos ahorramos al emplear una placa impresa.

Para realizar una simplificación de este circuito y entenderlo mejor, en la siguiente imagen se explica que pines corresponden a cada función de nuestro CNC shield:

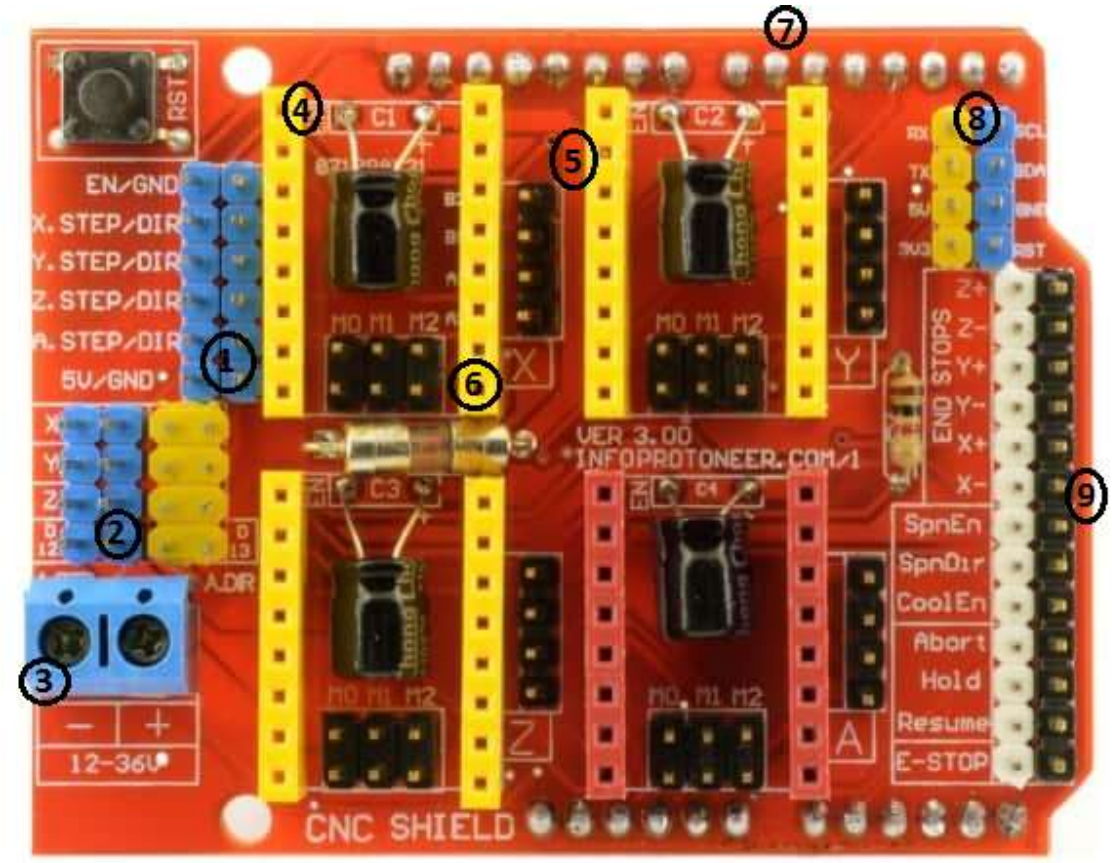


Figura 51. CNC Shield configuración pines

1. Pines para cada motor de STEP (paso) y DIR (dirección).
2. Estos pines sirven para clonar un eje tal y como lo habíamos explicado anteriormente.
3. La entrada de voltaje en continua, desde 12 V a 36 V.
4. Zócalos para la conexión de los controladores A4988 u otro compatible como el DRV8825 en caso de necesitar más potencia.
5. Estas conexiones son las que van directamente al motor para el control del bobinado.
6. Pines para controlar los micro-pasos, siempre atendiendo las limitaciones de nuestro controlador.
7. Estos pines soldados son los que comunican con el Arduino y van a cada conexión pertinente del CNC shield.
8. Estos son los pines que se quedan libres para nuestro uso.
9. Estas salidas son utilizadas para los finales de carrera de cada eje y para las opciones del control del husillo, refrigeración y la parada de emergencia.

El esquema general:

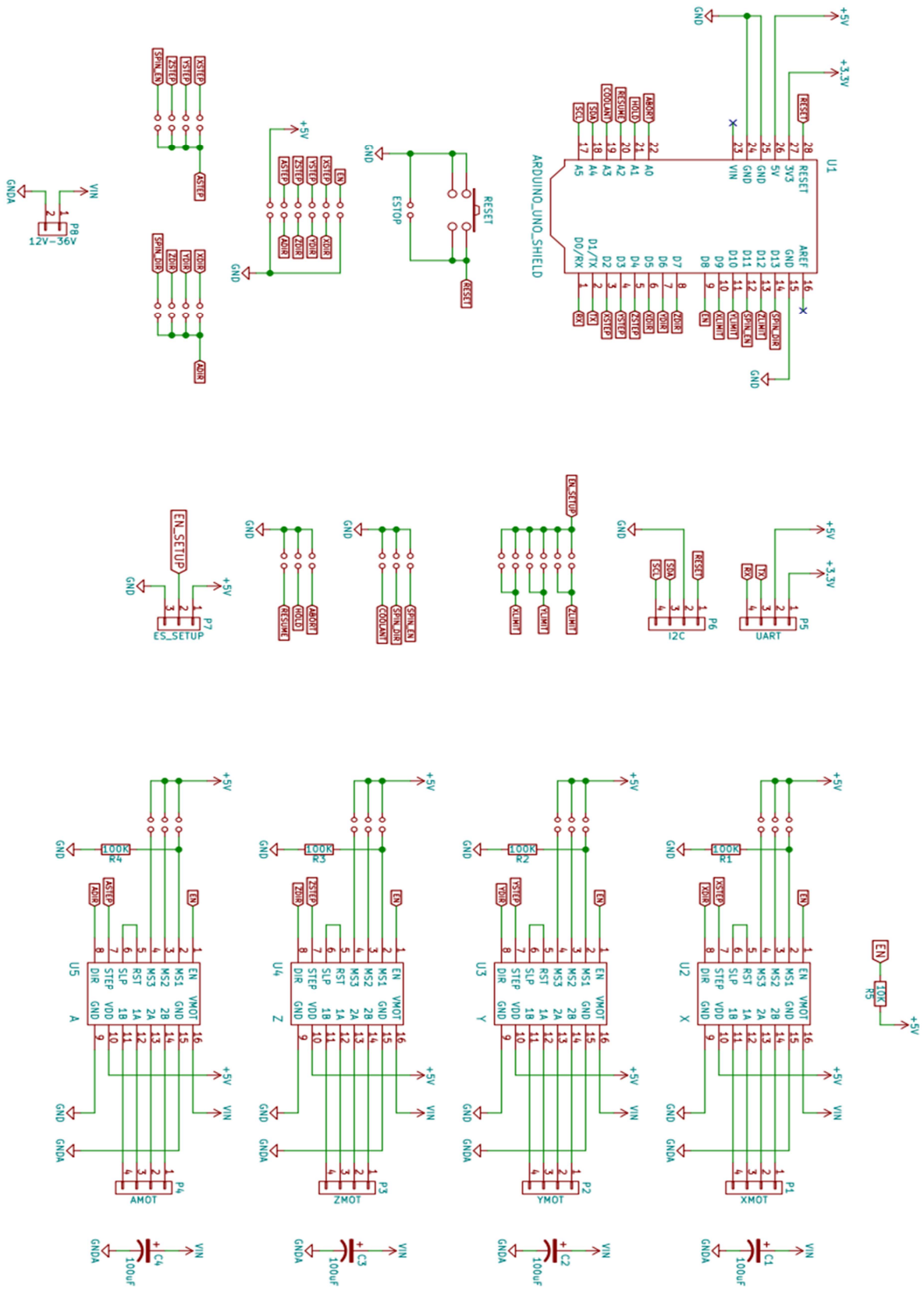


Figura 52. Esquema electrónico CNC Shield.

Para observarla con más detalle procedemos a analizarla parte por parte. Primero nos encontramos con un elemento conocido:

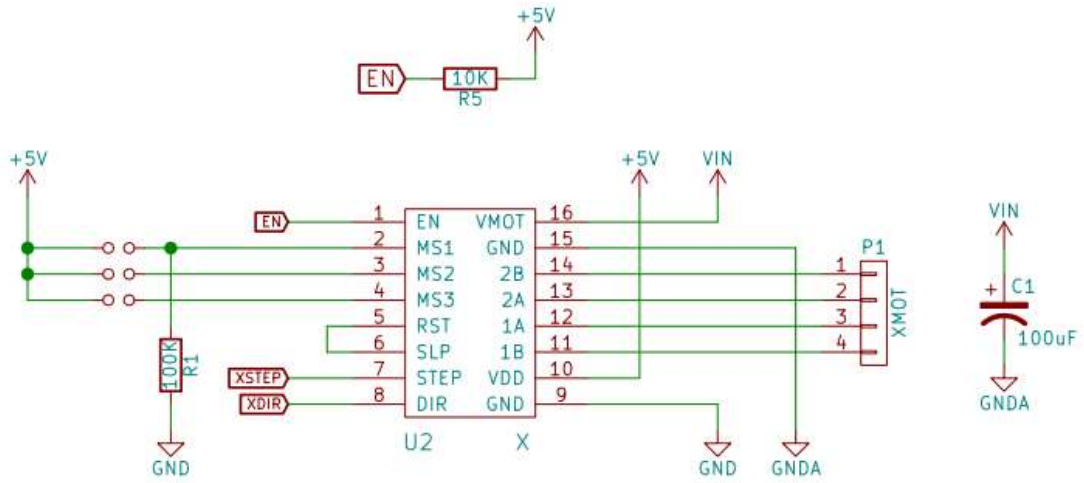


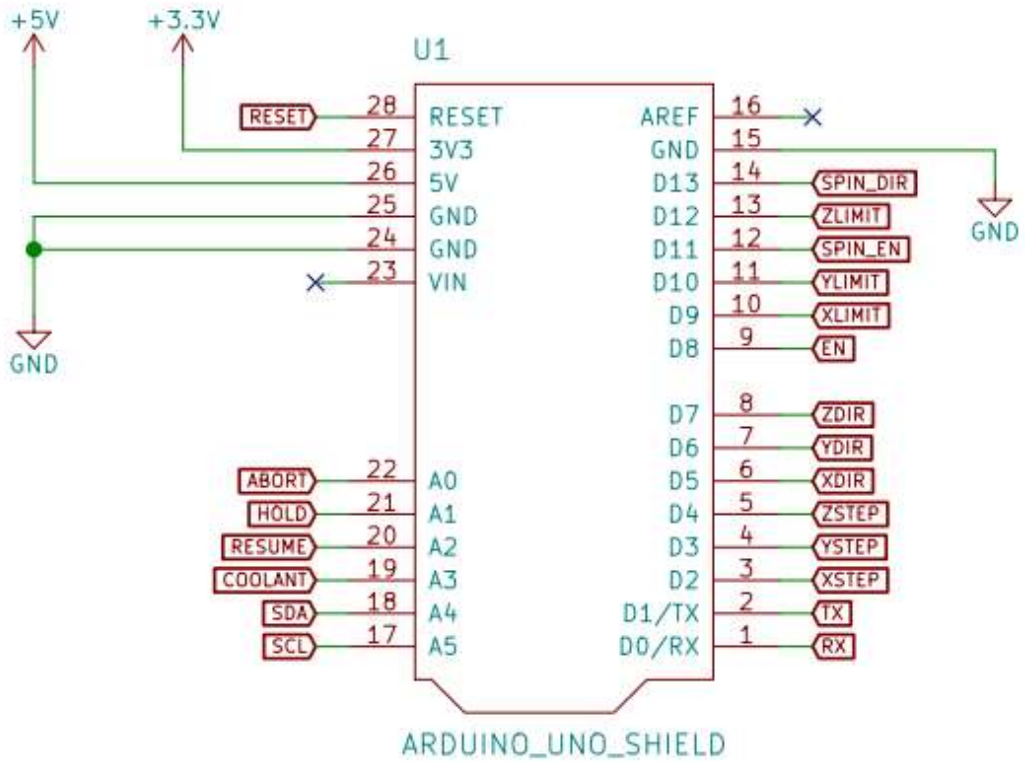
Figura 53. Esquema electrónico A4988.

Efectivamente, este es el esquema de un controlador A4988, donde nos indica como están conectados sus pines. Esto nos resulta familiar ya que hice una prueba un un key stepper y básicamente tiene el mismo funcionamiento, podemos ver, por ejemplo, los pines 2B, 2A, 1A y 1B, que van directamente al acople del motor.

Por tanto, la única diferencia sería en los pines que nos marcan los pasos y la dirección del motor (STEP y DIR). Este esquema está repetido 4 veces una por cada eje, el cuarto eje podemos crear uno totalmente independiente o simplemente clonarlo a partir de otro eje que tengamos ya sea X, Y, o Z.

Este cuarto eje lo utilizaremos para clonar el eje X y así dotar al sistema de un mayor equilibrio. Cabe aclarar que cuándo me refiero a clonar simplemente es que cuando pongamos en funcionamiento 1 eje se verá compuesto por dos motores que recibirán las mismas órdenes desde nuestro programa, moviéndose a la vez.

Este es el esquema del Arduino y en ella nos indica las conexiones de cada pin.



Como podemos ver al utilizar la CNC shield nos quedan muy pocos pines libres, pero que en nuestro caso no necesitamos de más. Por eso en otro tipo de proyectos es mejor la utilización de otro tipo de placa de Arduino.

Con esto terminamos con la extensión CNC para nuestro proyecto, en resumen, este tipo de placas se utilizan normalmente por el ahorro de tiempo y cableado que nos supone la adquisición de una.

Esto es debido a que en su circuito impreso se realizan todas las conexiones necesarias que de otra manera hubieran sido cables que van del Arduino hasta los controladores creando una malla de conexiones muy poco clara y económica.

Finalmente, en aplicaciones industriales de mayor escala esta extensión pasa a ser directamente una placa integrada, siendo esta mucho más cara, pero con muchas novedades respecto a nuestra extensión CNC.

7.5 Finales de carrera

Estos componentes electrónicos son los encargados de que nuestra máquina entienda su posición sobre el espacio. Cuando cargamos el código e iniciamos nuestro robot, el propio sistema cree que dónde parte es la posición 0, 0, 0, hecho que pudiera estar equivocado y restarle precisión a la máquina. Entonces, para determinar la posición inicial recurrimos a estos elementos.

Figura 54. Final de carrera

Tenemos 3 pines:

1. **C o COM**. Este es el común, el que deberemos de soldar sin importar la configuración del final de carrera
2. **NO**. Normally open, normalmente abierto. El botón impide que pase corriente y en el momento que pulsamos se cierra dejando pasar la corriente.
3. **NC**. Normally closed, normalmente cerrado. Al contrario que el anterior este se mantiene cerrado hasta que pulsamos el botón y se abre.

La elección de nuestra configuración será la de NC ya que en caso de averías o desconexiones indeseadas de los cables Arduino detecta esto y automáticamente deja de alimentar a nuestro circuito.

En el caso de NO la máquina seguiría sin ningún límite ya que en el caso anterior el Arduino no detectaría que hay algún cable desconectado, produciendo daños en nuestro sistema.

7.6 Fuente de alimentación

Utilizaremos una fuente de alimentación de 12V y 8,5A en la salida para alimentar nuestros motores a través de la conexión en la CNC shield.

Figura 55. Fuente de alimentación

Esta fuente de alimentación es perfecta para nuestra maqueta ya que dispone de la potencia necesaria para mover todos los motores PaP.

8. Parte de control (GRBL)

Por último, vamos a tratar el tema de la programación en Arduino y el tipo de código que vamos a utilizar para controlar nuestra máquina CNC.

Como explicamos anteriormente, el fundamento de los motores paso a paso es sencilla, cada pulso que el Arduino manda al motor este avanza un paso. Esto resulta ser una ventaja ya que de esta forma podemos saber exactamente cuánto se ha movido el motor en pulsos y así determinar su posición.

Esto en la práctica no es suficiente ya que en el mundo real no hay nada que se mida mediante pulsos, así que necesitamos una manera con la que poder comunicarse mediante milímetros.


```
; generated by WitBoxGUI
M107
M190 S45 ; set bed temperature
M104 S175 ; set temperature
G28 ; home all axes
G1 Z5 F5000 ; lift nozzle
M109 S175 ; wait for temperature to be reached
G21 ; set units to millimeters
G90 ; use absolute coordinates
M82 ; use absolute distances for extrusion
G92 E0
G1 Z0.400 F7800.000
G1 E-2.00000 F2400.00000
G92 E0
G1 X50.367 Y88.236 F7800.000
G1 E2.00000 F2400.00000
G1 X52.527 Y87.420 E2.27422 F1800.000
G1 X53.784 Y87.301 E2.42419
G1 X146.216 Y87.301 E13.40185
G1 X148.490 Y87.698 E13.67607
G1 X150.496 Y88.842 E13.95029 F1800.000
G1 X151.995 Y90.598 E14.22452
G1 X152.811 Y92.758 E14.49874
```

Figura 56. G-CODE.

Es cuando nos encontramos con el lenguaje G-CODE. Este lenguaje de programación es el más utilizado y a la vez uno de los primeros en desarrollarse, se basa principalmente en traducir nuestras órdenes de una forma que el robot las entienda, cómo por ejemplo donde ir, que trayectoria debe seguir y a que velocidad tiene que hacer dicho proceso.

La mayoría de los programas que utilizan este tipo de código suelen tener alguna forma de exportar y generar G-CODE sin necesidad de tener ningún conocimiento sobre este.

Podemos utilizar programas cómo el Inkscape, un programa de gráficos vectoriales libre, para realizar formas, dibujos, logotipos y, mediante un plugin, transformarlo en G-CODE.

Si necesitamos un firmware que tenga aceleraciones y deceleraciones suaves, el G-CODE es nuestra solución ya que en el momento de reposo de los motores vence a su inercia acelerando poco a poco.

Plugin: complemento o plug-in, es un programa informático que se relaciona con otra para agregarles nuevas funcionalidades, a veces, muy específicas.

El G-CODE hace que cada línea cada segmento de movimientos se convierta en una lista de trapecios. Esta lista de trapecios se envía al Arduino convirtiéndolos en pulsos que son enviados al motor.

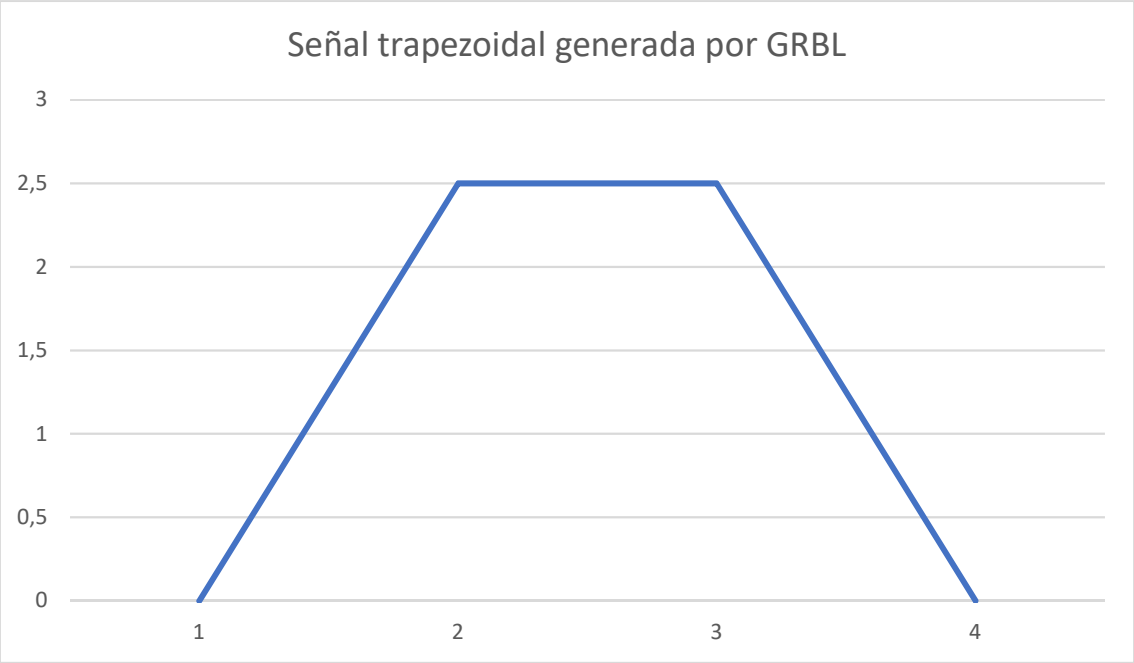


Figura 57. Señal trapezoidal emitida por GRBL.

La aceleración o deceleración en nuestro sistema tiene mucha importancia ya que necesitamos que la máquina se mueva con precisión. En las fresadoras tipo CNC cuando necesitamos que las líneas sean prácticamente perfectas, necesitamos algún tipo de solución que nos garantice que, por ejemplo, se puedan realizar líneas con giros bruscos y movimientos complejos.

Para esto el GRBL utiliza el "*look ahead*", esto hace que el programa lea líneas de código antes de que llegue a producirse ese movimiento para así poder anticiparse y recalcular su trayectoria, aceleración o deceleración.

Por lo que podemos observar este software nos permite lograr con sencillez y rapidez una precisión más que aceptable. Aunque en nuestro caso se trata de una máquina "*pick n'place*" que puede que en un principio no sea tan importante la precisión que como lo puede ser en una fresadora, pero aun así para recoger piezas en el espacio se necesita de un movimiento suave y concreto. Porque con algún fallo en la trayectoria podríamos no recoger bien una pieza y que esta se caiga causando un problema en toda la línea de producción.

En el ámbito industrial lo que vemos son ordenadores enteros que trabajan exclusivamente para el funcionamiento de una máquina CNC, dotándola de más procesamiento de cálculo y por tanto siendo mucho más precisa.

En este trabajo no vamos a profundizar mucho en este lenguaje, pero sí que describiremos los códigos más esenciales para saber que está haciendo el software cuando este mueva los motores.

Esta placa es totalmente compatible con el IDE (entorno de desarrollo del Arduino) de Arduino y por un precio más económico que el original. Por lo tanto, el primer paso para comprobar que tiene un funcionamiento correcto será descargarnos el IDE y comprobar que nuestro PC reconozca la placa. Para esto iniciamos el programa y asignamos el puerto donde está conectada la placa (COM3), para finalizar cargamos un pequeño programa almacenado como ejemplo en una librería del propio software de Arduino, llamado 'blink' que simplemente hace parpadear un led de nuestra placa.

Con esto estaremos preparados para utilizar nuestro Arduino y empezar a enviarle programas y órdenes.

Para empezar con la instalación del GRBL tendremos que instalarnos el archivo .zip y descomprimirlo en nuestro ordenador donde lo que haremos será añadir la librería que hemos descargado y cargarlo a nuestro Arduino. Esto es lo que debería salirnos cuando subamoselGRBL:

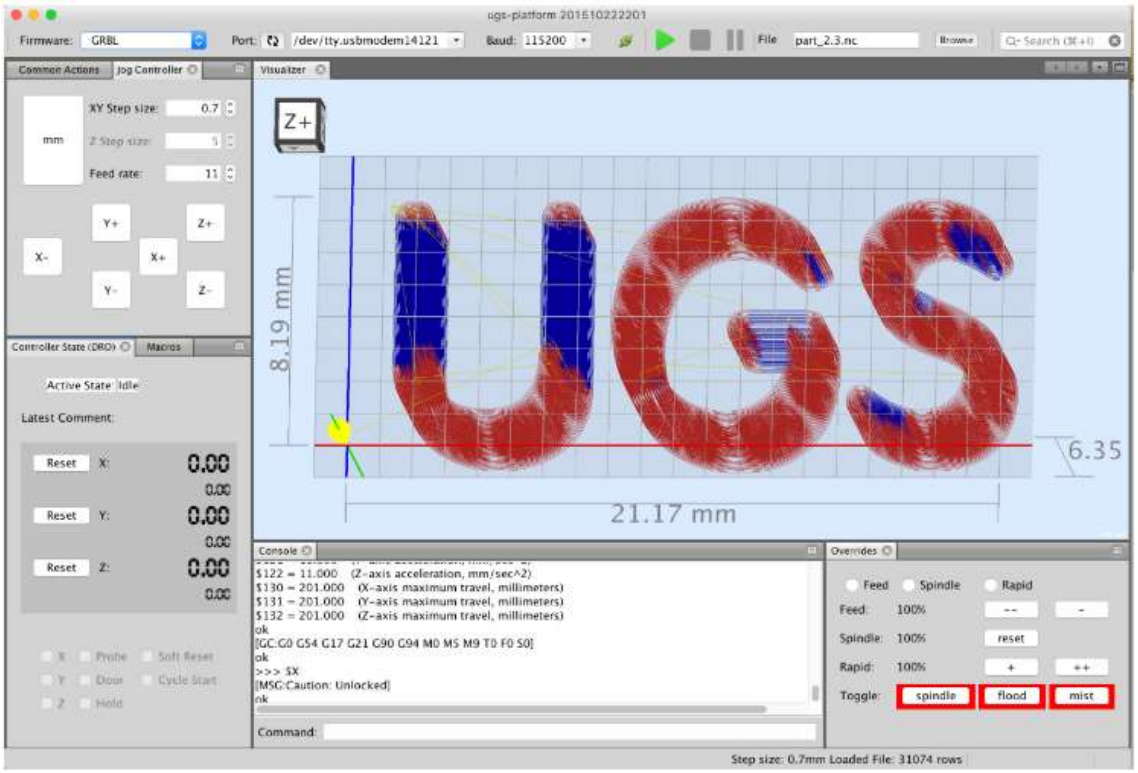
Figura 58. IDE Arduino.

Simplemente nos indica unos sencillos pasos para asegurarnos de que el programa vaya a tener un funcionamiento correcto. Ahora mismos sólo contamos con el firmware, pero necesitamos una interfaz capaz de comunicar al Arduino lo que queremos que haga, para esta tarea utilizaremos el Universal G-CODE Sender.

8.1 Universal G-CODE sender

Una plataforma gcode con todas las funciones que se utiliza para interactuar con controladores CNC avanzados como GRBL. Universal Gcode Sender es una aplicación Java autónoma que incluye todas las dependencias externas, lo que significa que, si tiene la configuración de Java Runtime Environment, UGS proporciona el resto.

Esto hace que la instalación sea más sencilla ya que no es necesario instalar nuevas librerías.



Este software cuenta con una previsualización del programa a realizar muy útil, así como otras características:

- Capacidad de trabajar en diferentes sistemas operativos ya sea Windows, OSX, Linux o Raspberry Pi.
- Señalización y visualización 3D en tiempo real con feedback de posición.
- Nos indica la duración del proceso.
- Incluye unos comandos especiales para modificar los diferentes parámetros como pueden ser la velocidad, velocidad de paso…

9. Presupuesto

En este apartado se expone el estudio económico del desarrollo de nuestra maqueta, principalmente el coste en horas para la construcción del modelo y el coste de sus componentes. Lo hemos dividido los componentes en según a que partes pertenecían:

Parte eléctrica	Precio	Unidades	Total
Arduino UNO R3 compatible CH340	3,39 €	1	3,39 €
Controladores A4988	3,40 €	4	13,60 €
CNC Shield	3,70 €	1	3,70 €
Motor paso a paso Nema 17 0,8 A	10,44 €	4	41,76 €
Interruptor de fin de carrera	1,30 €	6	7,80 €
Fuente de alimentación 12V 8,5A	13,29 €	1	13,29 €
		Total parte eléctrica	83,54 €

Parte mecánica	Precio	Unidades	Total
Varilla roscada 3D T8 de 500mm	8,83 €	2	17,66 €
Varilla roscada 3D T8 de 400mm	6,92 €	1	6,92 €
Eje lineal, 8x500mm varilla de acero	7,84 €	2	15,68 €
Eje lineal, 8x400mm varilla de acero	5,97 €	1	5,97 €
Tornillos M2 y M3 con tuerca	5,39 €	1	5,39 €
Rodamiento 8mm 608ZZ 8x22x7	0,44 €	3	1,32 €
Cojinete de acero para CNC 8mm	1,79 €	3	5,37 €
Acople para ejes OD 19mm	1,77 €	3	5,31 €
		Total parte mecánica	63,62 €

10. Trabajos futuros

Este proyecto se realiza con el objetivo de asemejarse a un robot cartesiano paletizador industrial. Al ser un prototipo y sobre todo siendo nuestra primera vez realizando este tipo de proyecto, podemos llegar a desarrollar posibles mejoras en todas sus partes. Este apartado recoge algunas posibles modificaciones que podrían aumentar su durabilidad y rendimiento.

La realización de la maqueta se ha llevado a cabo de la manera más sencilla y eficiente posible, tanto económicamente como estructuralmente. Proponemos algunas mejoras respecto a la maqueta realizada cómo también su aplicación futura a un proceso industrial.

Primero que todo tenemos que saber que las mejoras que puedan aplicarse a la maqueta tienen una repercusión directa a su aplicación industrial.

En primer lugar, la estructura del modelo se ha realizado con piezas realizadas mediante una impresora 3D, estas piezas poseen una buena resistencia mecánica además de adaptarse perfectamente a nuestras necesidades.

- Una alternativa y posible mejora sería la realización de estas piezas mediante **acero**, un material mucho más resistente que el plástico de las piezas, para así dotar el sistema de una fiabilidad superior.

Otro apartado esencial son los motores. Hemos utilizado motores paso a paso de una determinada potencia pensando en las necesidades de su utilización, pero esto no significa que sea eficiente. Con esto nos referimos a que no estemos utilizando los motores a pleno rendimiento y tengamos pérdidas económicas ya que posiblemente estemos utilizando más energía de la que realmente necesitamos. Por esto propongo lo siguiente:

- Dotar a los motores de unos medidores de intensidad para controlar que todos funcionan correctamente y que ninguno tiene fallos o esté consumiendo demasiado. Esta solución se emplea mucho en aplicaciones industriales donde los motores están en constante funcionamiento.

Por otro lado, tenemos la parte de control que la hemos realizado mediante componentes económicos y de fácil acceso, pero no por esto eran las únicas:

- Arduino es un buen controlador por su facilidad su entorno, pero para ciertas aplicaciones se puede quedar corto. Entonces para aplicaciones de más potencia y precisión podríamos utilizar una Raspberri pi algo más complejo que Arduino ya que se trata casi de un ordenador.

11. Conclusiones

El objetivo de este proyecto de final de carrera consistía en desarrollar una maqueta de un robot cartesiano tipo CNC.

Empezamos buscando información sobre este tipo de sistemas. Enseguida encontramos toda la información necesaria, ya que descubrimos que cuenta con una comunidad enorme y que nos facilitó la adquisición de conocimientos totalmente necesarios.

Seguido de esto empezamos a probar configuraciones. Al principio teníamos pensado en hacer la parte de transmisión mediante correas, pero su complejidad a la hora de trabajar con correas nos hizo pensar en cambiar el tipo de transmisión. Es cuando nos encontramos con el husillo, no era mejor que las correas, pero si que podía ofrecernos ventajas. Su sencillez de acople a los motores y la cantidad de elementos que estaban diseñados para el husillo nos hizo pensar en realizar todo el sistema con esto.

Una vez determinada el tipo de transmisión pasamos a elaborar el diseño de la infraestructura. Antes de empezar la infraestructura tuvimos que saber que elementos necesita para que nuestro robot funcione, esto es, su parte mecánica, eléctrica y de control. Una vez decidido todo esto y después de la realización y descarte de algunos bocetos procedimos a elaborar la infraestructura.

Desde un primer momento me decidí en realizar estas piezas en impresora 3D, pero para esto antes deberíamos a aprender modelaje en 3D. Descubrimos el FreeCAD y fue cuestión de tiempo aprender.

Una vez realizado todas las piezas, algunas más problemáticas que otras, procedimos al montaje de todo el sistema, motores, husillos, tornillería… La parte de electrónica me resultó más sencilla porque había muchos conceptos de los cuales había adquirido en la carrera. Determinamos que potencia necesitábamos y nos dispusimos a desarrollar la parte de control.

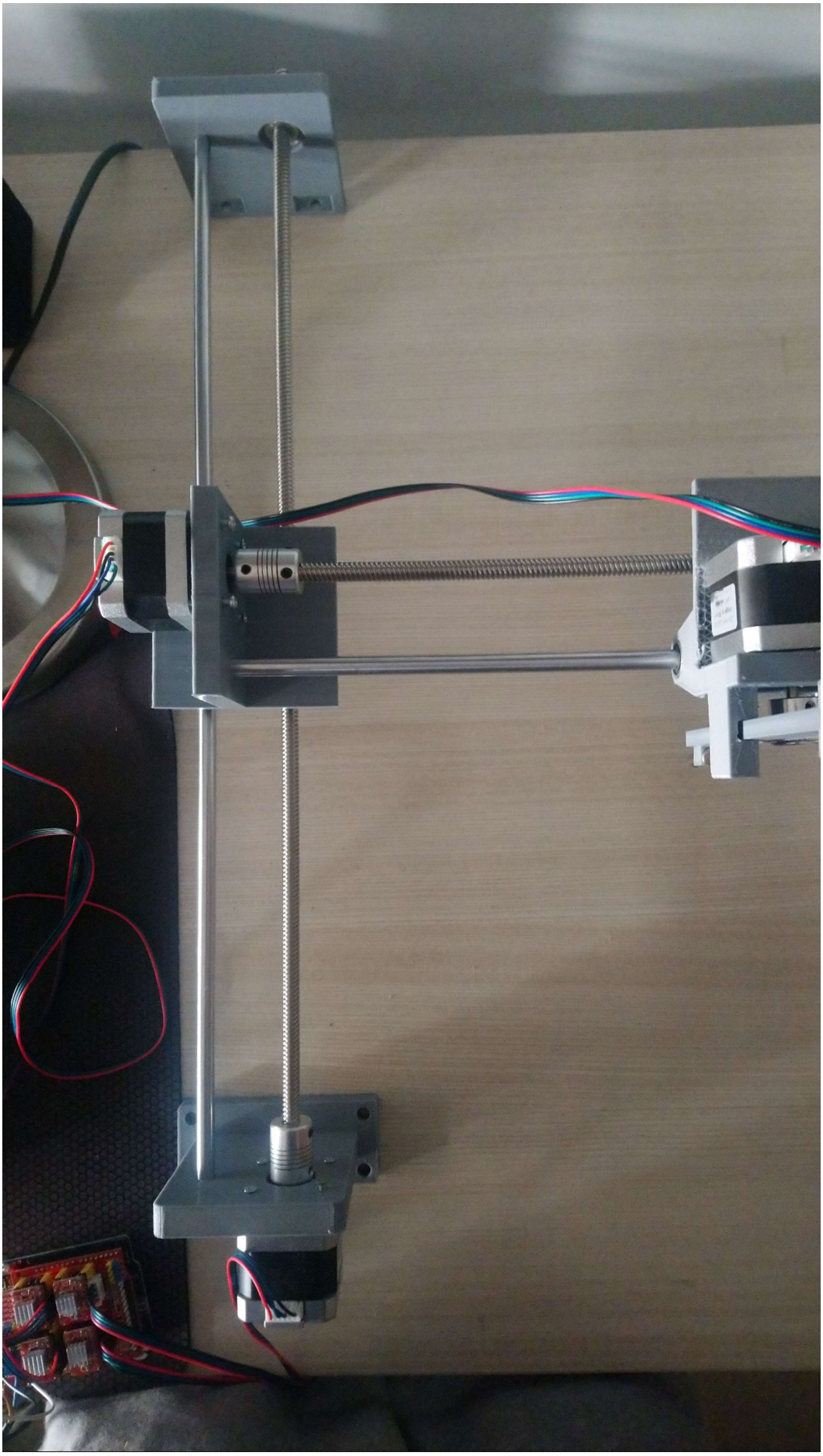
Por último, instalé el software necesario (Arduino y GRBL) para controlar este robot. Tras algunos intentos fallidos por compatibilidad de ordenadores al fin conseguimos cargar el firmware GRBL en nuestra placa de Arduino. Con todo esto hecho ya sólo nos quedaba realizar un programa para visualizar y comprobar que el robot podía moverse en los tres ejes.

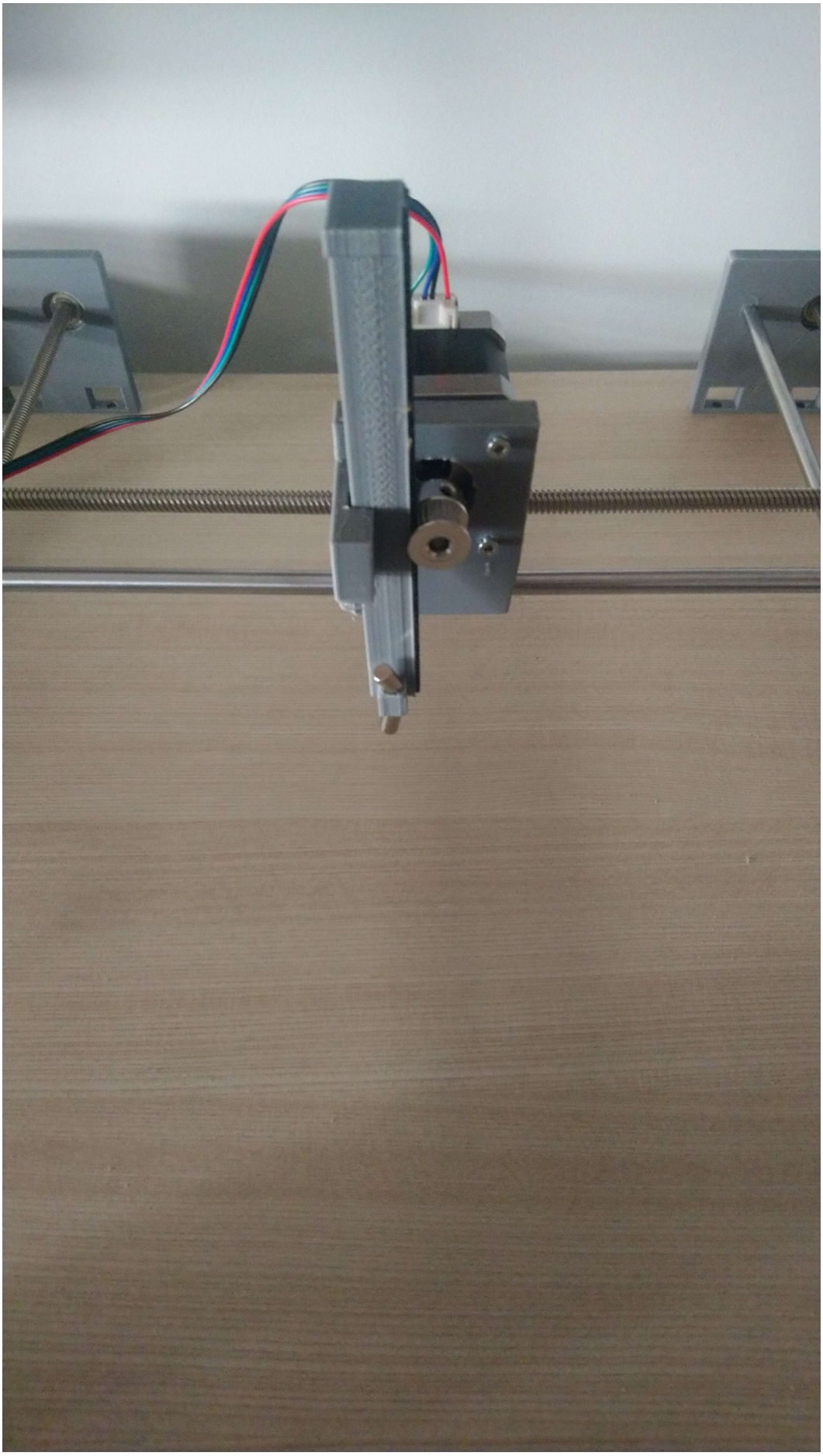
Cómo conclusión personal he realizado un proyecto por mi propia cuenta sin haber hecho nunca algo parecido siendo el resultado satisfactorio. He podido aprender modelaje 3D, Arduino y otro tipo de cosas cómo mecanismos, todo con el objetivo de la realización de un robot CNC cartesiano paletizador.

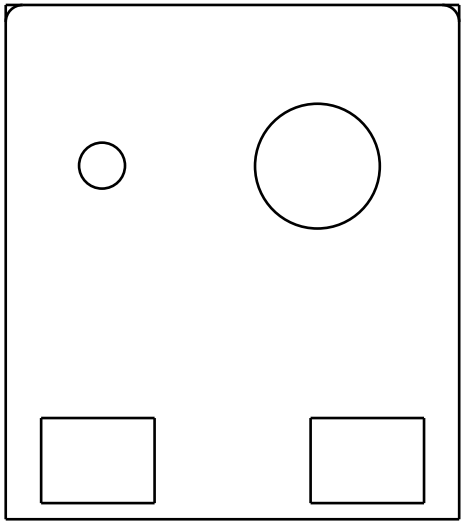
12. Bibliografía

1. "Sitio web Errece" http://www.errece.es/productos/maxportic_paletizado_p9.php
2. "Sitio web robot casrtesiano Omron" https://industrial.omron.es/es/solutions/product-solutions/cartesian-robotic-solution
3. "Robot cartesiano" https://es.wikipedia.org/wiki/Robot_de_coordenadas_cartesianas
4. "Sitio web e-ika, productos empleados" https://www.e-ika.com
5. "Sitio web FreeCAD" https://www.freecadweb.org/
6. "Sitio web Staticboards, motores" https://www.staticboards.es/blog/dominar-motor-paso-a-paso-con-grbl/
7. "Sitio web Nema 17" https://descargas.cetronic.es/NEMA17.pdf
8. "Sitio web características anycubic i3 mega" https://all3dp.com/es/1/anycubic-i3-mega-impresora-3d-analisis/
9. "Sitio web Simplify3D" https://www.simplify3d.com/
10. "Blog Staticboards controladores" https://www.staticboards.es/blog/drv8825-vs-a4988/
11. "Sitio web Polulu A4988" https://www.pololu.com/product/1182
12. "Información Arduino" https://es.wikipedia.org/wiki/Arduino
13. "Sitio web Arduino" https://www.arduino.cc/
14. "Sitio web CNC Shield" https://blog.protoneer.co.nz/arduino-cnc-shield/
15. "Información G-CODE" https://es.wikipedia.org/wiki/G-code
16. "Sitio web github, GRBL" https://github.com/grbl/grbl
17. "Sitio web github, UGS" https://winder.github.io/ugs_website/#universal-gcode-sender

ANEXOS

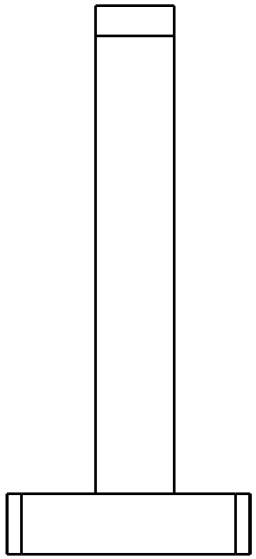


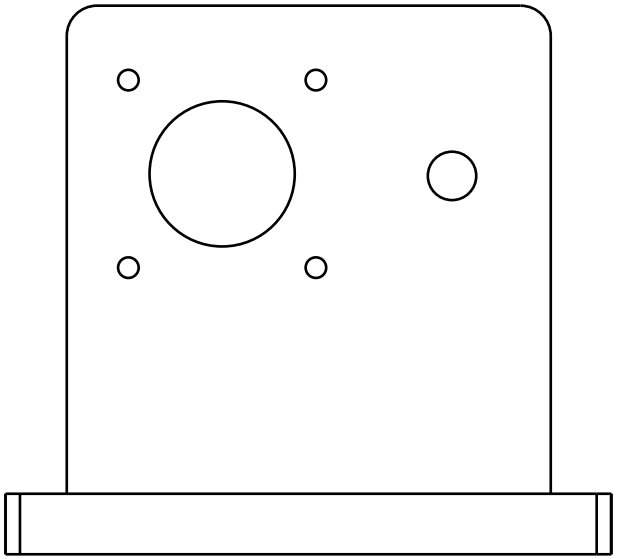


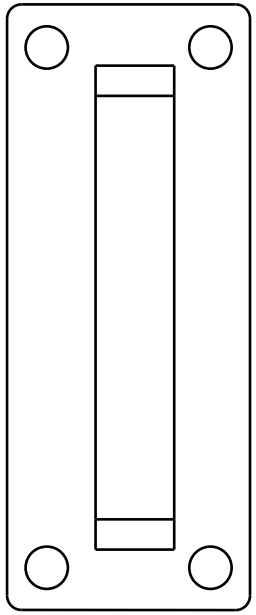


Created by:	Title:	Size:	Sheet:	Scale:
Ferran Bernad Aguilar	Soporte final eje X	A4	2/4	1/0.8

Supplementary information:	Part number:	
FreeCAD DRAWING	Drawing no.:	DN
	Date: 7/06/2019	Revision: REV A







Autor:	Titulo:	Tamaño:	Página:	Escala:
Ferran Bernad Aguilar	Soporte motor eje X	A4	1/ 4	0.8

Supplementary information:

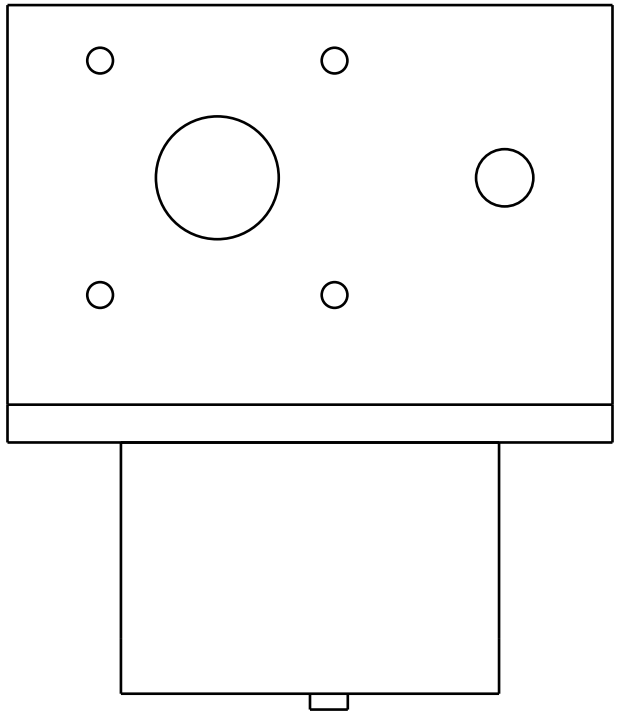
FreeCAD DRAWING

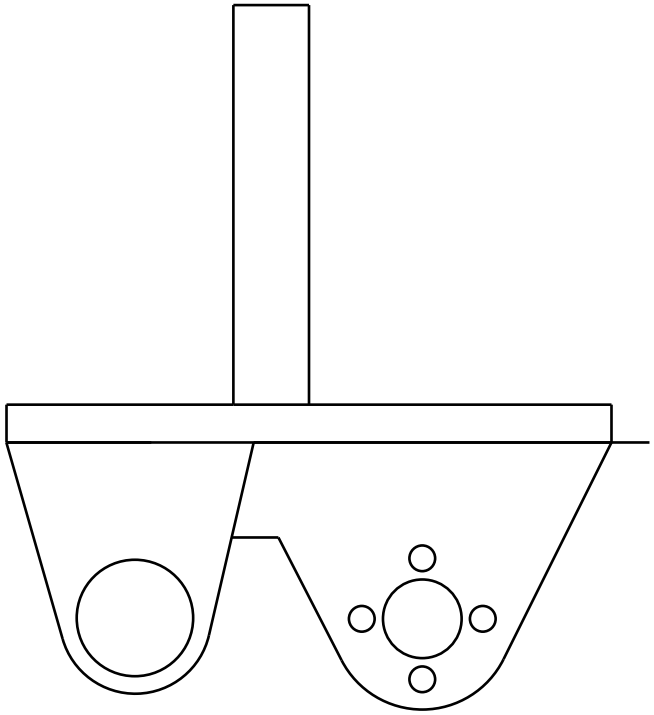
Part number: PN

Drawing no.: DN

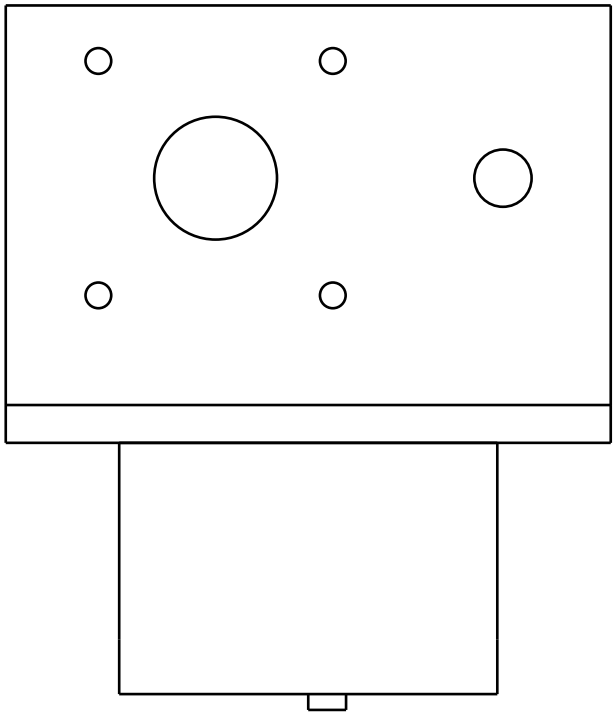
Date: 15/06/2019

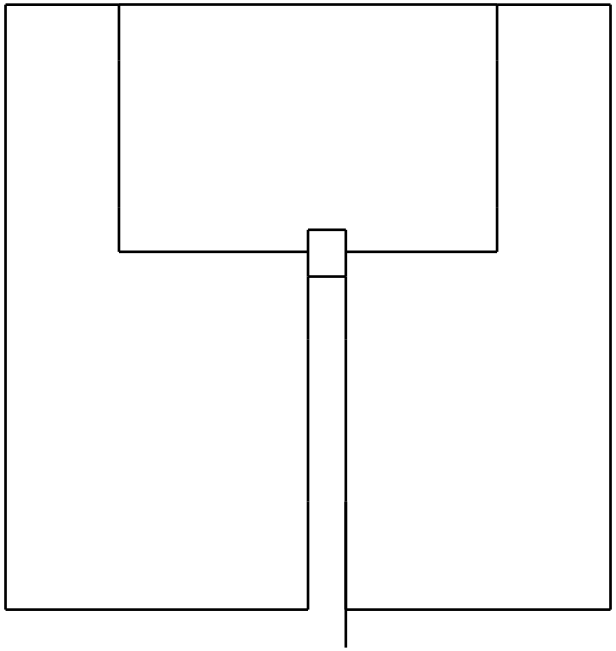
Revision: REV A



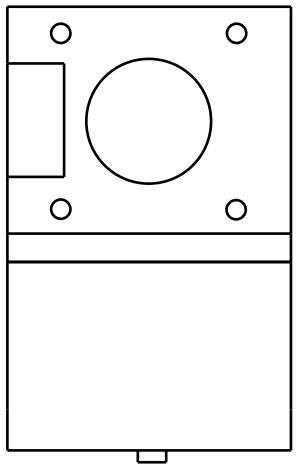


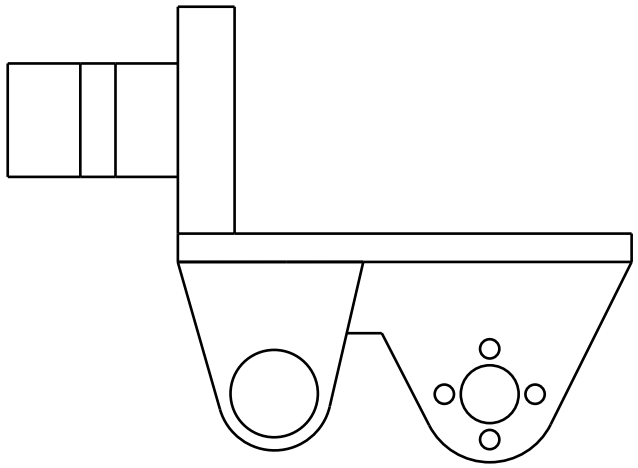
Created by: Ferran Bernad Aguilar	Title: Carrete eje y	Size: A4	Sheet: 3/ 4	Scale: 1/0.8
Supplementary information: FreeCAD DRAWING		Part number: PN Drawing no.: DN		
		Date: 3/06/2019		Revision: REV A

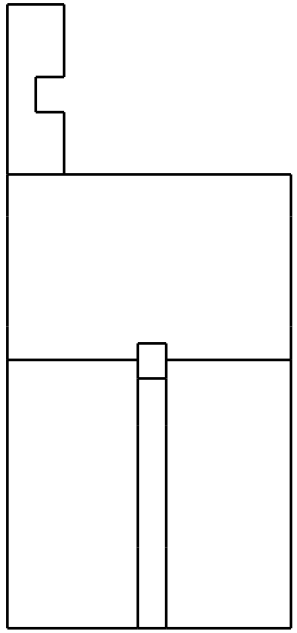




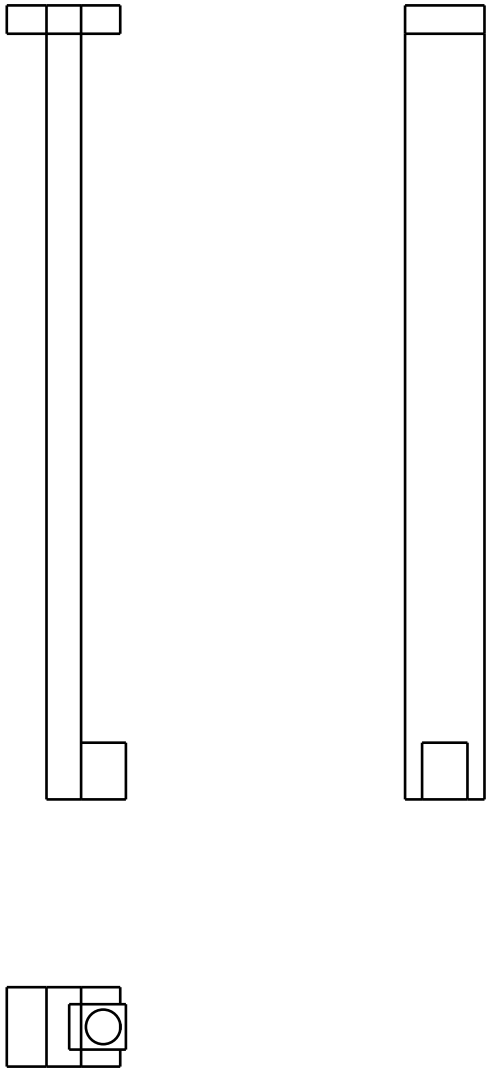
Created by:	Title:	Size:	Sheet:	Scale:
Ferran Bernad Aguilar	Carrete eje y	A4	4 / 4	1/0.8
Supplementary information:		Part number: PN		
FreeCAD DRAWING		Drawing no.: DN		
		Date: 3/06/2019		Revision: REV A







DESIGNED BY: Ferran Bernad Aguilar DATE: 2/07/2019		Carrete eje Z		G _
SIZE A4				F _
				E _
				D _
SCALE 1	WEIGHT (kg)	DRAWING NUMBER 1/2	SHEET 1	C _
				B _
This drawing is our property; it can't be reproduced or communicated without our written consent.				A _



DESIGNED BY: Ferran Bernad Aguilar DATE: 03/07/2019		Cremallera		G _
SIZE A4				F _
				E _
				D _
SCALE	WEIGHT (kg)	DRAWING NUMBER	SHEET 2/2	C _
				B _
This drawing is our property; it can't be reproduced or communicated without our written consent.				A _

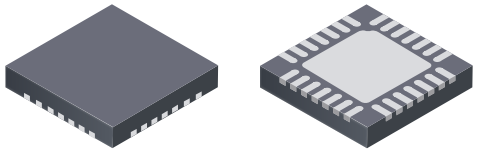
DMOS Microstepping Driver with Translator And Overcurrent Protection

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $^1/_2$, $^1/_4$, $^1/_8$, and $^1/_{16}$

Package:

28-contact QFN
with exposed thermal pad
5 mm × 5 mm × 0.90 mm
(ET package)



Approximate size

Description

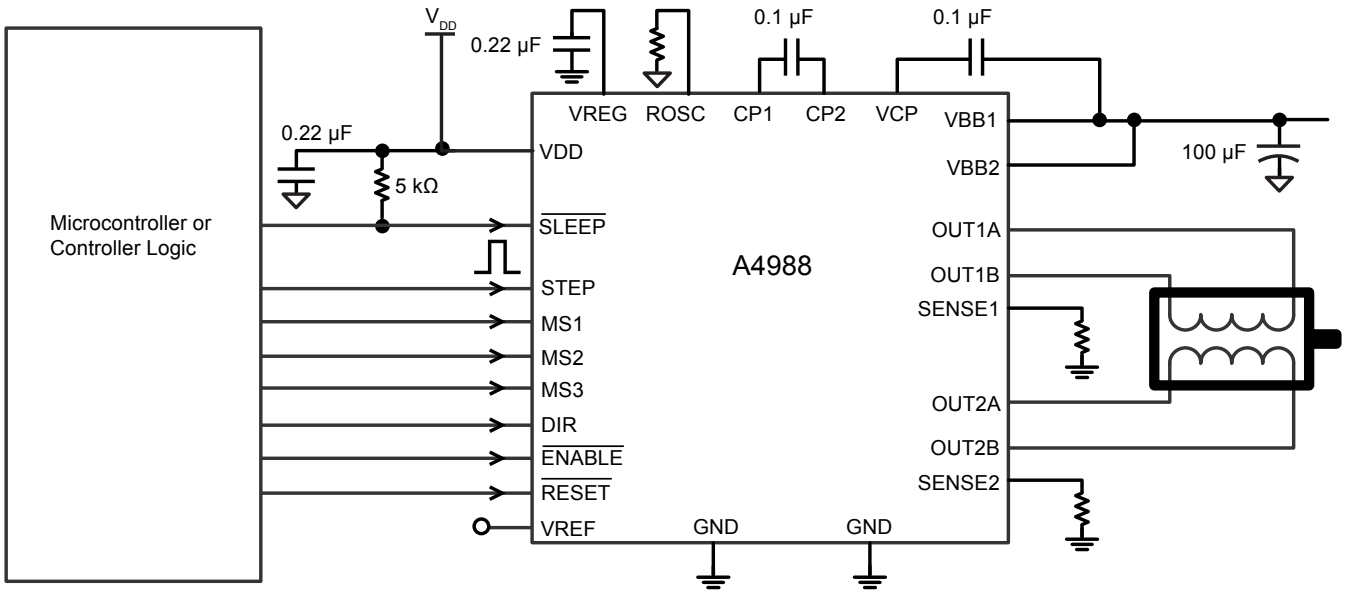
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ±2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

Continued on the next page…

Typical Application Diagram



Description (continued)

Internal synchronous rectification control circuitry is provided to improve power dissipation during PWM operation. Internal circuit protection includes: thermal shutdown with hysteresis, undervoltage lockout (UVLO), and crossover-current protection. Special power-on sequencing is not required.

The A4988 is supplied in a surface mount QFN package (ES), 5 mm × 5 mm, with a nominal overall package height of 0.90 mm and an exposed pad for enhanced thermal dissipation. It is lead (Pb) free (suffix –T), with 100% matte tin plated leadframes.

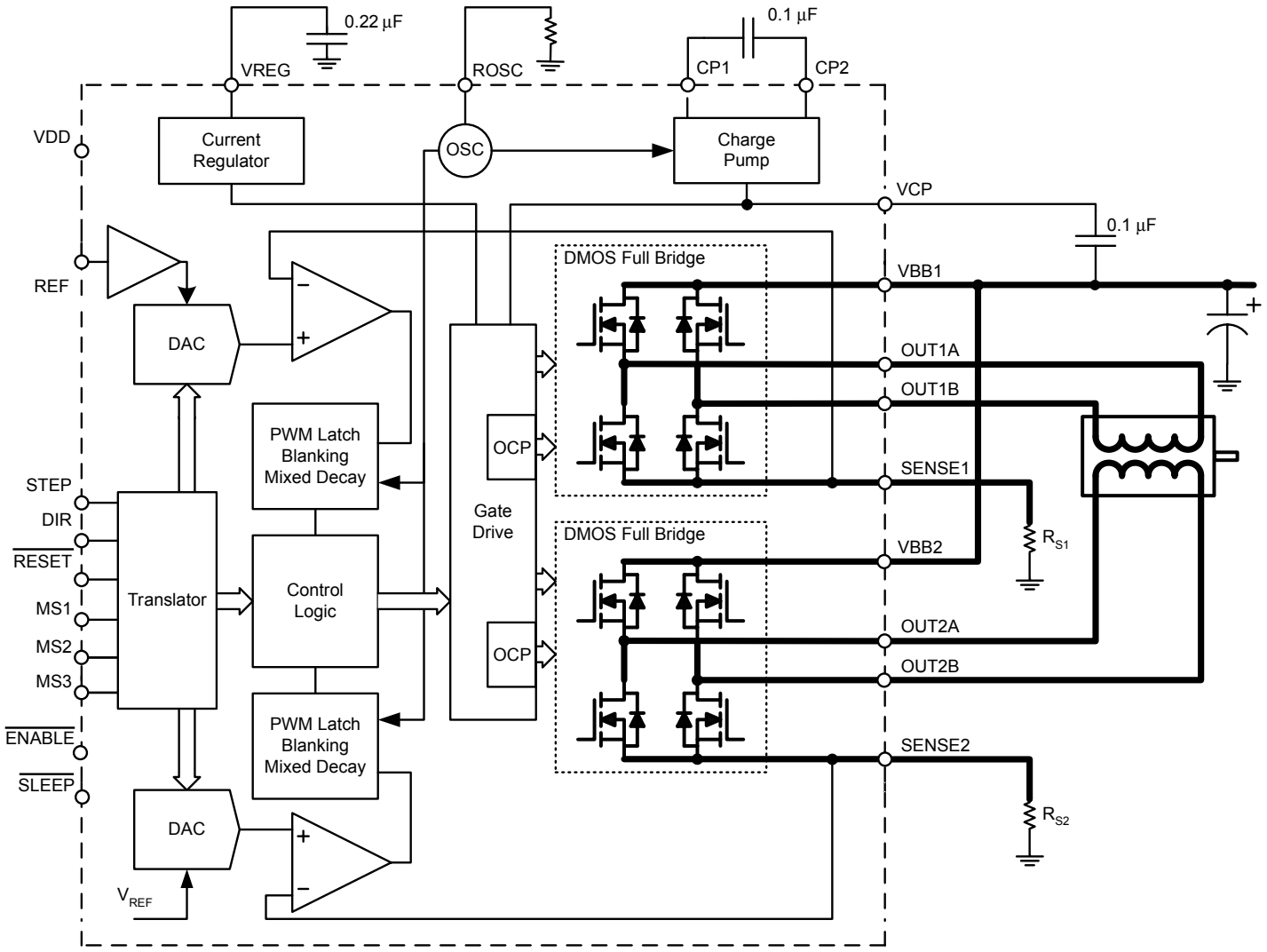
Selection Guide

Part Number	Package	Packing
A4988SETTR-T	28-contact QFN with exposed thermal pad	1500 pieces per 7-in. reel

Absolute Maximum Ratings

Characteristic	Symbol	Notes	Rating	Units
Load Supply Voltage	V_{BB}		35	V
Output Current	I_{OUT}		±2	A
Logic Input Voltage	V_{IN}		–0.3 to 5.5	V
Logic Supply Voltage	V_{DD}		–0.3 to 5.5	V
Motor Outputs Voltage			–2.0 to 37	V
Sense Voltage	V_{SENSE}		–0.5 to 0.5	V
Reference Voltage	V_{REF}		5.5	V
Operating Ambient Temperature	T_A	Range S	–20 to 85	ºC
Maximum Junction	T_J(max)		150	ºC
Storage Temperature	T_{stg}		–55 to 150	ºC

Functional Block Diagram



ELECTRICAL CHARACTERISTICS[1] at T_A = 25°C, V_{BB} = 35 V (unless otherwise noted)

Characteristics	Symbol	Test Conditions	Min.	Typ.[2]	Max.	Units
Output Drivers						
Load Supply Voltage Range	V_{BB}	Operating	8	–	35	V
Logic Supply Voltage Range	V_{DD}	Operating	3.0	–	5.5	V
Output On Resistance	R_{DSON}	Source Driver, I_{OUT} = –1.5 A	–	320	430	mΩ
		Sink Driver, I_{OUT} = 1.5 A	–	320	430	mΩ
Body Diode Forward Voltage	V_F	Source Diode, I_F = –1.5 A	–	–	1.2	V
		Sink Diode, I_F = 1.5 A	–	–	1.2	V
Motor Supply Current	I_{BB}	f_{PWM} < 50 kHz	–	–	4	mA
		Operating, outputs disabled	–	–	2	mA
Logic Supply Current	I_{DD}	f_{PWM} < 50 kHz	–	–	8	mA
		Outputs off	–	–	5	mA
Control Logic						
Logic Input Voltage	$V_{IN(1)}$		V_{DD}×0.7	–	–	V
	$V_{IN(0)}$		–	–	V_{DD}×0.3	V
Logic Input Current	$I_{IN(1)}$	V_{IN} = V_{DD}×0.7	–20	<1.0	20	µA
	$I_{IN(0)}$	V_{IN} = V_{DD}×0.3	–20	<1.0	20	µA
Microstep Select	R_{MS1}	MS1 pin	–	100	–	kΩ
	R_{MS2}	MS2 pin	–	50	–	kΩ
	R_{MS3}	MS3 pin	–	100	–	kΩ
Logic Input Hysteresis	$V_{HYS(IN)}$	As a % of V_{DD}	5	11	19	%
Blank Time	t_{BLANK}		0.7	1	1.3	µs
Fixed Off-Time	t_{OFF}	OSC = VDD or GND	20	30	40	µs
		R_{OSC} = 25 kΩ	23	30	37	µs
Reference Input Voltage Range	V_{REF}		0	–	4	V
Reference Input Current	I_{REF}		–3	0	3	µA
Current Trip-Level Error[3]	err_I	V_{REF} = 2 V, %$I_{TripMAX}$ = 38.27%	–	–	±15	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 70.71%	–	–	±5	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 100.00%	–	–	±5	%
Crossover Dead Time	t_{DT}		100	475	800	ns
Protection						
Overcurrent Protection Threshold[4]	I_{OCPST}		2.1	–	–	A
Thermal Shutdown Temperature	T_{TSD}		–	165	–	°C
Thermal Shutdown Hysteresis	T_{TSDHYS}		–	15	–	°C
VDD Undervoltage Lockout	V_{DDUVLO}	V_{DD} rising	2.7	2.8	2.9	V
VDD Undervoltage Hysteresis	$V_{DDUVLOHYS}$		–	90	–	mV

[1]For input and output current specifications, negative current is defined as coming out of (sourcing) the specified device pin.

[2]Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits.

[3]V_{ERR} = [(V_{REF}/8) – V_{SENSE}] / (V_{REF}/8).

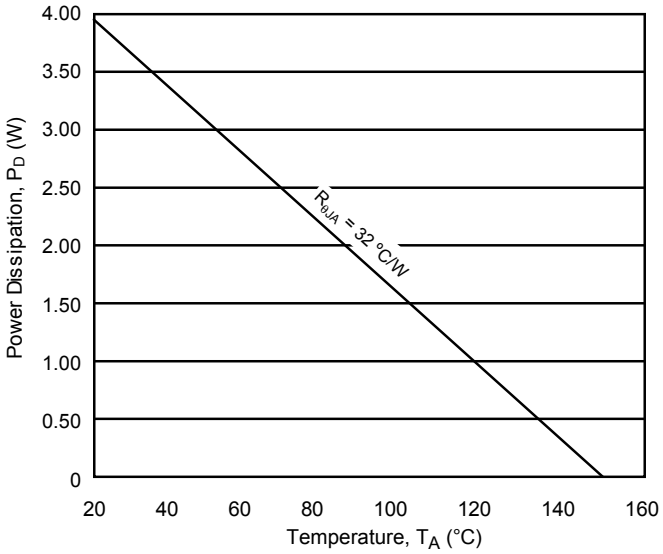
[4]Overcurrent protection (OCP) is tested at T_A = 25°C in a restricted range and guaranteed by characterization.

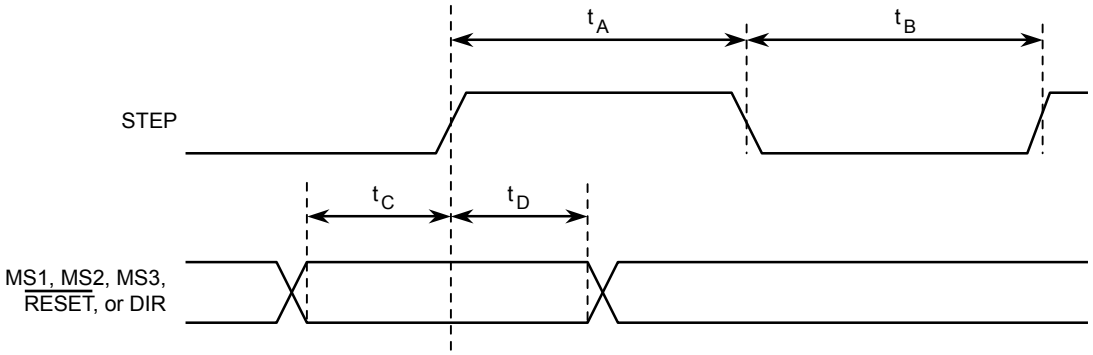
THERMAL CHARACTERISTICS

Characteristic	Symbol	Test Conditions*	Value	Units
Package Thermal Resistance	$R_{\theta JA}$	Four-layer PCB, based on JEDEC standard	32	ºC/W

*Additional thermal information available on Allegro Web site.

Power Dissipation versus Ambient Temperature





Time Duration	Symbol	Typ.	Unit
STEP minimum, HIGH pulse width	t_A	1	µs
STEP minimum, LOW pulse width	t_B	1	µs
Setup time, input change to STEP	t_C	200	ns
Hold time, input change to STEP	t_D	200	ns

Figure 1: Logic Interface Timing Diagram

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

Functional Description

Device Operation. The A4988 is a complete microstepping motor driver with a built-in translator for easy operation with minimal control lines. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth, and sixteenth-step modes. The currents in each of the two output full-bridges and all of the N-channel DMOS FETs are regulated with fixed off-time PWM (pulse width modulated) control circuitry. At each step, the current for each full-bridge is set by the value of its external current-sense resistor (R_{S1} and R_{S2}), a reference voltage (V_{REF}), and the output voltage of its DAC (which in turn is controlled by the output of the translator).

At power-on or reset, the translator sets the DACs and the phase current polarity to the initial Home state (shown in Figures 9 through 13), and the current regulator to Mixed Decay Mode for both phases. When a step command signal occurs on the STEP input, the translator automatically sequences the DACs to the next level and current polarity. (See Table 2 for the current-level sequence.) The microstep resolution is set by the combined effect of the MSx inputs, as shown in Table 1.

When stepping, if the new output levels of the DACs are lower than their previous output levels, then the decay mode for the active full-bridge is set to Mixed. If the new output levels of the DACs are higher than or equal to their previous levels, then the decay mode for the active full-bridge is set to Slow. This automatic current decay selection improves microstepping performance by reducing the distortion of the current waveform that results from the back EMF of the motor.

Microstep Select (MSx). The microstep resolution is set by the voltage on logic inputs MSx, as shown in Table 1. The MS1 and MS3 pins have a 100 kΩ pull-down resistance, and the MS2 pin has a 50 kΩ pull-down resistance. When changing the step mode the change does not take effect until the next STEP rising edge.

If the step mode is changed without a translator reset, and absolute position must be maintained, it is important to change the step mode at a step position that is common to both step modes in order to avoid missing steps. When the device is powered down, or reset due to TSD or an over current event the translator is set to the home position which is by default common to all step modes.

Mixed Decay Operation. The bridge operates in Mixed decay mode, at power-on and reset, and during normal running according to the ROSC configuration and the step sequence, as shown in Figures 9 through 13. During Mixed decay, when the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF}. After that, it switches to Slow decay mode for the remainder of t_{OFF}. A timing diagram for this feature appears on the next page.

Typically, mixed decay is only necessary when the current in the winding is going from a higher value to a lower value as determined by the state of the translator. For most loads automatically-selected mixed decay is convenient because it minimizes ripple when the current is rising and prevents missed steps when the current is falling. For some applications where microstepping at very low speeds is necessary, the lack of back EMF in the winding causes the current to increase in the load quickly, resulting in missed steps. This is shown in Figure 2. By pulling the ROSC pin to ground, mixed decay is set to be active 100% of the time, for both rising and falling currents, and prevents missed steps as shown in Figure 3. If this is not an issue, it is recommended that automatically-selected mixed decay be used, because it will produce reduced ripple currents. Refer to the Fixed Off-Time section for details.

Low Current Microstepping. Intended for applications where the minimum on-time prevents the output current from regulating to the programmed current level at low current steps. To prevent this, the device can be set to operate in Mixed decay mode on both rising and falling portions of the current waveform. This feature is implemented by shorting the ROSC pin to ground. In this state, the off-time is internally set to 30 µs.

Reset Input ($\overline{\text{RESET}}$). The $\overline{\text{RESET}}$ input sets the translator to a predefined Home state (shown in Figures 9 through 13), and turns off all of the FET outputs. All STEP inputs are ignored until the $\overline{\text{RESET}}$ input is set to high.

Step Input (STEP). A low-to-high transition on the STEP

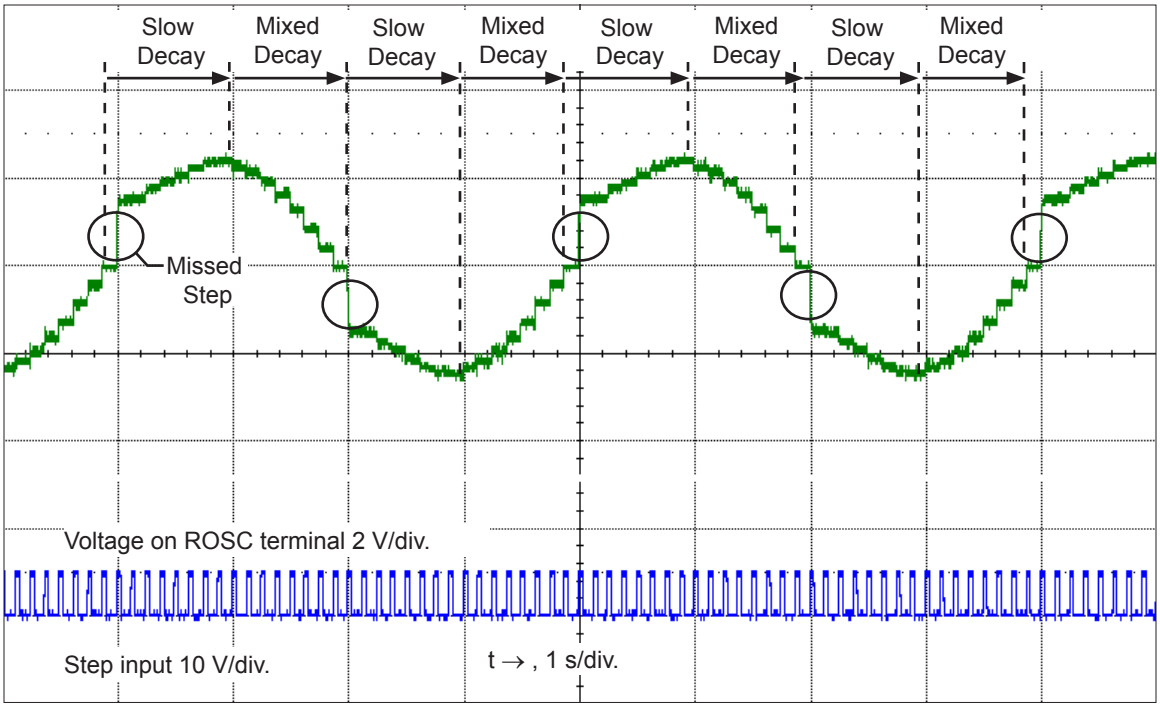


Figure 2: Missed Steps in Low-Speed Microstepping

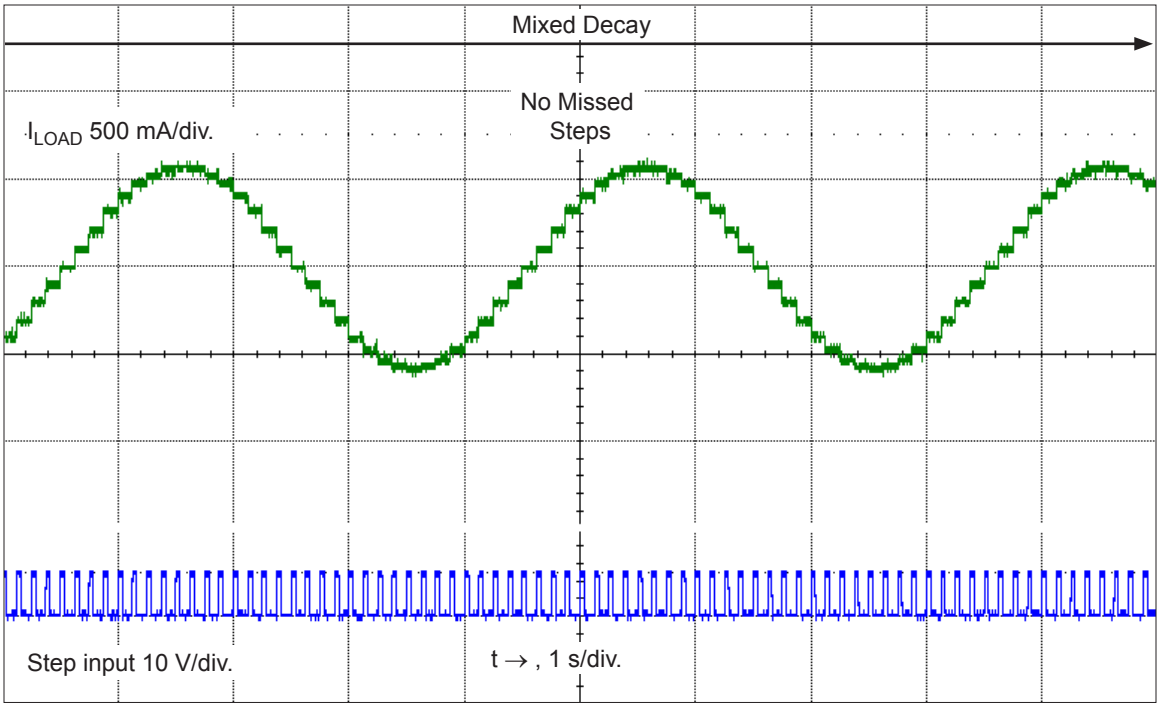


Figure 3: Continuous Stepping Using Automatically-Selected Mixed Stepping (ROSC pin grounded)

input sequences the translator and advances the motor one increment. The translator controls the input to the DACs and the direction of current flow in each winding. The size of the increment is determined by the combined state of the MSx inputs.

Direction Input (DIR). This determines the direction of rotation of the motor. Changes to this input do not take effect until the next STEP rising edge.

Internal PWM Current Control. Each full-bridge is controlled by a fixed off-time PWM current control circuit that limits the load current to a desired value, I_{TRIP}. Initially, a diagonal pair of source and sink FET outputs are enabled and current flows through the motor winding and the current sense resistor, R_{Sx}. When the voltage across R_{Sx} equals the DAC output voltage, the current sense comparator resets the PWM latch. The latch then turns off the appropriate source driver and initiates a fixed off time decay mode

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).

The DAC output reduces the V_{REF} output to the current sense comparator in precise steps, such that

$$I_{trip} = (\%I_{TripMAX} / 100) \times I_{TripMAX}$$

(See Table 2 for $\%I_{TripMAX}$ at each step.)

It is critical that the maximum rating (0.5 V) on the SENSE1 and SENSE2 pins is not exceeded.

Fixed Off-Time. The internal PWM current control circuitry uses a one-shot circuit to control the duration of time that the DMOS FETs remain off. The off-time, t_{OFF}, is determined by the ROSC terminal. The ROSC terminal has three settings:

- ROSC tied to VDD — off-time internally set to 30 µs, decay mode is automatic Mixed decay except when in full step where decay mode is set to Slow decay
- ROSC tied directly to ground — off-time internally set to 30 µs, current decay is set to Mixed decay for both increasing and decreasing currents for all step modes.
- ROSC through a resistor to ground — off-time is determined by the following formula, the decay mode is automatic Mixed decay for all step modes except full step whic is set to slow decay.

$$t_{OFF} \approx R_{OSC} / 825$$

Where t_{OFF} is in µs.

Blanking. This function blanks the output of the current sense comparators when the outputs are switched by the internal current control circuitry. The comparator outputs are blanked to prevent false overcurrent detection due to reverse recovery currents of the clamp diodes, and switching transients related to the capacitance of the load. The blank time, t_{BLANK} (µs), is approximately

$$t_{BLANK} \approx 1\ \mu s$$

Shorted-Load and Short-to-Ground Protection. If the motor leads are shorted together, or if one of the leads is shorted to ground, the driver will protect itself by sensing the overcurrent event and disabling the driver that is shorted, protecting the device from damage. In the case of a short-to-ground, the device will remain disabled (latched) until the $\overline{\text{SLEEP}}$ input goes high or VDD power is removed. A short-to-ground overcurrent event is shown in Figure 4.

When the two outputs are shorted together, the current path is through the sense resistor. After the blanking time (≈1 µs) expires, the sense resistor voltage is exceeding its trip value, due to the overcurrent condition that exists. This causes the driver to go into a fixed off-time cycle. After the fixed off-time expires the driver turns on again and the process repeats. In this condition the driver is completely protected against overcurrent events, but the short is repetitive with a period equal to the fixed off-time of the driver. This condition is shown in Figure 5.

During a shorted load event it is normal to observe both a positive and negative current spike as shown in Figure 3, due to the direction change implemented by the Mixed decay feature. This is shown in Figure 6. In both instances the overcurrent circuitry is protecting the driver and prevents damage to the device.

Charge Pump (CP1 and CP2). The charge pump is used to generate a gate supply greater than that of VBB for driving the source-side FET gates. A 0.1 µF ceramic capacitor, should be connected between CP1 and CP2. In addition, a 0.1 µF ceramic capacitor is required between VCP and VBB, to act as a reservoir for operating the high-side FET gates.

Capacitor values should be Class 2 dielectric ±15% maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

V_{REG} (VREG). This internally-generated voltage is used to operate the sink-side FET outputs. The nominal output voltage of the VREG terminal is 7 V. The VREG pin must be decoupled with a 0.22 µF ceramic capacitor to ground. V_{REG} is internally monitored. In the case of a fault condition, the FET outputs of the A4988 are disabled.

Capacitor values should be Class 2 dielectric ±15% maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

Enable Input ($\overline{\text{ENABLE}}$). This input turns on or off all of the FET outputs. When set to a logic high, the outputs are disabled. When set to a logic low, the internal control enables the outputs as required. The translator inputs STEP, DIR, and MSx, as well as the internal sequencing logic, all remain active, independent of the $\overline{\text{ENABLE}}$ input state.

Shutdown. In the event of a fault, overtemperature (excess T_J) or an undervoltage (on VCP), the FET outputs of the A4988 are disabled until the fault condition is removed. At power-on, the UVLO (undervoltage lockout) circuit disables the FET outputs and resets the translator to the Home state.

Sleep Mode ($\overline{\text{SLEEP}}$). To minimize power consumption when the motor is not in use, this input disables much of the internal circuitry including the output FETs, current regulator, and charge pump. A logic low on the $\overline{\text{SLEEP}}$ pin puts the A4988 into Sleep mode. A logic high allows normal operation, as well as start-up (at which time the A4988 drives the motor to the Home microstep position). When emerging from Sleep mode, in order to allow the charge pump to stabilize, provide a delay of 1 ms before issuing a Step command.

Mixed Decay Operation. The bridge operates in Mixed Decay mode, depending on the step sequence, as shown in Figures 9 through 13. As the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF}. After that, it switches to Slow Decay mode for the remainder of t_{OFF}. A timing diagram for this feature appears in Figure 7.

Synchronous Rectification. When a PWM-off cycle is triggered by an internal fixed-off time cycle, load current recirculates according to the decay mode selected by the control logic. This synchronous rectification feature turns on the appropriate FETs during current decay, and effectively shorts out the body diodes with the low FET $R_{DS(ON)}$. This reduces power dissipation significantly, and can eliminate the need for external Schottky diodes in many applications. Synchronous rectification turns off when the load current approaches zero (0 A), preventing reversal of the load current.

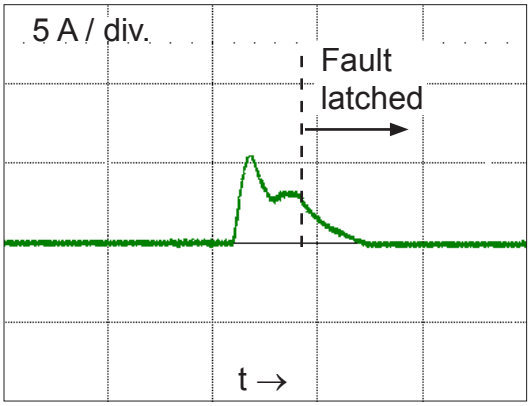


Figure 4: Short-to-Ground Event

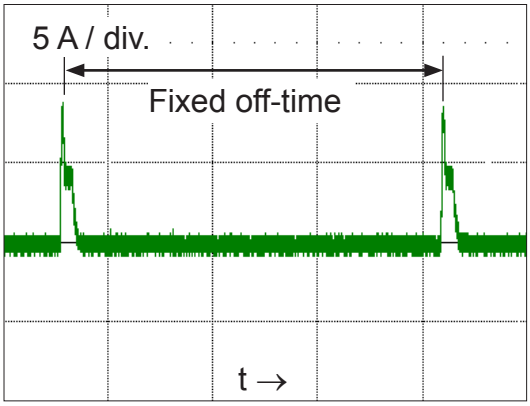


Figure 5. Shorted Load (OUTxA → OUTxB) in Slow Decay Mode

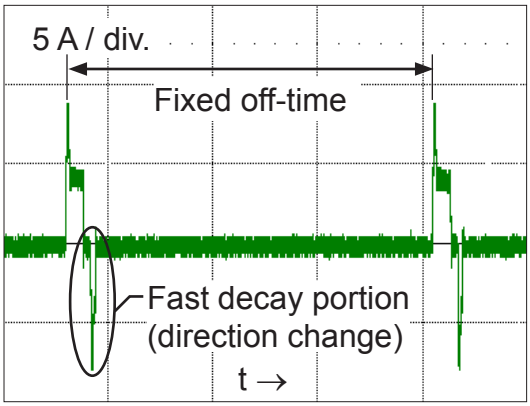
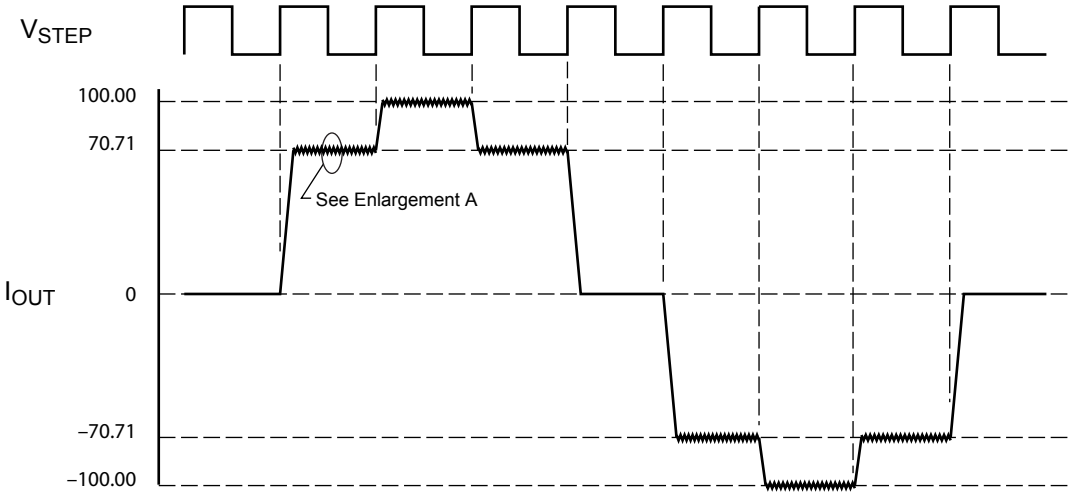
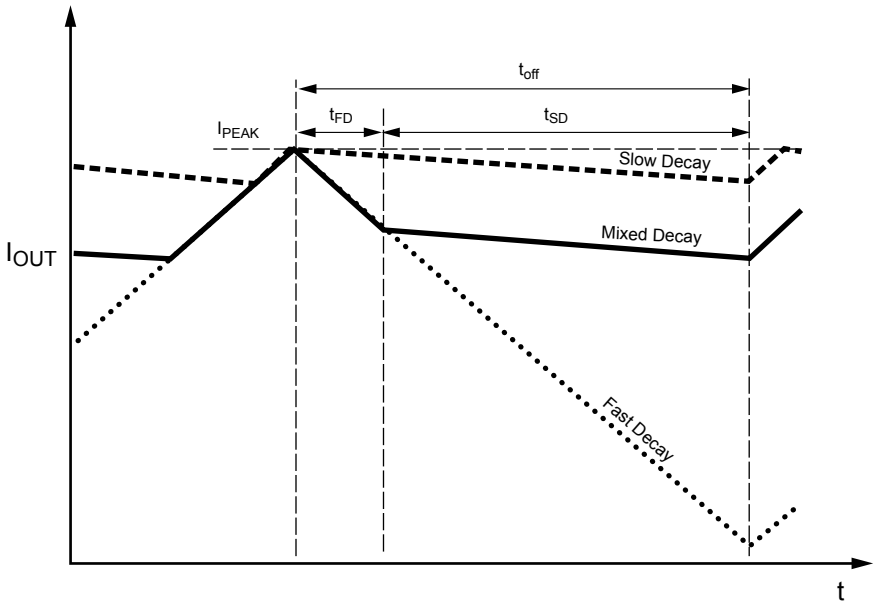


Figure 6: Shorted Load (OUTxA → OUTxB) in Mixed Decay Mode



Enlargement A



Symbol	Characteristic
t_{off}	Device fixed off-time
I_{PEAK}	Maximum output current
t_{SD}	Slow decay interval
t_{FD}	Fast decay interval
I_{OUT}	Device output current

Figure 7: Current Decay Modes Timing Chart

Application Layout

Layout. The printed circuit board should use a heavy ground-plane. For optimum electrical and thermal performance, the A4988 must be soldered directly onto the board. Pins 3 and 18 are internally fused, which provides a path for enhanced thermal dissipation. Theses pins should be soldered directly to an exposed surface on the PCB that connects to thermal vias are used to transfer heat to other layers of the PCB.

In order to minimize the effects of ground bounce and offset issues, it is important to have a low impedance single-point ground, known as a *star ground*, located very close to the device. By making the connection between the pad and the ground plane directly under the A4988, that area becomes an ideal location for a star ground point. A low impedance ground will prevent ground bounce during high current operation and ensure that the supply voltage remains stable at the input terminal.

The two input capacitors should be placed in parallel, and as close to the device supply pins as possible. The ceramic capacitor (CIN1) should be closer to the pins than the bulk capacitor (CIN2). This is necessary because the ceramic capacitor will be responsible for delivering the high frequency current components. The sense resistors, RSx , should have a very low impedance path to ground, because they must carry a large current while supporting very accurate voltage measurements by the current sense comparators. Long ground traces will cause additional voltage drops, adversely affecting the ability of the comparators to accurately measure the current in the windings. The SENSEx pins have very short traces to the RSx resistors and very thick, low impedance traces directly to the star ground underneath the device. If possible, there should be no other components on the sense circuits.

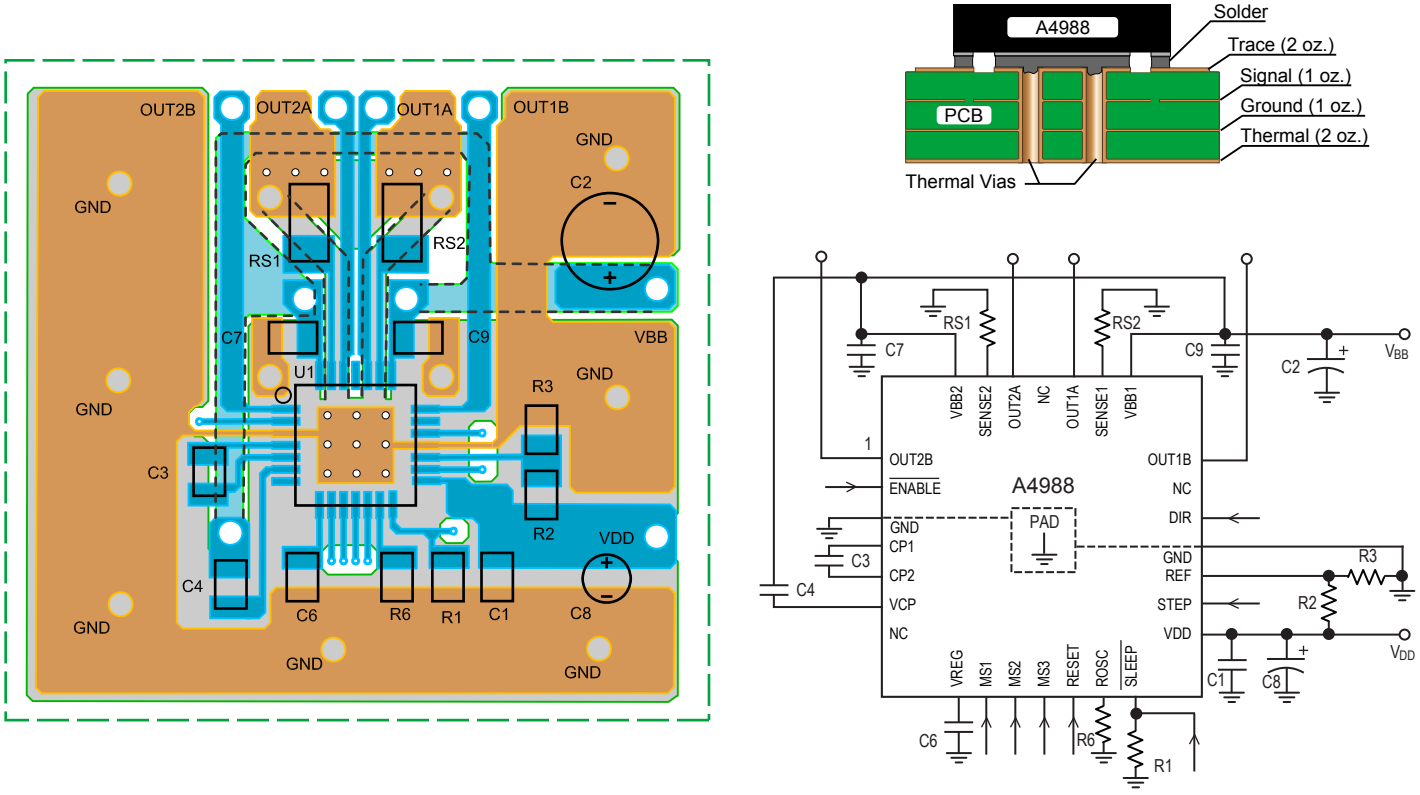
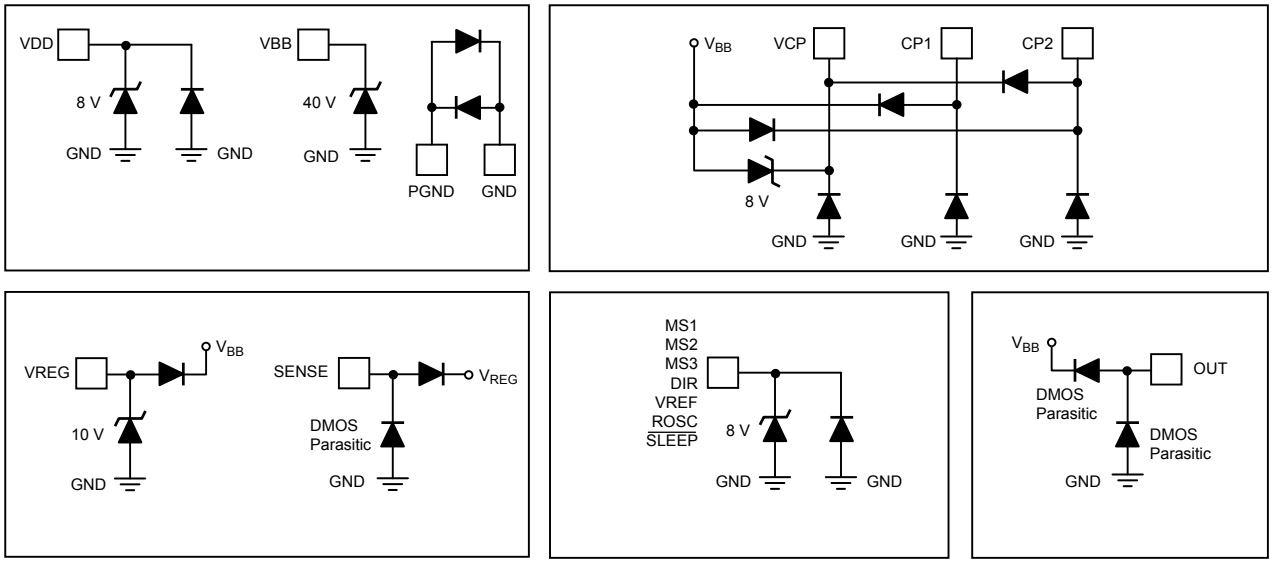
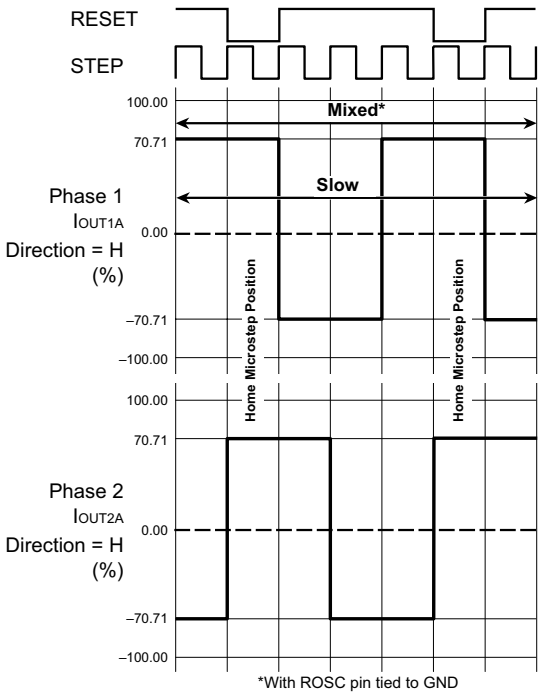


Figure 8: Typical Application and Circuit Layout

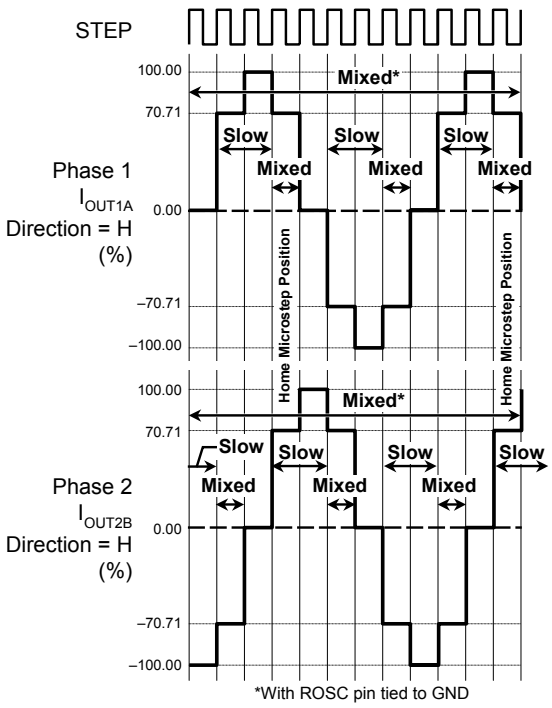
Pin Circuit Diagrams





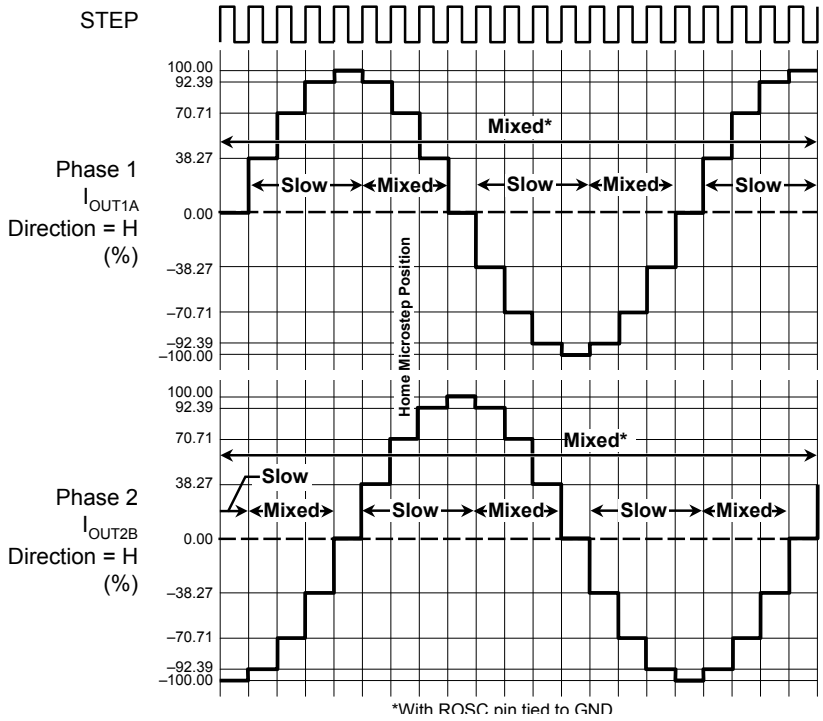
DIR= H

Figure 9: Decay Mode for Full-Step Increments



DIR= H

Figure 10: Decay Modes for Half-Step Increments



DIR= H

Figure 11: Decay Modes for Quarter-Step Increments

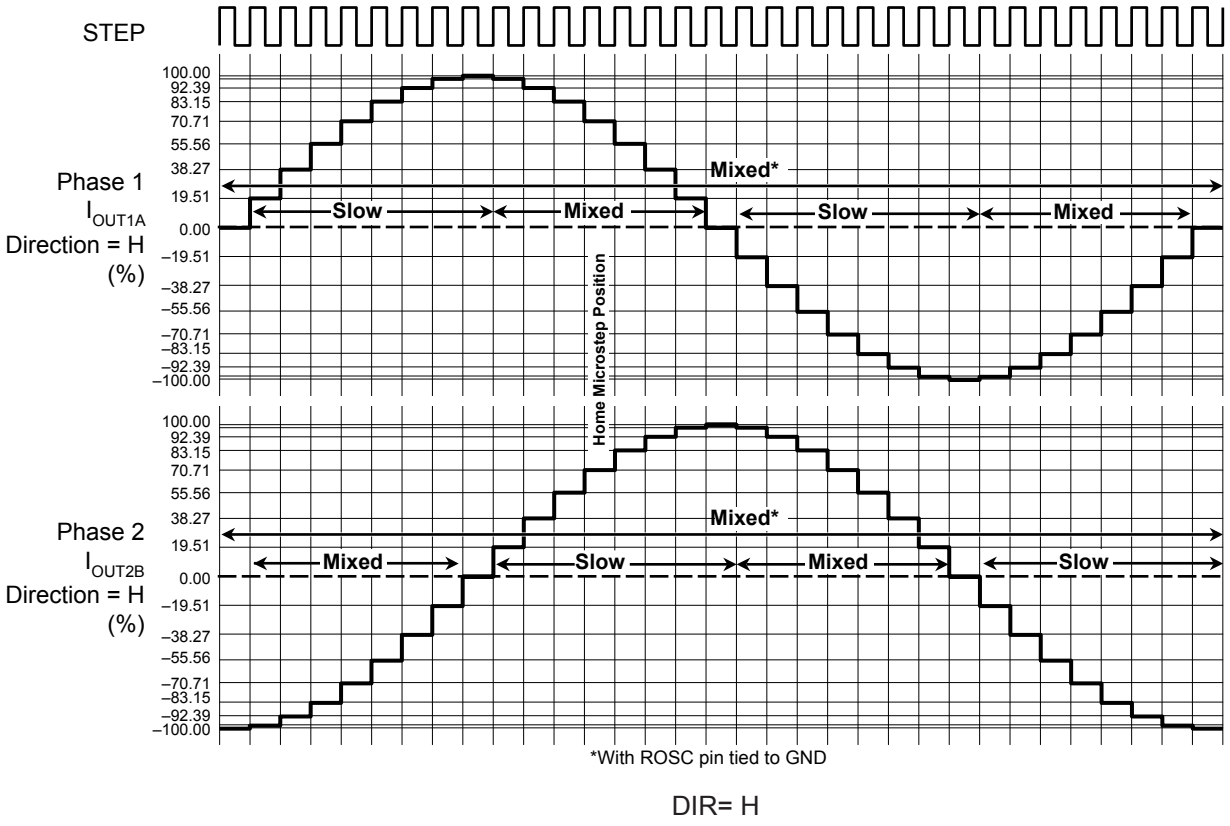


Figure 12: Decay Modes for Eighth-Step Increments

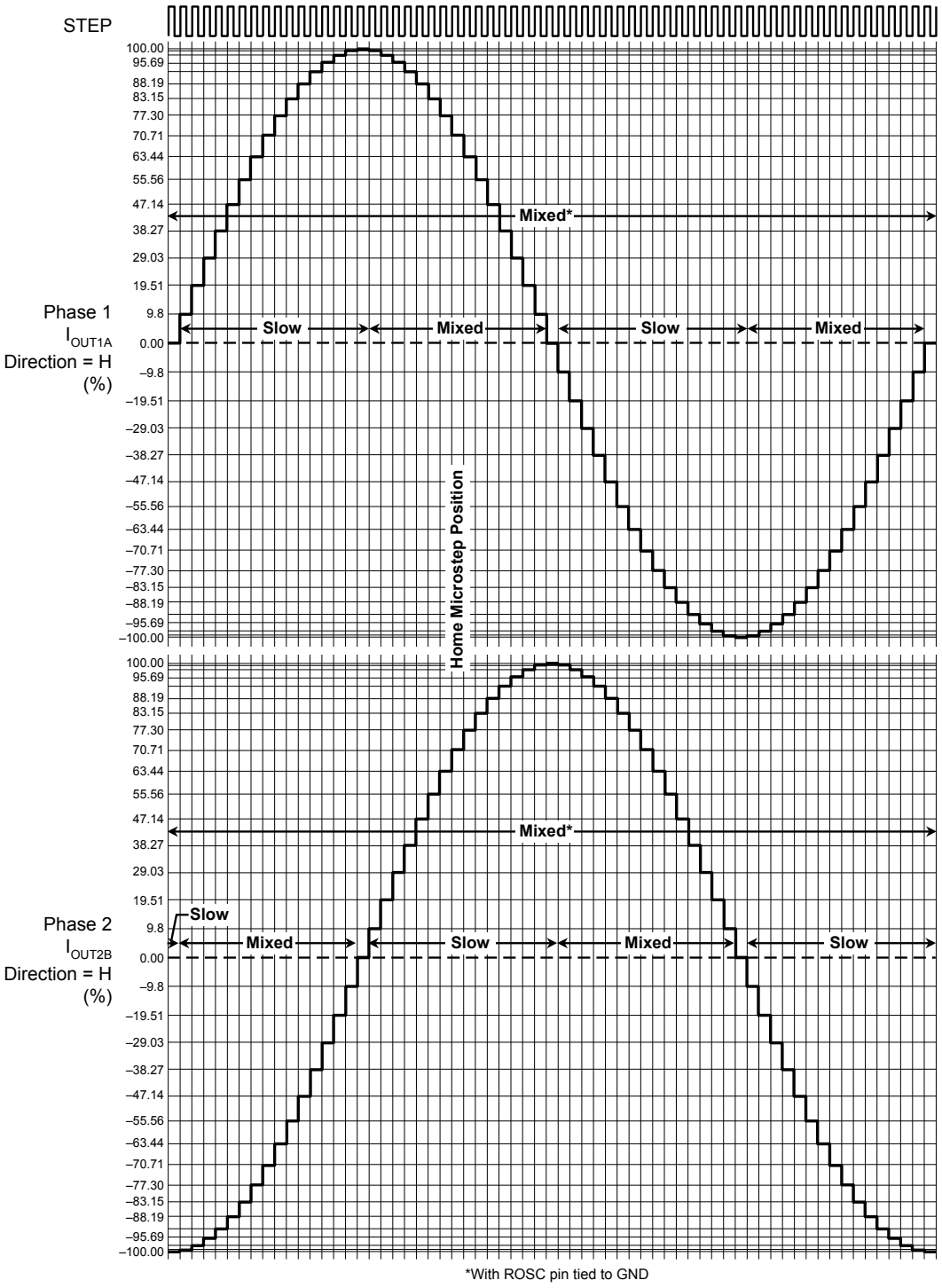


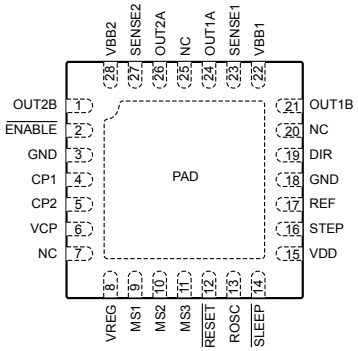
Figure 13: Decay Modes for Sixteenth-Step Increments

Table 2: Step Sequencing Settings

Home microstep position at Step Angle 45º; DIR = H

Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% $I_{tripMax}$] (%)	Phase 2 Current [% $I_{tripMax}$] (%)	Step Angle (º)	Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% $I_{tripMax}$] (%)	Phase 2 Current [% $I_{tripMax}$] (%)	Step Angle (º)
	1	1	1	1	100.00	0.00	0.0		5	9	17	33	–100.00	0.00	180.0
				2	99.52	9.80	5.6					34	–99.52	–9.80	185.6
			2	3	98.08	19.51	11.3				18	35	–98.08	–19.51	191.3
				4	95.69	29.03	16.9					36	–95.69	–29.03	196.9
		2	3	5	92.39	38.27	22.5			10	19	37	–92.39	–38.27	202.5
				6	88.19	47.14	28.1					38	–88.19	–47.14	208.1
			4	7	83.15	55.56	33.8				20	39	–83.15	–55.56	213.8
				8	77.30	63.44	39.4					40	–77.30	–63.44	219.4
1	2	3	5	9	70.71	70.71	45.0	3	6	11	21	41	–70.71	–70.71	225.0
				10	63.44	77.30	50.6					42	–63.44	–77.30	230.6
			6	11	55.56	83.15	56.3				22	43	–55.56	–83.15	236.3
				12	47.14	88.19	61.9					44	–47.14	–88.19	241.9
		4	7	13	38.27	92.39	67.5			12	23	45	–38.27	–92.39	247.5
				14	29.03	95.69	73.1					46	–29.03	–95.69	253.1
			8	15	19.51	98.08	78.8				24	47	–19.51	–98.08	258.8
				16	9.80	99.52	84.4					48	–9.80	–99.52	264.4
	3	5	9	17	0.00	100.00	90.0		7	13	25	49	0.00	–100.00	270.0
				18	–9.80	99.52	95.6					50	9.80	–99.52	275.6
			10	19	–19.51	98.08	101.3				26	51	19.51	–98.08	281.3
				20	–29.03	95.69	106.9					52	29.03	–95.69	286.9
		6	11	21	–38.27	92.39	112.5			14	27	53	38.27	–92.39	292.5
				22	–47.14	88.19	118.1					54	47.14	–88.19	298.1
			12	23	–55.56	83.15	123.8				28	55	55.56	–83.15	303.8
				24	–63.44	77.30	129.4					56	63.44	–77.30	309.4
2	4	7	13	25	–70.71	70.71	135.0	4	8	15	29	57	70.71	–70.71	315.0
				26	–77.30	63.44	140.6					58	77.30	–63.44	320.6
			14	27	–83.15	55.56	146.3				30	59	83.15	–55.56	326.3
				28	–88.19	47.14	151.9					60	88.19	–47.14	331.9
		8	15	29	–92.39	38.27	157.5			16	31	61	92.39	–38.27	337.5
				30	–95.69	29.03	163.1					62	95.69	–29.03	343.1
			16	31	–98.08	19.51	168.8				32	63	98.08	–19.51	348.8
				32	–99.52	9.80	174.4					64	99.52	–9.80	354.4

Pin-out Diagram

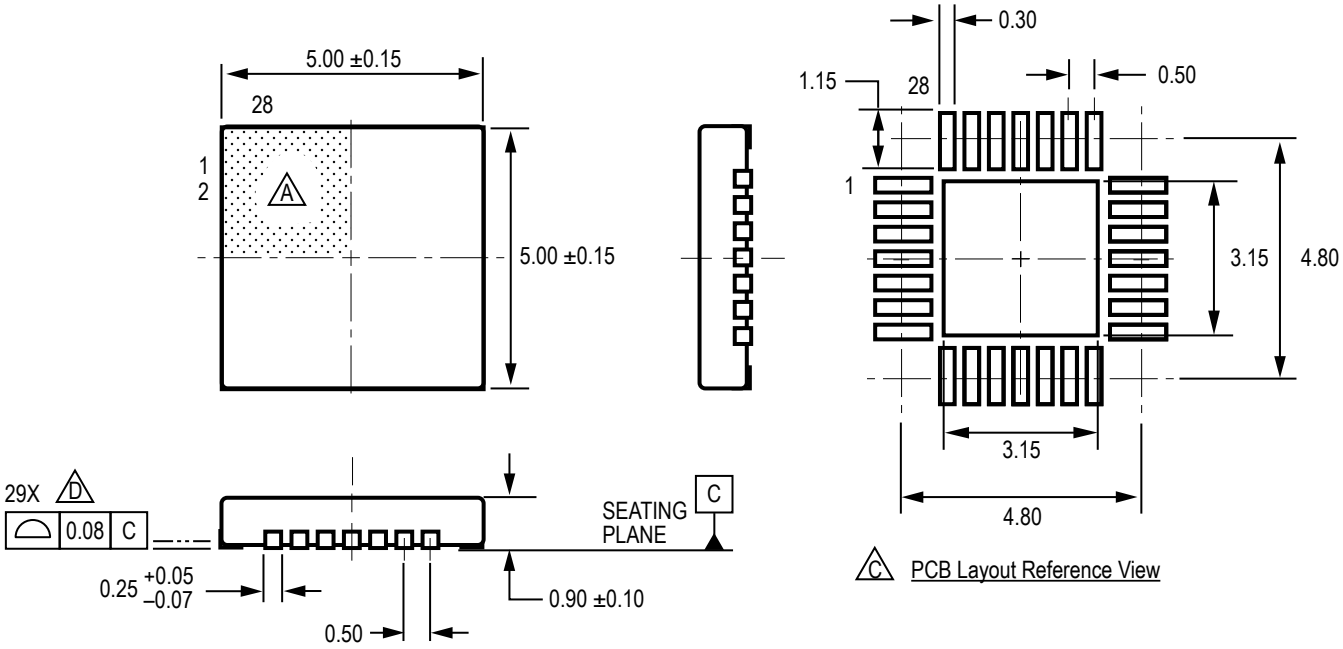


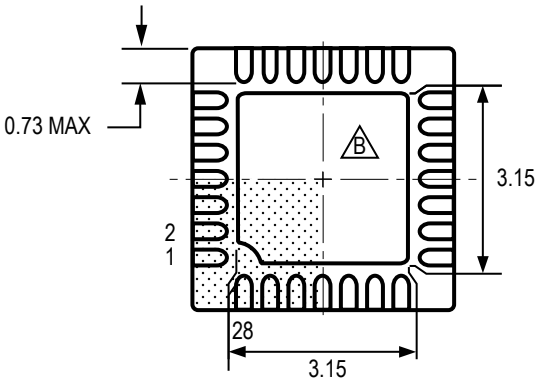
Terminal List Table

Name	Number	Description
CP1	4	Charge pump capacitor terminal
CP2	5	Charge pump capacitor terminal
VCP	6	Reservoir capacitor terminal
VREG	8	Regulator decoupling terminal
MS1	9	Logic input
MS2	10	Logic input
MS3	11	Logic input
$\overline{\text{RESET}}$	12	Logic input
ROSC	13	Timing set
$\overline{\text{SLEEP}}$	14	Logic input
VDD	15	Logic supply
STEP	16	Logic input
REF	17	G_m reference voltage input
GND	3, 18	Ground*
DIR	19	Logic input
OUT1B	21	DMOS Full Bridge 1 Output B
VBB1	22	Load supply
SENSE1	23	Sense resistor terminal for Bridge 1
OUT1A	24	DMOS Full Bridge 1 Output A
OUT2A	26	DMOS Full Bridge 2 Output A
SENSE2	27	Sense resistor terminal for Bridge 2
VBB2	28	Load supply
OUT2B	1	DMOS Full Bridge 2 Output B
$\overline{\text{ENABLE}}$	2	Logic input
NC	7, 20, 25	No connection
PAD	–	Exposed pad for enhanced thermal dissipation*

*The GND pins must be tied together externally by connecting to the PAD ground plane under the device.

ET Package, 28-Pin QFN with Exposed Thermal Pad





For Reference Only; not for tooling use
(reference JEDEC MO-220VHHD-1)
Dimensions in millimeters
Exact case and lead configuration at supplier discretion within limits shown

A Terminal #1 mark area

B Exposed thermal pad (reference only, terminal #1 identifier appearance at supplier discretion)

C Reference land pattern layout (reference IPC7351 QFN50P500X500X100-29V1M);
All pads a minimum of 0.20 mm from all adjacent pads; adjust as necessary to meet application process requirements and PCB layout tolerances; when mounting on a multilayer PCB, thermal vias at the exposed thermal pad land can improve thermal dissipation (reference EIA/JEDEC Standard JESD51-5)

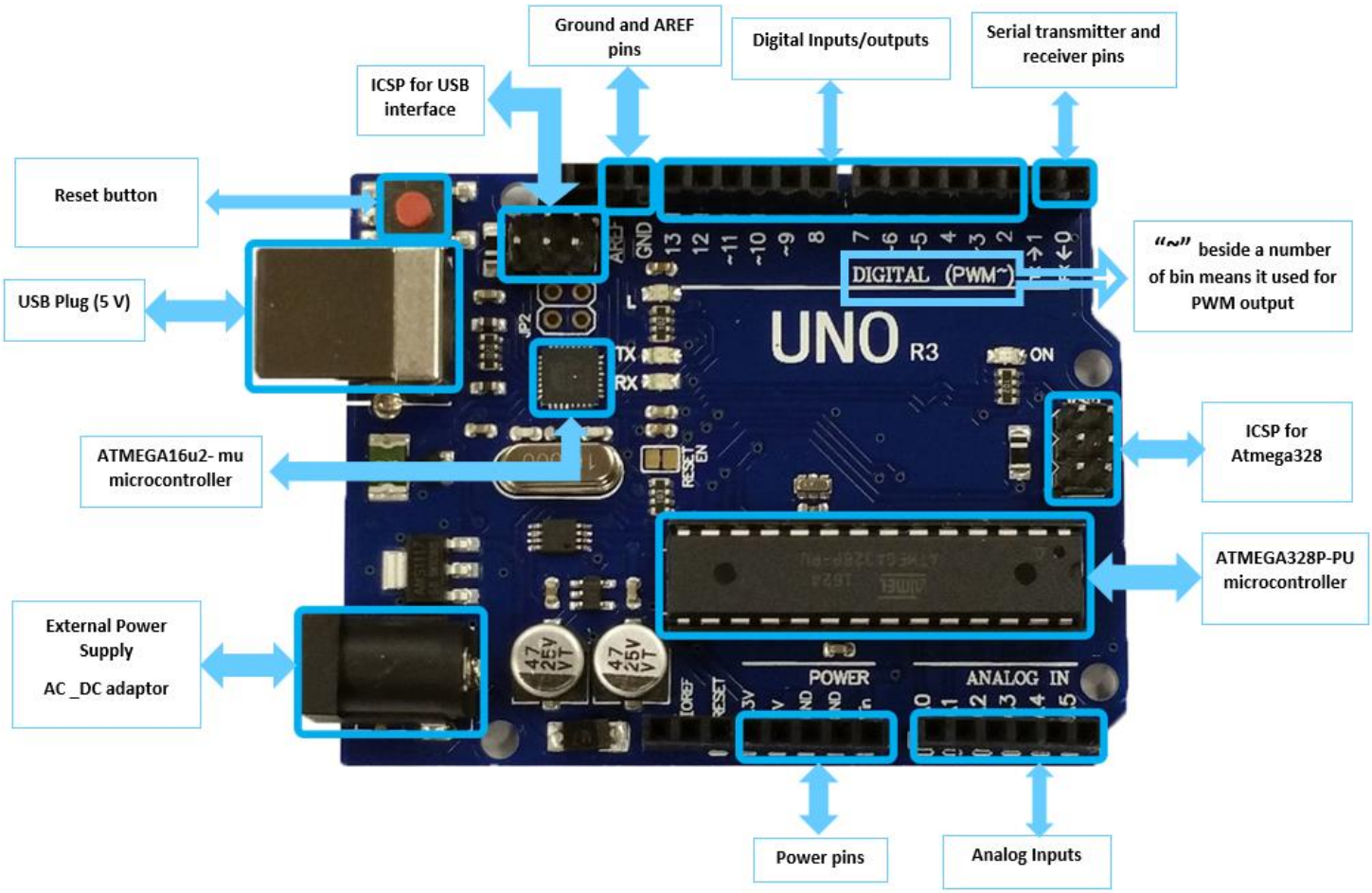
D Coplanarity includes exposed thermal pad and terminals

Revision History

Revision	Revision Date	Description of Revision
4	January 27, 2012	Update I_{OCPST}
5	May 7, 2014	Revised text on pg. 9; revised Figure 8 and Table 2

For the latest version of this document, visit our website:
www.allegromicro.com

Arduino Uno R3



INTRODUCTION

Arduino is used for building different types of electronic circuits easily using of both a physical programmable circuit board usually microcontroller and piece of code running on computer with USB connection between the computer and Arduino.

Programming language used in Arduino is just a simplified version of C++ that can easily replace thousands of wires with words.

ARDUINO UNO-R3 PHYSICAL COMPONENTS

ATMEGA328P-PU microcontroller

The most important element in Arduino Uno R3 is ATMEGA328P-PU is an 8-bit Microcontroller with flash memory reach to 32k bytes. It's features as follow:

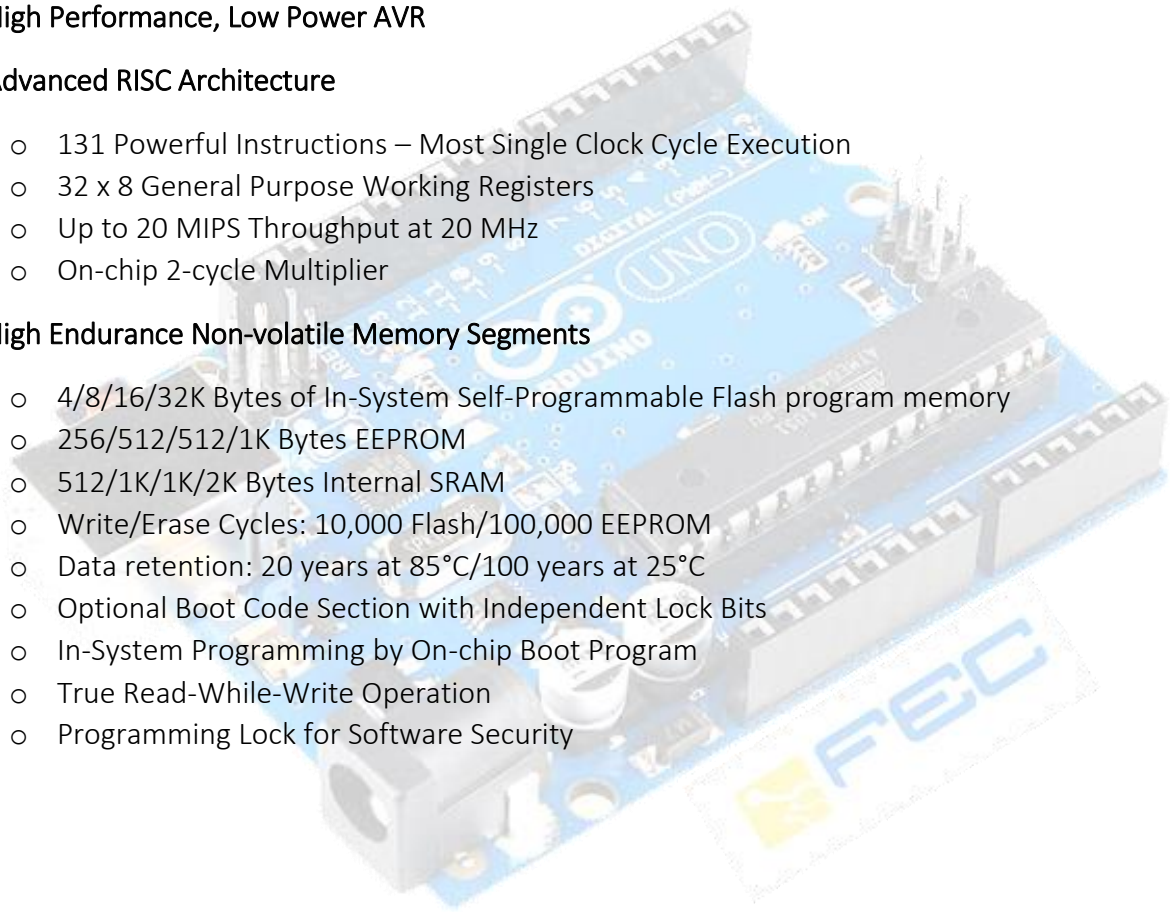
• High Performance, Low Power AVR

• Advanced RISC Architecture

- 131 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Up to 20 MIPS Throughput at 20 MHz
- On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory Segments

- 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
- 256/512/512/1K Bytes EEPROM
- 512/1K/1K/2K Bytes Internal SRAM
- Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
- Data retention: 20 years at 85°C/100 years at 25°C
- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
- Programming Lock for Software Security



• Peripheral Features

- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Six PWM Channels
- 8-channel 10-bit ADC in TQFP and QFN/MLF package
- Temperature Measurement
- 6-channel 10-bit ADC in PDIP Package
- Temperature Measurement
- Programmable Serial USART

- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface (Philips I2 C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

• Special Microcontroller Features

- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages

- 23 Programmable I/O Lines
- 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

• Operating Voltage:

- 1.8 - 5.5V

• Temperature Range:

- -40°C to 85°C

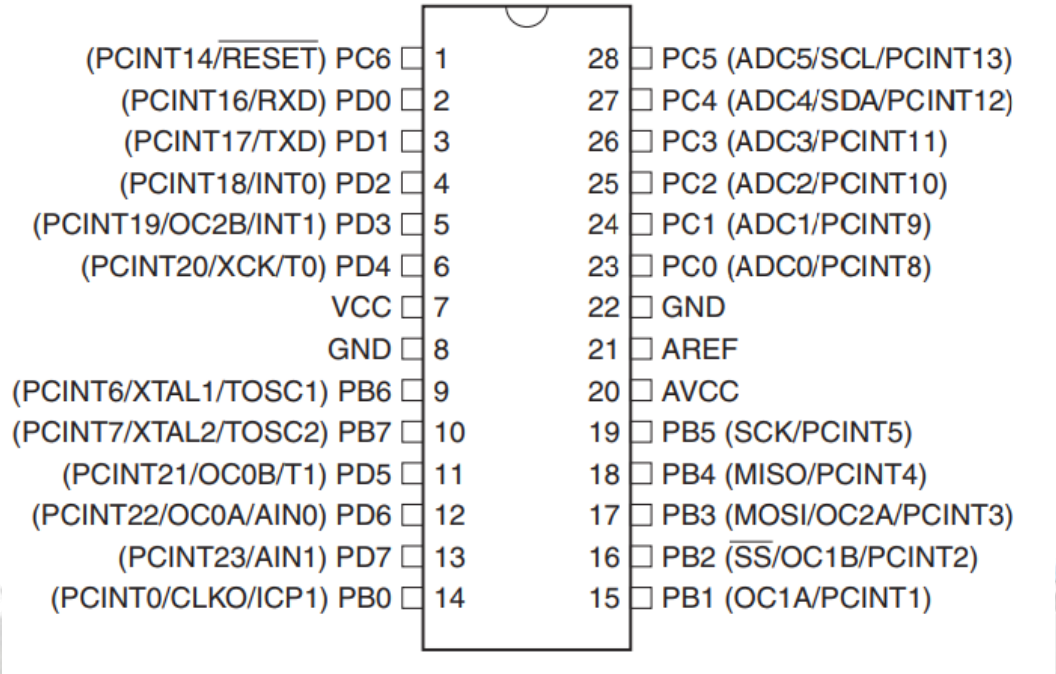
• Speed Grade:

- 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V

• Power Consumption at 1 MHz, 1.8V, 25°C

- Active Mode: 0.2 mA
- Power-down Mode: 0.1 µA
- Power-save Mode: 0.75 µA (Including 32 kHz RTC)

- Pin configuration



ATMEGA16u2- mu microcontroller

Is a 8-bit microcontroller used as USB driver in Arduino uno R3 it's features as follow:

- High Performance, Low Power AVR

• Advanced RISC Architecture

- 125 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Fully Static Operation
- Up to 16 MIPS Throughput at 16 MHz

• Non-volatile Program and Data Memories

- 8K/16K/32K Bytes of In-System Self-Programmable Flash
- 512/512/1024 EEPROM
- 512/512/1024 Internal SRAM
- Write/Erase Cycles: 10,000 Flash/ 100,000 EEPROM
- Data retention: 20 years at 85˚C/ 100 years at 25˚C

- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by on-chip Boot Program hardware-activated after reset
- Programming Lock for Software Security

• USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion

- Complies fully with Universal Serial Bus Specification REV 2.0
- 48 MHz PLL for Full-speed Bus Operation: data transfer rates at 12 Mbit/s
- Fully independent 176 bytes USB DPRAM for endpoint memory allocation
- Endpoint 0 for Control Transfers: from 8 up to 64-bytes
- 4 Programmable Endpoints:
  - IN or Out Directions
  - Bulk, Interrupt and Isochronous Transfers
  - Programmable maximum packet size from 8 to 64 bytes
  - Programmable single or double buffer
- Suspend/Resume Interrupts
- Microcontroller reset on USB Bus Reset without detach
- USB Bus Disconnection on Microcontroller Request

• Peripheral Features

- One 8-bit Timer/Counters with Separate Prescaler and Compare Mode (two 8-bit PWM channels)
- One 16-bit Timer/Counter with Separate Prescaler, Compare and Capture Mode(three 8-bit PWM channels)
- USART with SPI master only mode and hardware flow control (RTS/CTS)
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

• On Chip Debug Interface (debug WIRE)

• Special Microcontroller Features

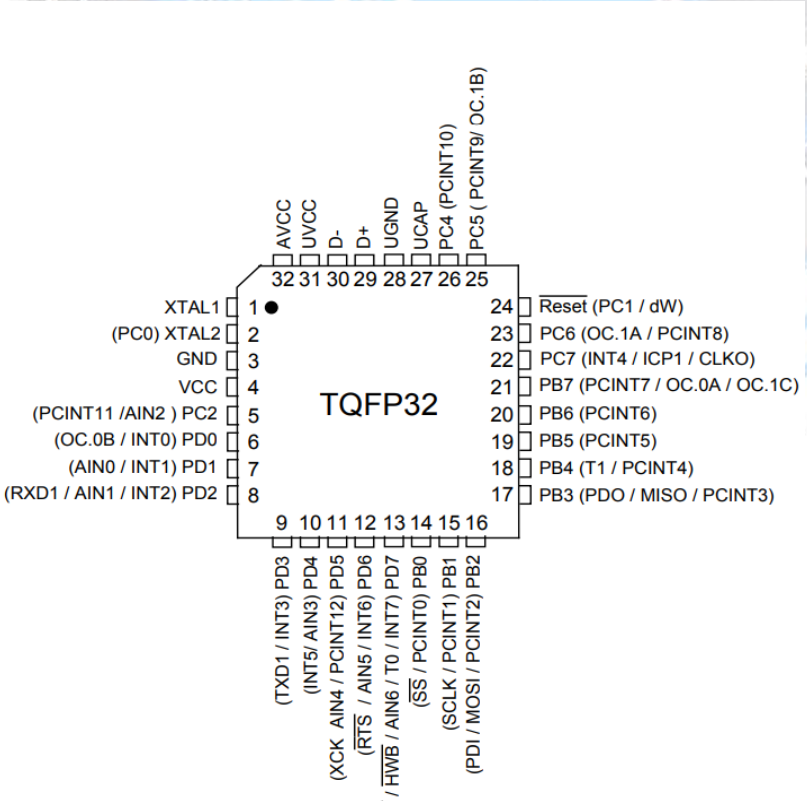
- Power-On Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Five Sleep Modes: Idle, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages

- 22 Programmable I/O Lines
- QFN32 (5x5mm) / TQFP32 packages

- Operating Voltages
  - 2.7 - 5.5V
- Operating temperature
  - Industrial (-40°C to +85°C)
- Maximum Frequency
  - 8 MHz at 2.7V - Industrial range
  - 16 MHz at 4.5V - Industrial range

- Pin configuration



OTHER ARDUINO UNO R3 PARTS

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode (), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 k Ohms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

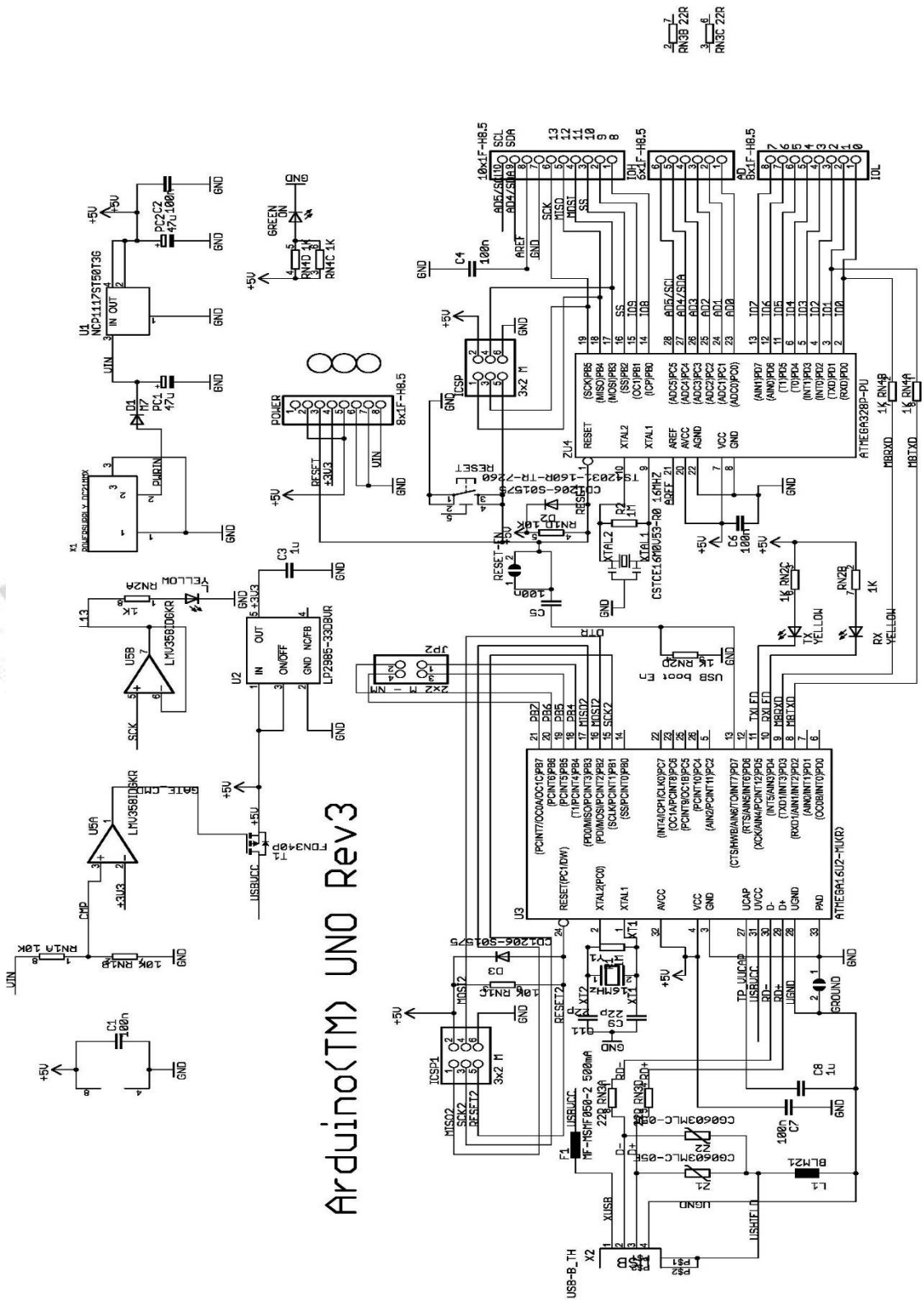
The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e.1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

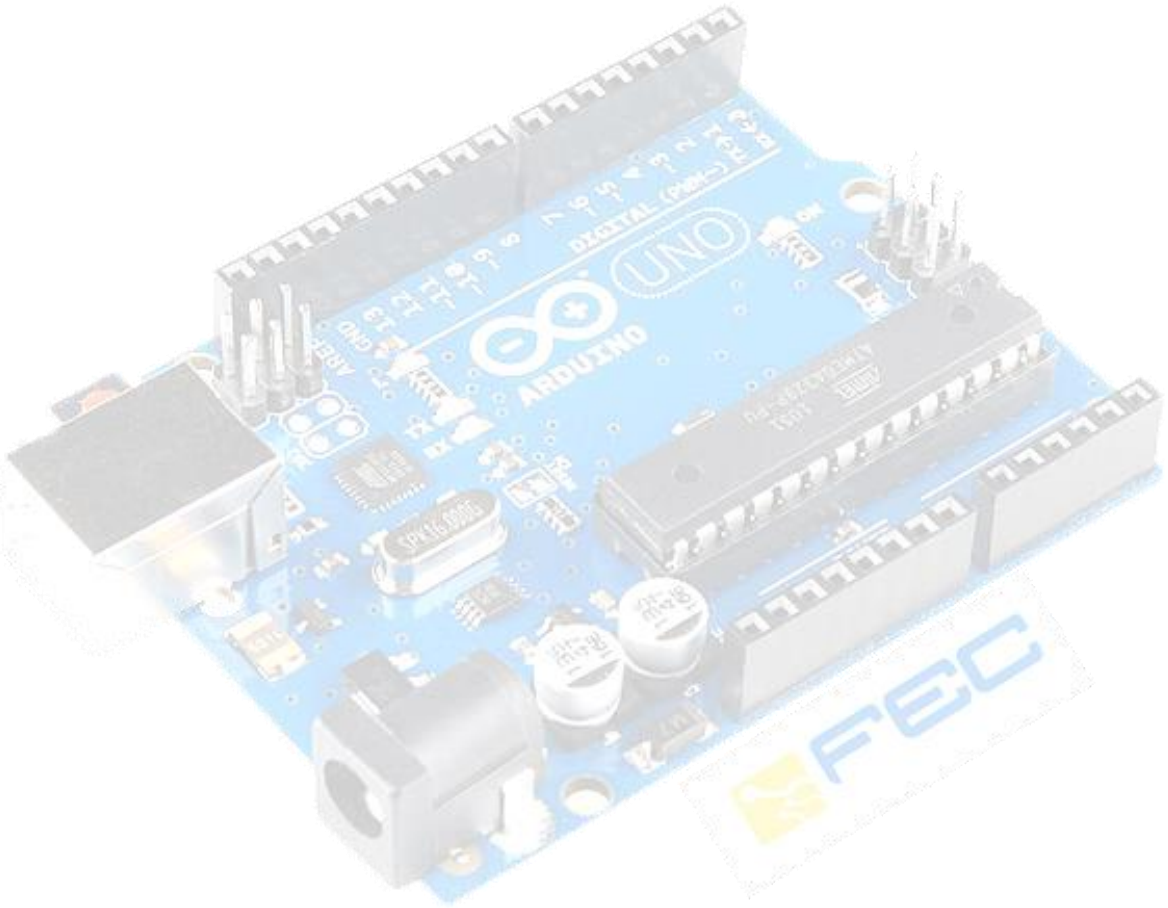
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF: Reference voltage for the analog inputs. Used with analogReference().
- Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

ARDUINO UNO R3 SCHEMATIC DIAGRAM




UNO
ARDUINO