

INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

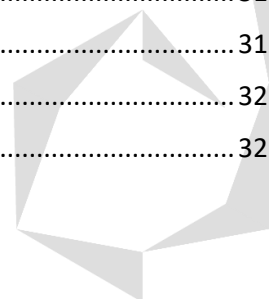
ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE ARDUINO, VHDL Y VERILOG

IDE ARDUINO, XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Motor a Pasos Unipolar y Bipolar  
con Arduino, Verilog y VHDL

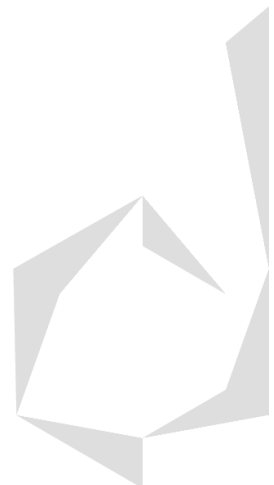
# Contenido

<b>Partes y Tipos de Motores Eléctricos AC/DC</b> .....	4
Tipos de Motores Eléctricos AC/DC .....	4
Bobinas y Electroimanes .....	5
Partes de un Motor DC con Escobillas: Conmutador, Estator, Armadura y Rotor .....	7
Conexión en Serie o Paralelo del Estator y Rotor de un Motor DC .....	8
<b>Manejo de la Corriente en las Bobinas de los Motores</b> .....	9
Diodo de Marcha Libre.....	9
Etapa de Control y Etapa de Potencia en un Circuito con Motores .....	10
<b>Etapa de Potencia:</b> Corriente Demandada por el Motor al Agregar una Carga.....	10
<b>Etapa de Control:</b> Aislamiento de los Circuitos Lógicos .....	11
<b>Motor a Pasos</b> .....	14
<b>Motor a Pasos Unipolar 28BYJ-48 y su Controlador ULN2003</b> .....	17
Pasos del Motor a Pasos Unipolar: Simple, Doble y Medio en <b>Arduino</b> .....	18
<b>Código Arduino</b> - Paso Simple: .....	20
<b>Código Arduino</b> - Paso Simple - Array: .....	21
<b>Código Arduino</b> - Paso Doble:.....	22
<b>Código Arduino</b> - Paso Doble - Array:.....	23
<b>Código Arduino</b> - Paso Medio:.....	24
<b>Código Arduino</b> - Paso Medio - Array:.....	26
TLD Motor a Pasos Unipolar: Paso Simple, Doble y Medio en <b>Verilog y VHDL</b> .....	27
<b>Código Verilog</b> – Paso Simple: .....	28
Diagrama TLD:.....	28
Divisor de Reloj: .....	28
Control Motor a Pasos Unipolar: .....	29
Módulo TLD:.....	29
Código UCF:.....	30
Simulación Paso Simple: .....	30
<b>Código Verilog</b> – Paso Doble: .....	31
Diagrama TLD:.....	31
Divisor de Reloj: .....	32
Control Motor a Pasos Unipolar: .....	32



Módulo TLD:.....	33
Código UCF:.....	34
Simulación Paso Doble:.....	34
<b>Código Verilog – Medio Paso:</b> .....	34
Diagrama TLD:.....	34
Divisor de Reloj: .....	35
Control Motor a Pasos Unipolar: .....	35
Módulo TLD:.....	36
Código UCF:.....	37
Simulación Medio Paso:.....	37
<b>Código VHDL – Paso Simple:</b> .....	38
Divisor de Reloj: .....	38
Control Motor a Pasos Unipolar: .....	38
Módulo TLD:.....	39
Código UCF:.....	40
<b>Código VHDL – Paso Doble:</b> .....	40
Divisor de Reloj: .....	40
Control Motor a Pasos Unipolar: .....	41
Módulo TLD:.....	41
Código UCF:.....	42
<b>Código VHDL – Medio Paso:</b> .....	42
Divisor de Reloj: .....	42
Control Motor a Pasos Unipolar: .....	43
Módulo TLD:.....	44
Código UCF:.....	45
<b>Motor a Pasos Bipolar NEMA 17 y su Controlador A4988</b> .....	45
Pasos del Motor a Pasos Bipolar: Completo, Medio, 1/4, 1/8 y 1/16 de Paso en <b>Arduino</b> .....	49
<b>Código Arduino</b> - Paso Doble o Completo:.....	51
<b>Código Arduino</b> – 1/2 Paso, 1/4 de Paso, 1/8 de Paso y 1/16 de Paso: .....	51
TLD Motor a Pasos Bipolar: Paso Completo, Medio, 1/4, 1/8 y 1/16 en <b>Verilog y VHDL</b> .....	52
<b>Código Verilog</b> - Paso Doble, 1/2, 1/4, 1/8 y 1/16:.....	54
Diagrama TLD:.....	54
Divisor de Reloj: .....	55

Control Motor a Pasos Bipolar:.....	55
Módulo TLD:.....	56
Código UCF:.....	56
Simulación Paso Completo, Medio, 1/4, 1/8 y 1/16:.....	57
<b>Código VHDL - Paso Doble, 1/2, 1/4, 1/8 y 1/16: .....</b>	<b>57</b>
Diagrama TLD:.....	57
Divisor de Reloj: .....	58
Control Motor a Pasos Bipolar:.....	59
Módulo TLD:.....	59
Código UCF:.....	60
Referencias: .....	61



# Partes y Tipos de Motores Eléctricos AC/DC

## Tipos de Motores Eléctricos AC/DC

A continuación, se presenta la clasificación de los motores eléctricos:

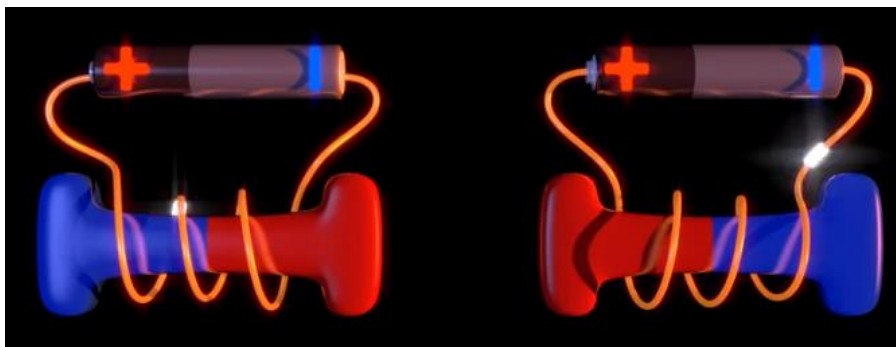
- **Motores DC:** Son impulsados por medio de corriente directa (pilas, rectificadores, etc).
  - **Motores con Escobillas:** Son motores que **realizan su conmutación con unas terminales llamadas escobillas**, que debido a su fricción deterioran el motor a través del tiempo.
    - ✚ **Motor DC Sencillo:** **Motor que gira continuamente sin control** a altas velocidades.
    - ✚ **Motorreductor:** Es un motor DC con escobillas sencillo, pero que cuenta con una serie de engranajes en su punta que reducen su velocidad, pero aumentan su torque (fuerza de rotación). **Este motor gira de forma continua sin control.**
    - ✚ **Servomotor:** El servomotor es un motor DC con escobillas sencillo, pero que cuenta con el control **más preciso** de posición y velocidad de todos los motores eléctricos, debido a que tiene una entrada de retroalimentación. **Este motor puede detenerse en un punto deseado y mantener esa posición.**
  - **Motores sin Escobillas (Brushless) o BLDC:** Son motores que **realizan su conmutación sin las terminales llamadas escobillas, lo hacen a través de controladores externos** y duran más porque como no tienen escobillas, no se deterioran al generar fricción cuando giran, por lo que son más rápidos, fuertes y mejores al usarse como generadores eléctricos.
    - ✚ **Motor BLDC Outrunner:** Es un motor trifásico, por lo que cuenta con 3 terminales para controlar el motor (A, B y C) y en su base tiene 3 sensores de efector hall, que ayudan a determinar la posición del rotor. **El estator (parte fija) de este se encuentra en su centro y su rotor (parte que gira) en su perímetro. Este motor gira de forma continua sin control.**
    - ✚ **Motor BLDC Inrunner:** Es también un motor trifásico que cuenta con 3 terminales para controlar el motor y 3 sensores de efector hall para determinar la posición de su rotor. Su principal diferencia con el motor **Outrunner** es que **el estator (parte fija) de este se encuentra en su perímetro y su rotor (parte que gira) en su centro**, de la misma forma como pasa con los **motores DC con escobillas**. **Este motor gira de forma continua sin control.**
    - ✚ **Motor a pasos (o Motor paso a paso):** Es un motor unipolar o bipolar que cuenta con 4 o 2 bobinas para controlar su movimiento, dependiendo de si es unipolar o bipolar sus bobinas tienen un punto común de alimentación o no. Además, en la punta de cada bobina cuenta con conductores ferromagnéticos en forma de dientes que crearán los pasos del motor, haciendo que gire algo lento, pero proporcionándole **gran precisión** en su control de posición y velocidad. **Este motor puede detenerse en un punto deseado y mantener esa posición.**
- **Motores AC:** Funcionan mediante el suministro de corriente alterna (toma de luz o generador AC).
  - **Motor Síncrono:** La velocidad del rotor es igual a la velocidad del campo magnético en el estator. **Este motor gira de forma continua sin control.**
  - **Motor Asíncrono:** La velocidad del rotor NO es igual a la velocidad del campo magnético en el estator. **Este motor gira de forma continua sin control.**

## Bobinas y Electroimanes

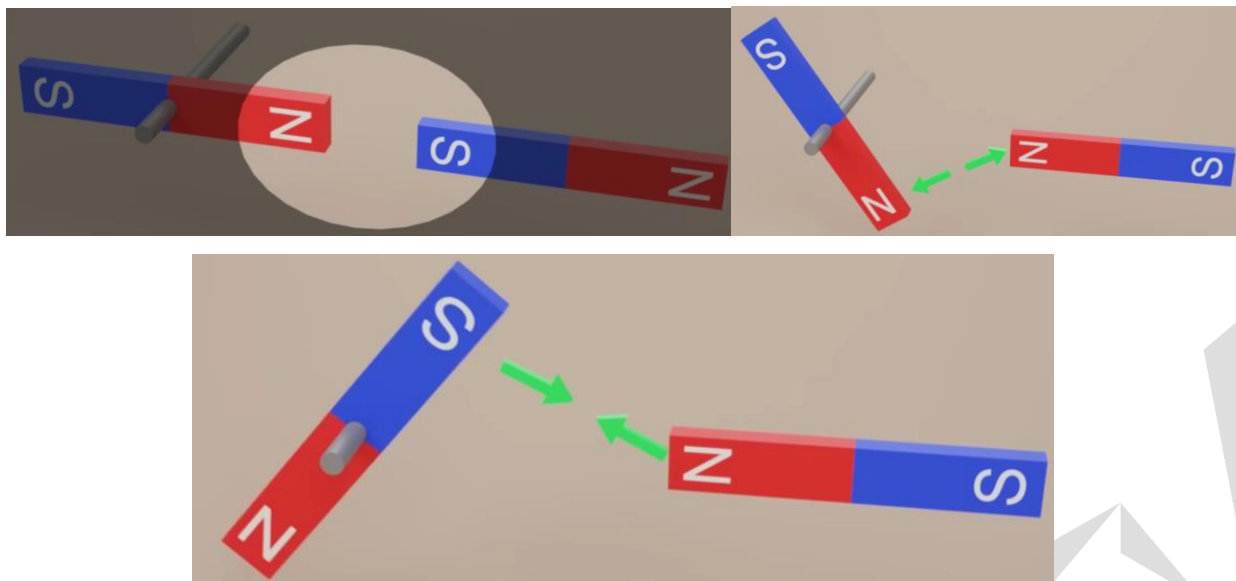
Para comprender el funcionamiento de cualquier motor eléctrico se debe conocer el funcionamiento básico de un electroimán, un electroimán suele estar compuesto de:

- **Un metal ferromagnético:** Es un material que es conductor de campo magnético.
- **Una bobina o inductor:** Es un alambre compuesto de un metal conductor eléctrico (usualmente cobre) enrollado alrededor del material ferromagnético mencionado previamente.

Al activar el electroimán, el metal del núcleo se comportará como un imán mientras exista un flujo de corriente eléctrica en el cable, pudiendo cambiar su polaridad según la forma en la que se orienten las vueltas del embobinado o invirtiendo el sentido de la fuente de alimentación (la pila). **Al hacerlo por mucho tiempo el metal se calienta por un efecto llamado corriente de Foucault (o Eddy).**

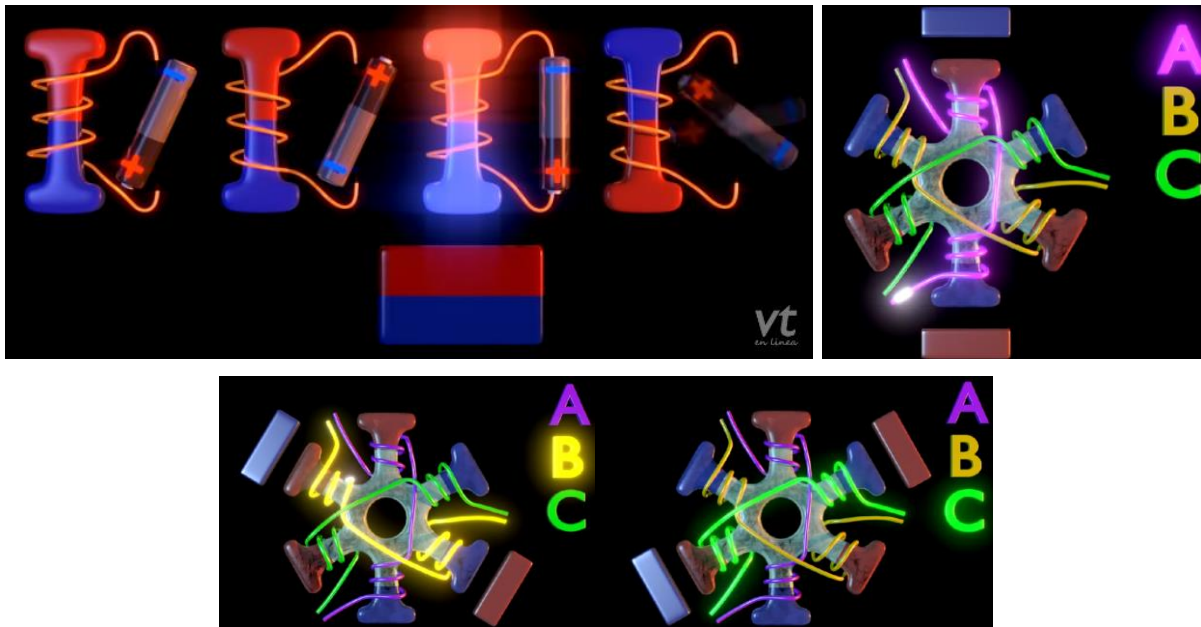


Esta acción convierte un metal ferromagnético cualquiera en un imán, y por medio de un magneto real (imán permanente) u otro electroimán se puede atraer o repeler sus polos (siempre y cuando el electroimán se encuentre encendido), generando así el **movimiento mecánico del motor**, siguiendo el principio magnético donde se dicta que **polos iguales se repelen y polos opuestos se atraen**.

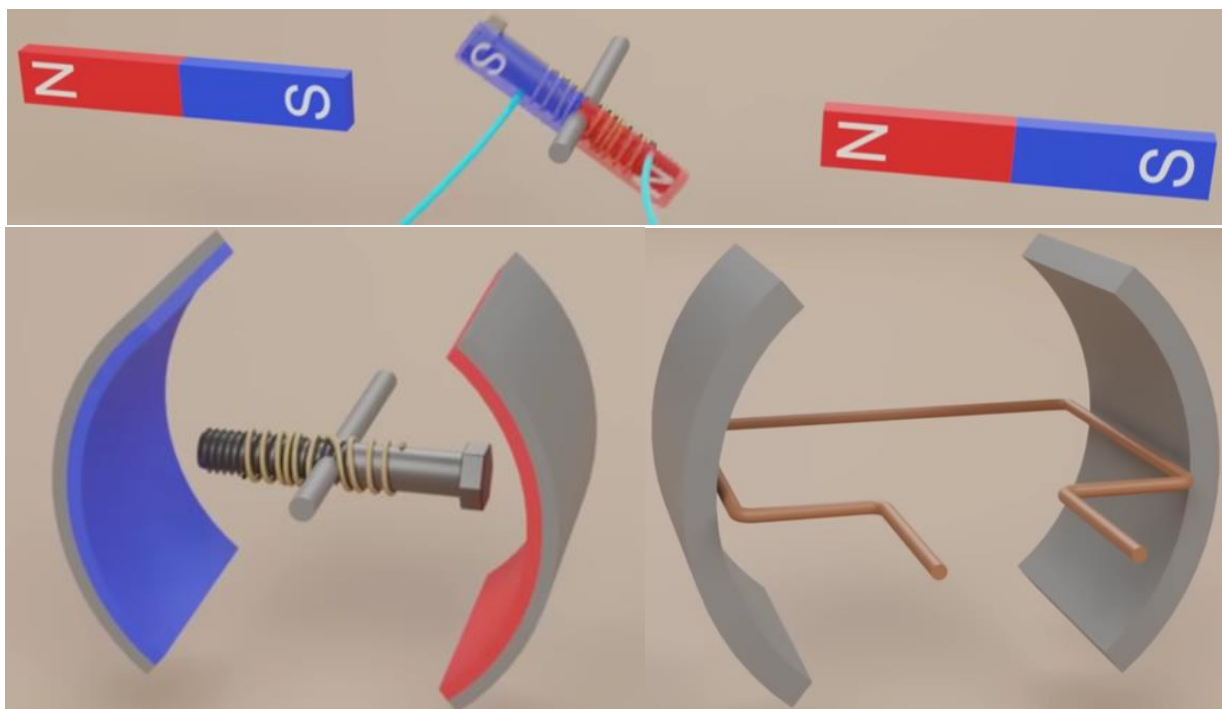


Basándonos en los conceptos explicados previamente, existen dos maneras globales de efectuar el movimiento de un motor eléctrico, que serán explicadas a continuación:

1. **Giro del imán permanente:** Se puede contar con varios **electroimanes estáticos** para **mover el imán permanente de un motor** a la posición donde se requiera, en este ejemplo sencillo, solo uno debe estar encendido a la vez. El concepto es más aplicado en **motores DC sin escobillas**.



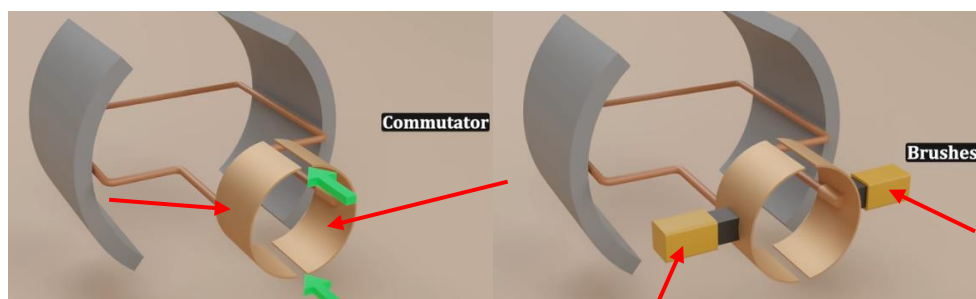
2. **Giro del electroimán:** Se puede contar con dos o más **imanes permanentes estáticos** para **mover el electroimán de un motor**, donde en vez de tener una bobina con un material ferromagnético dentro, se usarán varias bobinas que cambien el sentido de su corriente para invertir el sentido de su polaridad y al hacerlo serán movidas por medio dos o más imanes permanentes a su lado. Este concepto es más aplicado en **motores DC con escobillas**.



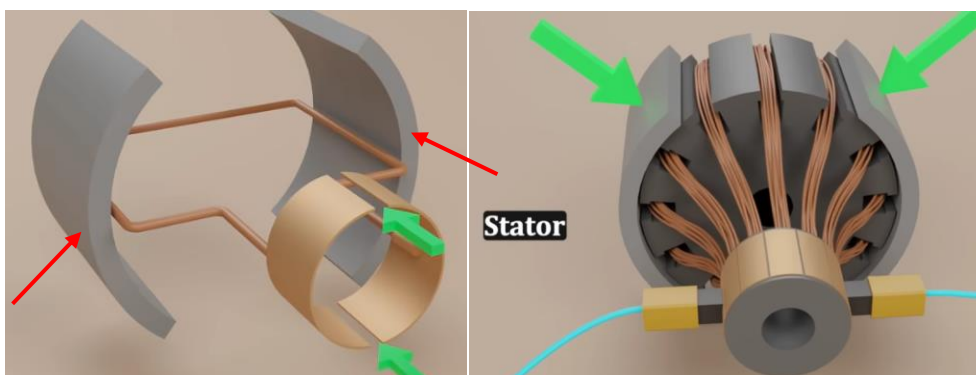
## Partes de un Motor DC con Escobillas: Conmutador, Estator, Armadura y Rotor

Ahora que ya conocemos el efecto básico que mueve a los motores, debemos conocer el nombre de cada una de sus partes.

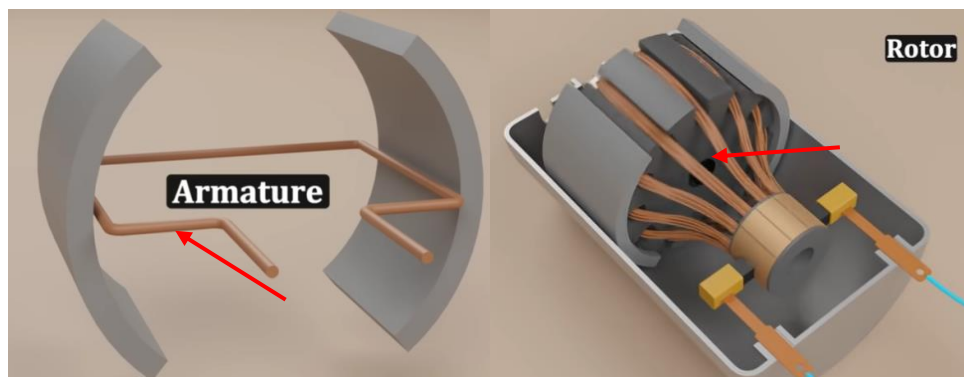
Los motores con escobillas se alimentan con corriente directa y la parte que realiza su conmutación, o sea el cambio de dirección en la corriente de las bobinas pertenecientes a los electroimanes para generar su movimiento mecánico, son las mismas **escobillas** (o **brushes**) en conjunto con el **conmutador** (o **colector**), ya que estas invierten el sentido de la polaridad del **electroimán interno que gira**.



Los imanes permanentes usados para mover los electroimanes que se encienden y apagan en el motor son denominados como **estator** (stator), siendo la parte que se mantiene estática en el motor.

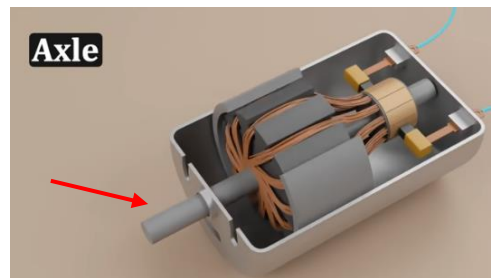


El metal ferromagnético usado como núcleo del electroimán se llama **rotor** y está conformado por varias láminas aisladas en vez de ser completamente sólido, para así evitar que se creen corrientes parásitas llamadas corrientes de Foucault (o Eddy) que lo calienten en vez de moverlo, además las bobinas que lo rodean se llaman **armadura** (armature) y la combinación de ambas es la **parte que gira en el motor**.





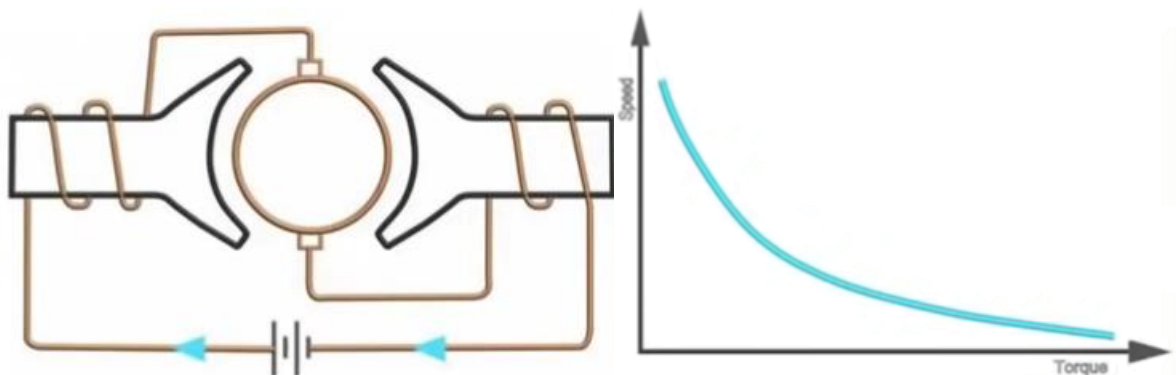
En la parte central del rotor se coloca el **eje del motor** (axle) y con esto finalizamos de nombrar todas las partes importantes de un motor DC con escobillas.



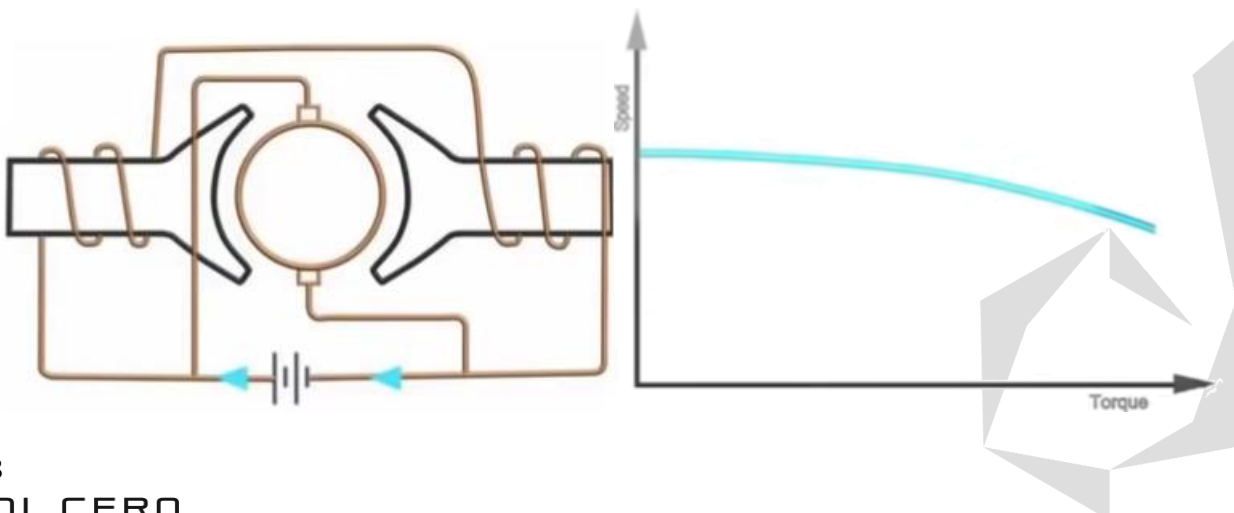
### Conexión en Serie o Paralelo del Estator y Rotor de un Motor DC

La estructura descrita anteriormente es la más común en motores DC con escobillas, pero a veces el **estator** no es un imán permanente, sino otro electroimán que puede estar conectado en serie o en paralelo con el electroimán del **rotor**, logrando así las diferentes características descritas a continuación:

- **Motor DC conectado en Serie:** Este motor tiene un buen torque o par (y velocidad) de arranque (al iniciar el movimiento del motor), pero su velocidad disminuye drásticamente al ponerle una carga, osea un peso que el motor mueva.



- **Motor DC conectado en Paralelo (Motor en Derivación o Shunt):** Este motor tiene un bajo torque (y velocidad) de arranque (al iniciar el movimiento del motor), pero es capaz de mantener una velocidad casi constante al ponerle una carga, osea un peso que el motor mueva.



# Manejo de la Corriente en las Bobinas de los Motores

Como el corazón de todos los tipos de motores es el electroimán, que está conformado por inductores (bobinas), una de sus propiedades fundamentales es que se opone a cambios bruscos en la corriente del circuito debido a su inductancia.

De acuerdo con la ley de Faraday, la autoinductancia es la capacidad de una bobina para generar una fuerza electromotriz (fem o emf) en respuesta a un cambio brusco en la corriente que lo atraviesa. La fem en realidad es una tensión eléctrica con polaridad inversa a la corriente cambiante, lo que causa una oposición al flujo de corriente y ralentiza su cambio, su ecuación es la siguiente, donde:

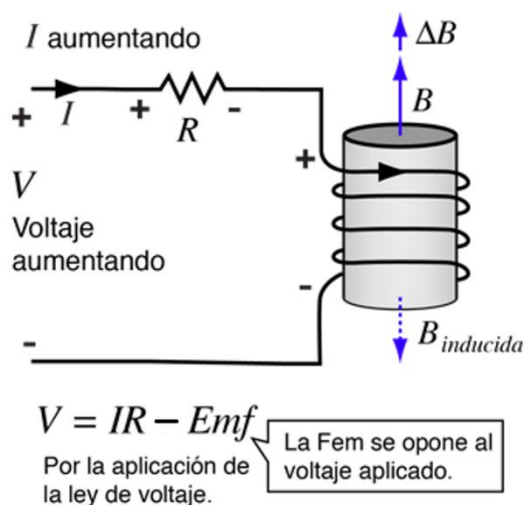
$B$  = Flujo magnético inducido en un material [Tesla]

$V_{fem}$  = Tensión de fuerza electromotriz inducida [V]

$L$  = Inductancia [Henrys]

$i$  = Corriente cambiante del circuito [Amperes]

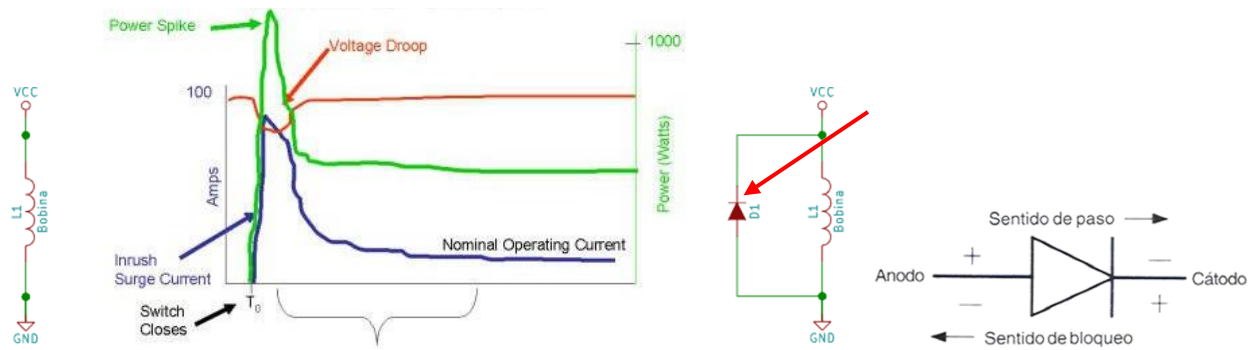
$$V_{fem} = V_{Emf} = L \left( \frac{di}{dt} \right) [V]$$



## Diodo de Marcha Libre

Una situación muy sencilla que se puede analizar para ejemplificar el efecto de la inductancia en los motores eléctricos es cuando se alimenta uno directamente:

- 1) En un inicio su bobina no está energizada, pero al conectarse por primera vez, se creará un campo magnético en ella.
- 2) **Al apagar su alimentación, la corriente cambiará bruscamente a ser cero**, situación que tratará de ser impedida por el inductor del motor.
  - a. Esto ocasionará que se cree una enorme **fem (tensión en sentido contrario)** en las bobinas del motor, que tratará de impedir que la corriente en la bobina cambie a ser cero bruscamente, aunque obviamente no lo podrá lograr ya que la alimentación ha sido removida. **Esta tensión generada puede ser hasta 10 veces mayor que la inicial en el circuito**, pero en sentido contrario, lo cual dañará a la fuente de alimentación.
- 3) Para evitar que la tensión inducida en el circuito dañe la fuente o los demás dispositivos electrónicos, se agrega el **diodo de marcha libre o diodo en antiparalelo**, el cual es un simple diodo, conectado en paralelo al motor y puesto en sentido contrario a la alimentación.
  - a. **Cuando el circuito esté encendido:** El diodo de marcha libre no hará nada.
  - b. **Cuando el circuito se apague:** Se creará la **fem (tensión en sentido contrario)** y es ahí cuando el **diodo de marcha libre** permitirá que esa tensión fluya a través de él, dejando que descargue su corriente sin dañar a los demás dispositivos electrónicos.



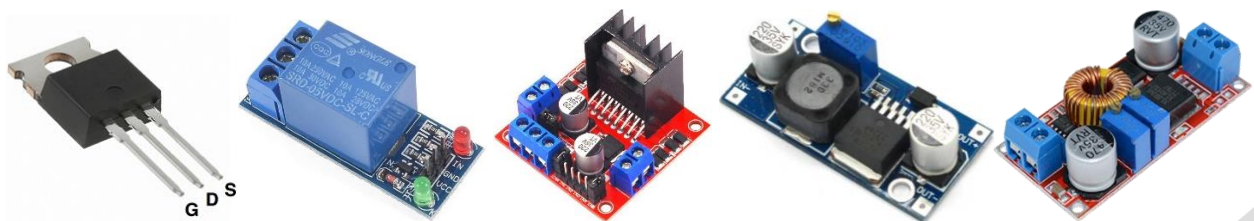
## Etapa de Control y Etapa de Potencia en un Circuito con Motores

### Etapa de Potencia: Corriente Demandada por el Motor al Agregar una Carga

Cuando un motor se enfrenta a una **carga** (fuerza o peso que se opone a su giro normal), debe generar suficiente torque para intentar superarla. El **torque** que ejerce el motor en su eje **está relacionado con la corriente que fluye a través de sus bobinas**. A medida que aumenta la carga, el motor necesita generar más **torque**, por lo que requiere un aumento en su **corriente** para superar la fuerza de oposición ocasionada por la **carga** y mantener su velocidad.

Por esta cuestión es que el circuito se divide en dos partes, una llamada **etapa de potencia** y otra llamada **etapa de lógica o control**:

- **Etapa de Potencia:** En esta etapa se manejan **tensiones o corrientes grandes** que pueden ser cambiantes o no, como las ocasionadas en un motor cuando se le agrega una **carga** que se opone a su rotación.
  - La etapa de potencia generalmente involucra componentes como:
    - Transistores MOSFET, relevadores (relés), controladores de motores (drivers), puentes H, convertidores de CD a CD boost, buck, etc.



- Motores CD con escobillas sencillos, motorreductores, servomotores, motores sin escobillas (brushless o BLDC), motores a pasos, motores AC, etc.



- **Etapa de Control:** Esta etapa maneja **tensiones y corrientes pequeñas**, es la parte del circuito que controla la **etapa de potencia**.
  - La etapa de control generalmente involucra componentes como:
    - Microcontroladores, FPGA, CPU (Raspberry Pi), sensores, circuitos lógicos, transistores BJT, amplificadores operacionales y otros dispositivos que operan a niveles de potencia más bajos o se centran en el procesamiento y la toma de decisiones.



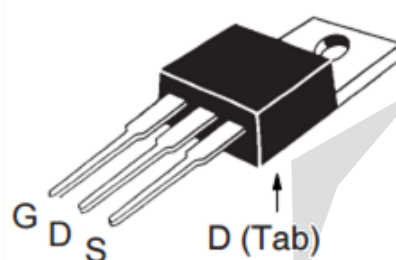
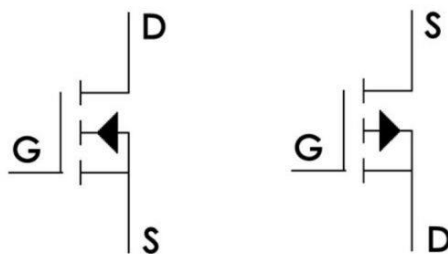
En pocas palabras, la etapa de control es el cerebro del circuito y la etapa de potencia es su músculo.

### Etapa de Control: Aislamiento de los Circuitos Lógicos

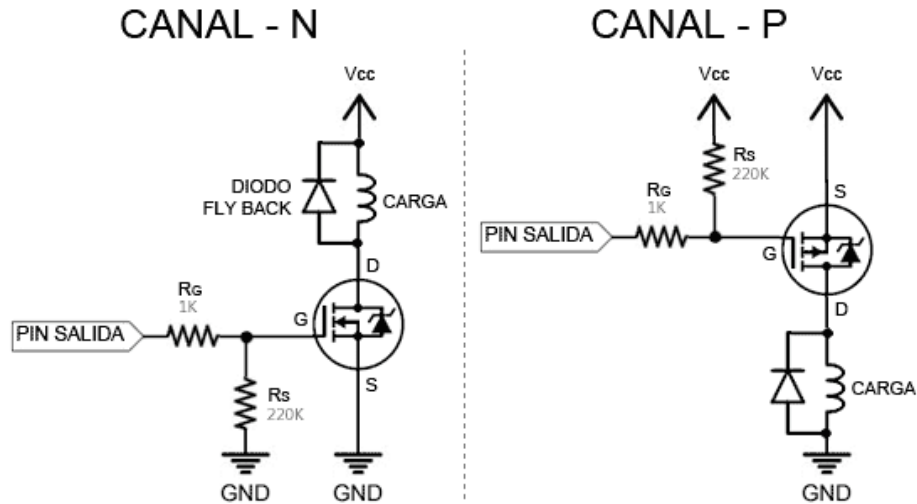
El objetivo principal de separar el circuito en una **etapa de control** y otra de **potencia** es que **no se dañen los componentes electrónicos que no manejan altos niveles de tensión y corriente**, para ello se utilizan ciertos dispositivos electrónicos que aíslan la etapa de control, separándola de la etapa de potencia, **incluso se utilizan fuentes de alimentación distintas en cada etapa**. Se suelen utilizar los 3 siguientes componentes electrónicos para realizar el aislamiento del circuito:

- **Transistor MOSFET:** Es un dispositivo electrónico que puede utilizarse en una configuración de **amplificador** o **interruptor** para separar la etapa de potencia y la etapa lógica.
  - **Amplificador:** Los transistores **pueden amplificar señales de control de bajo voltaje y corriente provenientes de la etapa lógica** para controlar dispositivos electrónicos que consumen mayor tensión y corriente en la **etapa de potencia**.
  - **Interruptor:** Los transistores **pueden usarse como interruptores para encender o apagar la fuente u otros dispositivos electrónicos de la etapa de potencia** por medio de una señal proveniente de la **etapa lógica**.
    - Se utilizan las diferentes configuraciones del transistor MOSFET, llamadas canal N o P, según la polaridad (dirección) de la tensión en el circuito.

#### MOSFET Canal N MOSFET Canal P



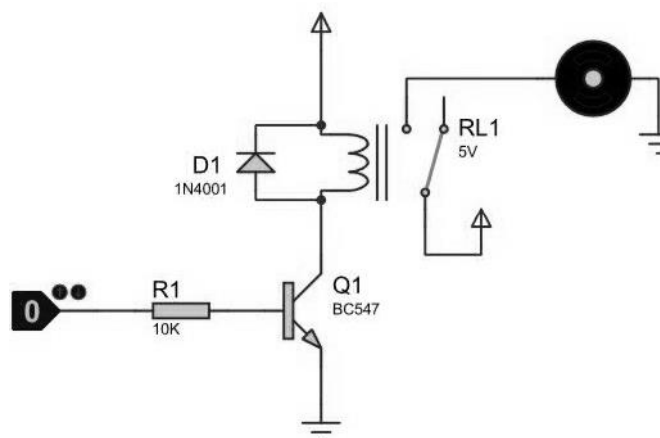
- A continuación, se muestra un ejemplo de conexión que aísla ambas etapas a través de un **transistor MOSFET**:



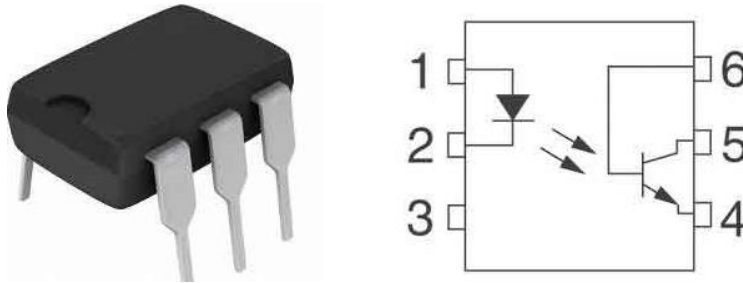
- **Relevador:** Un relevador o relé es un interruptor electromagnético que utiliza un electroimán interno para conectar o desconectar dos contactos mecánicos.
  - Los relés proporcionan un aislamiento eléctrico entre la **etapa de potencia** y la **etapa de control**, ya que no hay conexión eléctrica directa entre ellas.
  - Se pueden usar relevadores de forma individual o adquirir módulos que tienen varios.



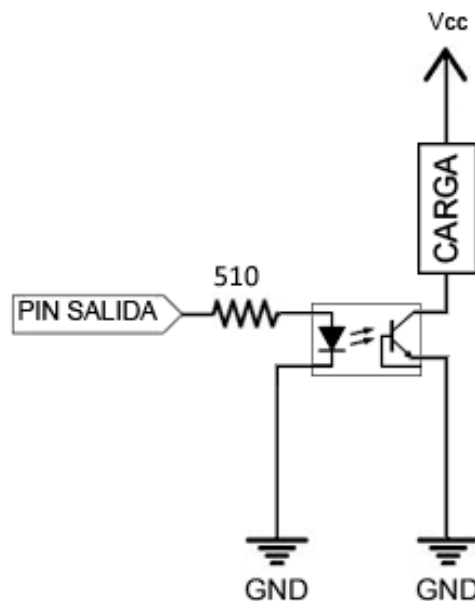
- A continuación, se muestra un ejemplo de conexión que aísla ambas etapas a través de un **relevador o relé**:



- **Optoacoplador (optoaislador):** Es un dispositivo electrónico que utiliza una combinación de un emisor de luz (generalmente un diodo emisor de luz o LED) y un receptor de luz (generalmente un fototransistor) para proporcionar un aislamiento eléctrico entre la **etapa de potencia** y la **etapa lógica**.
  - El optoacoplador proporciona un acoplamiento óptico; es el mejor aislamiento que se le puede dar a la **etapa de control**, ya que la protege de interferencias electromagnéticas, sobretensiones, ruidos y otros problemas que podrían generarse en la **etapa de potencia**.



- A continuación, se muestra un ejemplo de conexión que aísla ambas etapas a través de un **optoacoplador**:



En los ejemplos posteriores donde se explicará el funcionamiento y control de los motores no se separa el circuito en una **etapa de potencia** y **etapa lógica**, esto se realiza por simplicidad en la demostración, pero ya en aplicaciones reales si se debe realizar el aislamiento de ambas partes del circuito, **si no se realiza la separación, cuando el motor demande más potencia, dañará al microcontrolador, FPGA o Raspberry Pi** e incluso podría dañar el puerto serial que conecta la tarjeta de desarrollo con la computadora.

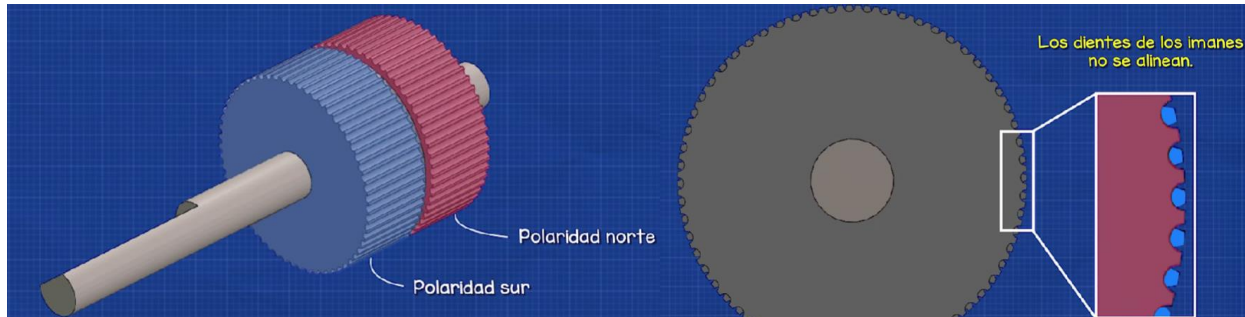




# Motor a Pasos

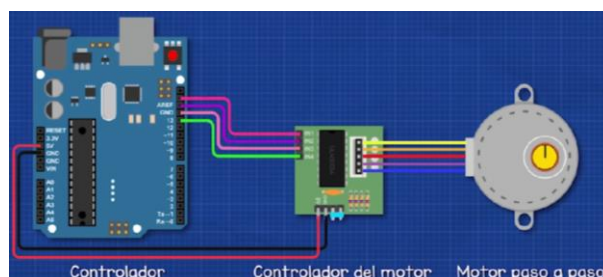
El motor a pasos, stepper o motor paso a paso es un motor sin escobillas (brushless) cuyo **estator** se encuentra en su **perímetro** y su **rotor** en su **parte central, conformado por un imán permanente que tiene dos mitades (polo norte y sur)** conocidas como **copas**, por lo que su centro gira como lo hace un motor DC convencional, aunque este usualmente tiene una velocidad de rotación menor.

Las copas del rotor cuentan con dientes en ambos polos del magneto, desalineados entre sí, cuya función es ocasionar el movimiento del motor en forma de pequeños pasos, por lo que es muy preciso.



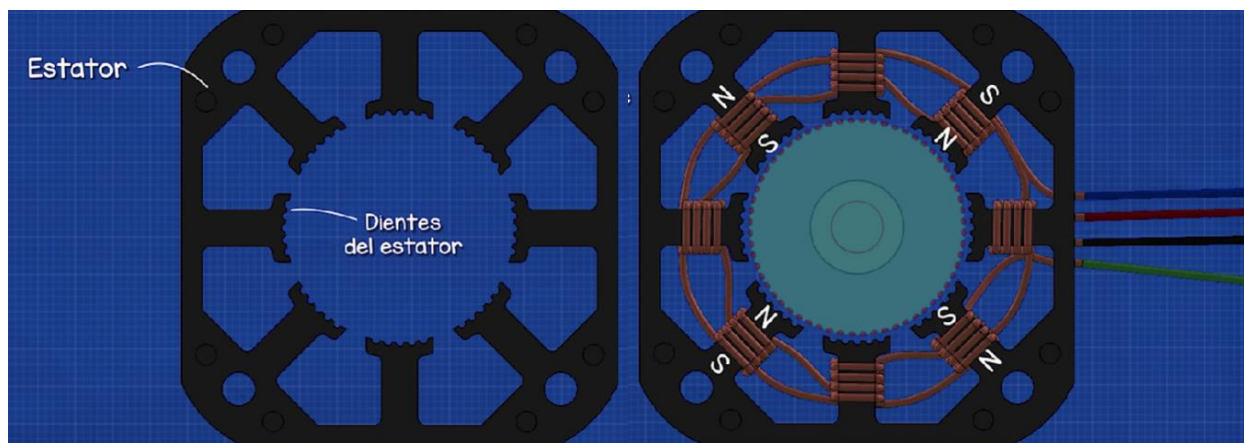
El motor a pasos puede ser "unipolar" o "bipolar", esto se refiere a la configuración de sus bobinas y a la forma en que se activan para generar su movimiento, infiriendo así en el módulo **controlador del motor** que se debe utilizar, también se incluye otro **controlador** que indique la secuencia de encendido y apagado de las bobinas; puede ser un Arduino (microcontrolador), FPGA o un circuito especializado:

- **Unipolar:** Los motores a pasos unipolares tienen cuatro fases (A, B, C y D), las cuales tienen un punto en común y cada bobina se conecta a una señal independiente de tensión.
  - Se utilizan debido a su pequeño tamaño, aunque generalmente tienen menos torque que los motores a pasos bipolares, en él las bobinas se activan secuencialmente aplicando voltaje a los extremos de las bobinas en relación con el punto común.
  - **Necesita un controlador con configuración de transistores Darlington:** Este amplifica la corriente de salida entregada a las 4 bobinas del motor a pasos unipolar.
- **Bipolar:** Los motores a pasos bipolares tienen dos fases (A y B) para controlar el movimiento del motor, estas no tienen un punto en común y cada bobina se conecta a una señal independiente.
  - Son ampliamente utilizados debido a su mayor torque en comparación con los motores a pasos unipolares, aunque su tamaño y peso también aumentan.
  - **Necesita un controlador con configuración de puente H:** Este enciende y apaga cada grupo de bobinas A y B en secuencia, para así invertir la dirección de la corriente en sus fases y cambiar su polaridad para efectuar el movimiento del motor.

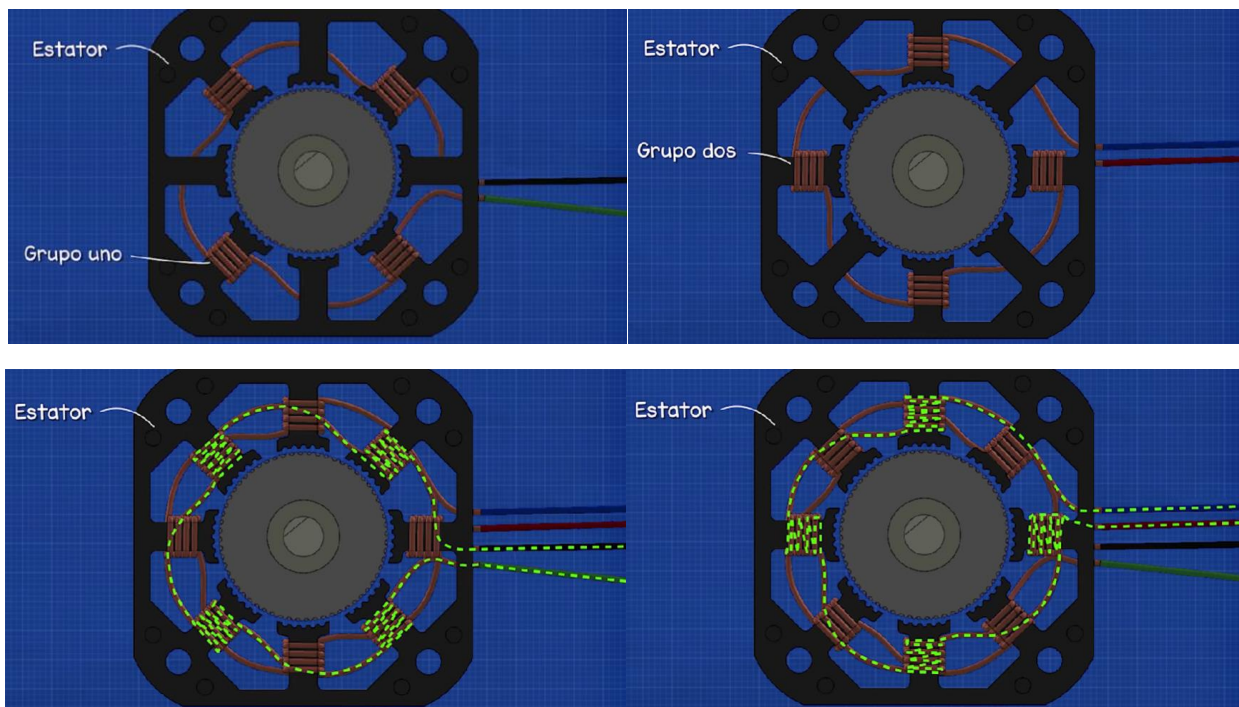


La punta de cada bobina del estator cuenta con conductores ferromagnéticos en forma de dientes que, entrando en contacto con los dientes de las copas, moverán el motor un paso a la vez, proporcionándole así **gran precisión** en su control de posición y velocidad. El **rotor** tiene **más dientes** que el **estator**, por lo que no todos los dientes pueden alinearse al mismo tiempo.

Cuando las bobinas del motor reciben corriente eléctrica, estas formarán campos electromagnéticos polarizados que se pasarán a los dientes del estator, y a su vez interactúan con los dientes del imán permanente del rotor, logrando así que el motor gire cuando estas se activen en una secuencia de pasos.

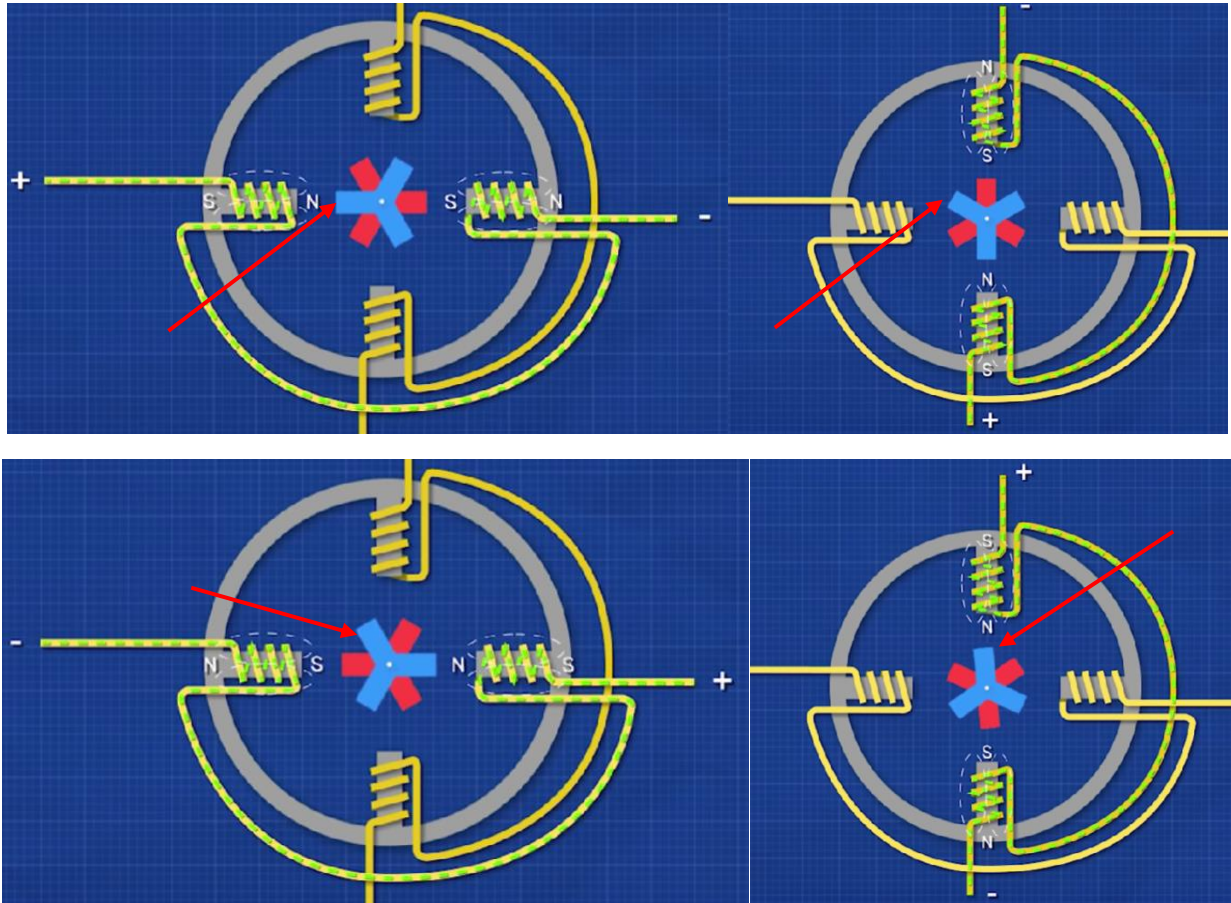


El giro del **motor bipolar** se logra porque **se conmuta la secuencia en la que se encienden y apagan sus 2 grupos de bobinas A y B**, invirtiendo además la polaridad de sus electroimanes con las dos terminales de cada una y un **punto H**, en el **motor unipolar** pasa lo mismo, pero con las 4 fases A, B, C y D y su terminal común, aunque en ese caso no se invierte el sentido de la corriente, solo se encienden secuencialmente y la corriente que se les entrega a las 4 bobinas se aumenta con **transistores Darlington**.

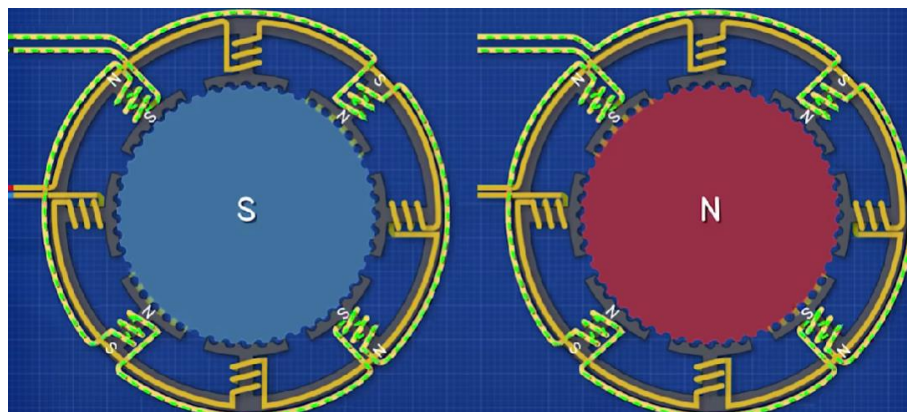




El número de bobinas y dientes darán como resultado el grado de rotación que se efectúa al dar cada paso del motor. En el siguiente ejercicio de ejemplo se crea un paso de 30° con 4 bobinas y 6 dientes, como se puede observar en las imágenes de abajo, logrando así que, cada vez que se active una bobina, intercaldando su polaridad, se dé una rotación de 30°:



Por lo tanto, es evidente ver que, en función del número de dientes del estator y el rotor, es como se mueve el motor, pero una última cosa que vale la pena mencionar es que, **en la configuración del motor a pasos bipolar, los dientes del polo sur de la copa se alinean con las bobinas de polaridad norte del estator y los dientes del polo norte de la copa hacen lo contrario**, es por eso que los motores bipolares proporcionan un mayor torque que los **unipolares**.

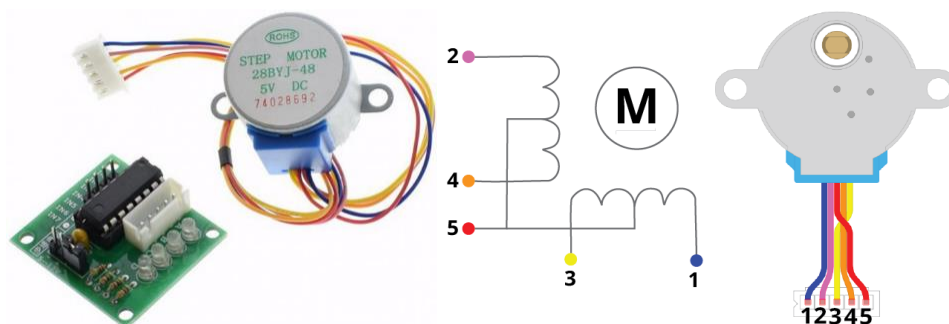


## Motor a Pasos Unipolar 28BYJ-48 y su Controlador ULN2003

Uno de los motores a pasos **unipolares** más comunes es el modelo **28BYJ-48**, tiene 4 bobinas A, B, C y D, por lo que cuenta con 5 cables, uno por cada bobina y una terminal común, manejadas por medio del controlador **ULN2003** que es una **configuración de transistores Darlington** cuya función es aumentar la corriente entregada a las 4 fases del motor. Las características del motor y controlador son:

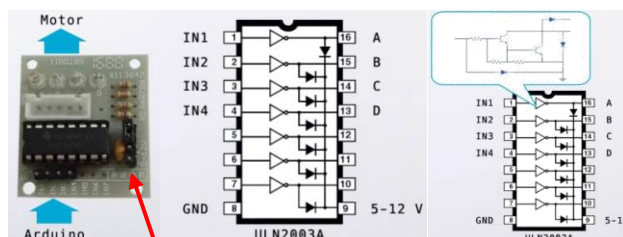
### Motor a pasos 28BYJ-48:

- El motor recibe 5 V de alimentación en la **terminal común (5)** de sus bobinas.
- Cuenta con 4 fases o bobinas **A, B, C y D** que consumen en promedio 60mA en cada una.
- **La velocidad de rotación del motor** se controla por medio de una frecuencia mínima de 1 Hz y máxima de 800 Hz. *Usar frecuencias menores a 100Hz por mucho tiempo calienta el motor.*
- Cuenta con un sistema de engranes que reducen su velocidad y aumentan su torque, por lo que el motor debe dar un total de 2,048 o 4,096 pasos para dar una vuelta completa, debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor (**reducción 1/64**), dando un giro por de  $5.625^\circ$  por paso al eje final.
    - **Para que el rotor original gire una vuelta se necesitan 8 ciclos de la secuencia de pasos**, el **paso simple y doble** constan de **4 acciones**, dando un giro de  $11.25^\circ$  por paso al eje original (**32 pasos totales**), pero el **medio paso consta de 8 acciones**, dando un giro de  $5.625^\circ$  por paso al eje original (**64 pasos totales**).



### Controlador de Motor ULN2003:

- El controlador recibe de 5 a 12V de alimentación en sus pines laterales (- + 5-12V).
- Cuenta con una compuerta NOT en cada uno de sus pines, por lo que invierte su nivel lógico.
  - Esto es de gran utilidad porque como la **terminal común** del **motor** recibe 5V, cada bobina se activa con un 0 lógico, en vez de con un 1 lógico, pero en el código se maneja con un 1.
- En sus pines **IN1**, **IN2**, **IN3** e **IN4** recibe las señales de conmutación del microcontrolador o FPGA, las invierte y con sus pines **A**, **B**, **C** y **D** (que tienen leds rojos) las entrega a las 4 bobinas del motor.
  - Puede manejar como máximo corrientes de salida de 500mA si el motor lo requiere.



## Pasos del Motor a Pasos Unipolar: Simple, Doble y Medio en Arduino.

Existen varias maneras para **controlar** el **motor a pasos unipolar**, se describirán 3 maneras, donde:

- Un número de acciones descritas en una secuencia de “**paso simple**”, “**paso doble**” o “**medio paso**” describen las señales dirigidas a las 4 bobinas del motor a pasos unipolar, donde **8 ciclos de dichas secuencias de pasos** giran una vez al rotor original, pero debido al sistema de engranajes del motor, **64 vueltas del rotor original corresponden a una sola vuelta del eje del motor final**.
  - Este motor puede detenerse en un punto deseado y mantener esa posición.
  - Dependiendo de la **precisión** del movimiento y el **torque** que se quiera obtener se usa un tipo de paso diferente.
- Se controla la **velocidad de rotación** del motor por medio de la **frecuencia con la que se mandan las secuencias de sus pasos**, pero estará limitada por el sistema de engranajes de su salida.

Paso Simple			
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

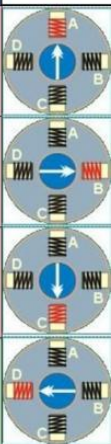
Paso Doble			
1	1	0	0
0	1	1	0
0	0	1	1
1	0	0	1

Medio Paso			
1	0	0	0
1	1	0	0
0	1	0	0
0	1	1	0
0	0	1	0
0	0	1	1
0	0	0	1
1	0	0	1

Se mostrarán modelos simplificados del motor unipolar con el fin de explicar las secuencias de sus pasos, en ellos el motor cuenta con solo 4 bobinas y la dirección de su rotor central está indicada con una flecha.

- Paso simple, control de onda o wave drive:** En este paso solo **se activa una fase a la vez** en el motor para accionar su rotación, por lo que:
  - La **secuencia del ciclo es de 4 acciones** y como resultado **la rotación no es muy suave**.
    - Se necesitan **32 pasos (8 ciclos de secuencias de 4 acciones)** para que el rotor original dé una vuelta completa, en total se necesitan **2,048 pasos (64 vueltas de 32 pasos cada una)** para que el eje final gire 1 vuelta.
  - El **torque** del motor **es bajo** y soporta máximo **63 gramos de carga (0.03006 [N·m])**.

Paso Simple				
Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON



- 2) **Paso doble, completo o normal:** En este paso **se activan dos fases a la vez** en el motor para accionar su rotación, alcanzando el **torque más grande de las 3 configuraciones**:
- La **secuencia del ciclo es de 4 acciones** y como resultado **la rotación no es muy suave**.
    - Se necesitan **32 pasos (8 ciclos de secuencias de 4 acciones)** para que el rotor original dé una **vuelta completa**, en total se necesitan **2,048 pasos (64 vueltas de 32 pasos cada una)** para que el eje final gire 1 vuelta.
  - Se alcanza el máximo torque** del motor, es el **doble** en comparación con el paso simple y soporta máximo **126 gramos de carga (0.03804 [N·m])**.

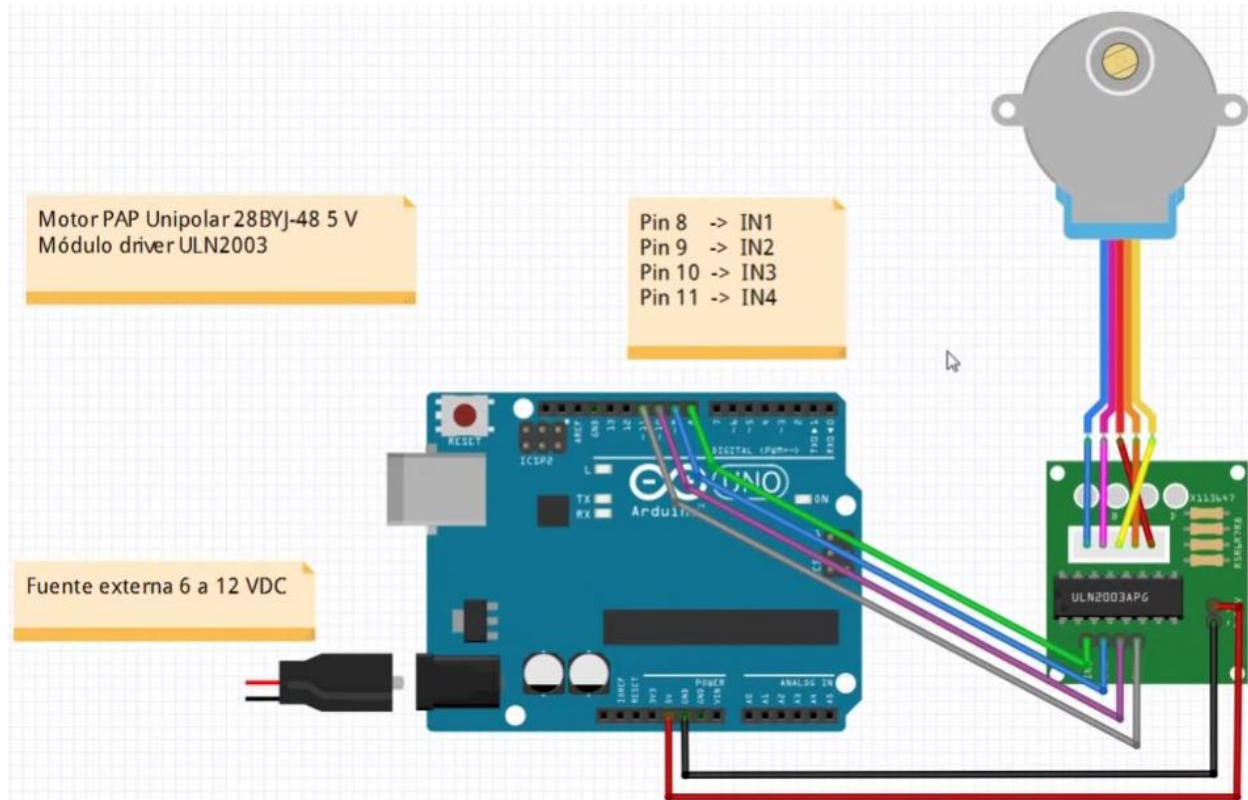
Paso Normal					
Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

- 3) **Medio paso:** En este paso **primero se activa una fase y luego se activan dos fases a la vez** en el motor para accionar su rotación, alcanzando la **mayor precisión de las 3 configuraciones**:
- El **ciclo de rotación es de 8 acciones** y como resultado **la rotación es mucho más suave** que en las configuraciones anteriores, alcanzando la **máxima precisión**.
    - Se necesitan **64 pasos (8 ciclos de secuencias de 8 acciones)** para que el rotor original dé una **vuelta completa**, en total se necesitan **4,096 pasos (64 vueltas de 64 pasos cada una)** para que el eje final gire 1 vuelta.
    - El alcanzar la máxima precisión afecta también en que es el paso más lento.
  - Como en ciertos momentos solo se activa una bobina, **el torque del motor se reduce en comparación con el paso doble**. La **carga** que soporta es aproximadamente **de 63 gramos (0.03006 [N·m])**, aunque como este motor no está hecho para realizar el medio paso, en realidad no se puede saber con exactitud cuanta carga aguanta.

Medio Paso					
Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	OFF	OFF	OFF	
2	ON	ON	OFF	OFF	
3	OFF	ON	OFF	OFF	
4	OFF	ON	ON	OFF	
5	OFF	OFF	ON	OFF	
6	OFF	OFF	ON	ON	
7	OFF	OFF	OFF	ON	
8	ON	OFF	OFF	ON	



## Diagrama de conexión del controlador (Arduino o FPGA), el controlador de motor y el motor a pasos:



## Código Arduino - Paso Simple:

```
/*30.1.1.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En este código se le aplica el Paso Simple.*/
//1_Paso simple: Solo se activa una fase a la vez en el motor para accionar su rotación, dando
//como resultado una precisión de 2,048 pasos y un torque mediocre que soporta 63 gramos.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 20; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
  pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
  pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
  pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
  torque, por lo que el motor debe dar un total de 2,048 pasos para dar una vuelta completa,
  esto debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un
  giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
  necesitan 32 pasos, dando un giro de 11.25° por paso al eje original, ya que a fuerza se
  deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.*/
}
```

```

Esta situación aunada a que:
- Un ciclo del paso simple consta de 4 pasos, da como resultado que un giro completo del rotor
conste de 32/4 = 8 ciclos.
En conclusión, el motor necesita dar:
#pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*64 = 4*8*64 = 2048 pasos para dar una vuelta.
Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
secuencia del paso simple consta de 4 pasos, que se indican dentro del mismo bucle for.*/
for(int i = 0; i < 512; i++){
  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
}

//Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}

```

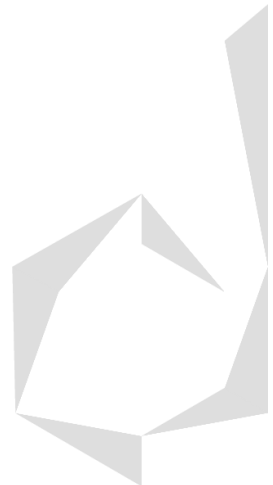
## Código Arduino - Paso Simple - Array:

```

/*30.1.2.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En el código anterior la secuencia se realizó
por medio de instrucciones digitalWrite, que mandan un bit, pero en este caso la programación se
realizará por medio de un array bidimensional que guarde los bits que se deben mandar, extraídos de
la tabla que describe el paso simple, incluida en el archivo 12.-Motor a Pasos.*/
//1 Paso simple: Solo se activa una fase a la vez en el motor para accionar su rotación, dando
//como resultado una precisión de 2,048 pasos y un torque mediocre que soporta 63 gramos.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 2; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.
/*Se utiliza la matriz bidimensional pasoSimple[Filas][Columnas] para indicar los bits que se debe
mandar al motor a pasos unipolar para realizar este tipo de paso, recordemos que el tamaño se cuenta
de forma normal desde 1, pero sus coordenadas se indican desde 0 en un array.*/
int pasoSimple[4][4] = {
  {1, 0, 0, 0},
  {0, 1, 0, 0},
  {0, 0, 1, 0},
  {0, 0, 0, 1}
};

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.

```



```

- segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
El número del pin que recibe este método como primer parámetro se puede declarar directamente
como un número o se puede declarar al inicio del programa como una variable.*/
pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
  torque, por lo que el motor debe dar un total de 2,048 pasos para dar una vuelta completa,
  esto debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un
  giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
  necesitan 32 pasos, dando un giro de 11.25° por paso al eje original, ya que a fuerza se
  deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.
  Esta situación aunada a que:
  - Un ciclo del paso simple consta de 4 pasos, da como resultado que un giro completo del rotor
  conste de 32/4 = 8 ciclos.
  En conclusión, el motor necesita dar:
  #pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*64 = 4*8*64 = 2048 pasos para dar una vuelta.
  Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
  secuencia del paso simple consta de 4 pasos, que se indican dentro del mismo bucle for.*/
  for(int i = 0; i < 512; i++){
    //Bucle for que recorre la matriz bidimensional pasoSimple que describe ese paso.
    for(int j = 0; j < 4; j++){
      /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
      específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
      constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
      digitalWrite(IN1, pasoSimple[j][0]); //Se mandan los bits que activan la bobina A.
      digitalWrite(IN2, pasoSimple[j][1]); //Se mandan los bits que activan la bobina B.
      digitalWrite(IN3, pasoSimple[j][2]); //Se mandan los bits que activan la bobina C.
      digitalWrite(IN4, pasoSimple[j][3]); //Se mandan los bits que activan la bobina D.
      /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
      delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
    }
  }

  //Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}
}

```

## Código Arduino - Paso Doble:

```

/*30.2.1.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En este código se le aplica el Paso Doble.*/
//2 Paso doble: Se activan dos fases a la vez en el motor para accionar su rotación, dando como
//resultado una precisión mediocre de 2,048 pasos, pero un torque máximo que soporta 126 gramos.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 2; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
  pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
  pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
  pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
  torque, por lo que el motor debe dar un total de 2,048 pasos para dar una vuelta completa,
  esto solo se aplica en el paso simple y doble debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un

```



```

giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
necesitan 32 pasos, dando un giro de 11.25° por paso al eje original, ya que a fuerza se
deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.
Esta situación aunada a que:
- Un ciclo del paso doble consta de 4 pasos, da como resultado que un giro completo del rotor
se conforme de 32/4 = 8 ciclos.
En conclusión, el motor necesita dar:
#pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*4 = 4*8*64 = 2048 pasos para dar una vuelta.
Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
secuencia del paso simple consta de 4 pasos, que se indican dentro del mismo bucle for.*/
for(int i = 0; i < 512; i++){
  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
  digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
  digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
  digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
}

//Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}

```

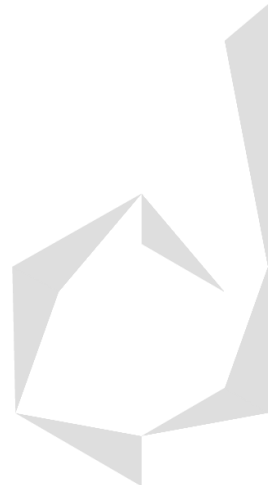
## Código Arduino - Paso Doble - Array:

```

/*30.2.2.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En el código anterior la secuencia se realizó
por medio de instrucciones digitalWrite, que mandan un bit, pero en este caso la programación se
realizará por medio de un array bidimensional que guarde los bits que se deben mandar, extraídos de
la tabla que describe el paso doble, incluida en el archivo 12.-Motor a Pasos.*/
//2_Paso doble: Se activan dos fases a la vez en el motor para accionar su rotación, dando como
//resultado una precisión mediocre de 2,048 pasos, pero un torque máximo que soporta 126 gramos.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
//El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 400; //Cambiano esta frecuencia se cambia la velocidad de rotación del motor.
//Se utiliza la matriz bidimensional pasoDoble[Filas][Columnas] para indicar los bits que se debe
mandar al motor a pasos unipolar para realizar este tipo de paso, recordemos que el tamaño se cuenta
de forma normal desde 1, pero sus coordenadas se indican desde 0 en un array.*/
int pasoDoble[4][4] = {
  {1, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 1},
  {1, 0, 0, 1}
};

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:

```





```

- primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
- segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
El número del pin que recibe este método como primer parámetro se puede declarar directamente
como un número o se puede declarar al inicio del programa como una variable.*/
pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
  torque, por lo que el motor debe dar un total de 2,048 pasos para dar una vuelta completa,
  esto solo se aplica en el paso simple y doble debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un
  giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
  necesitan 32 pasos, dando un giro de 11.25° por paso al eje original, ya que a fuerza se
  deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.
  Esta situación aunada a que:
  - Un ciclo del paso doble consta de 4 pasos, da como resultado que un giro completo del rotor
  se conforme de 32/4 = 8 ciclos.
  En conclusión, el motor necesita dar:
  #pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*64 = 4*8*64 = 2048 pasos para dar una vuelta.
  Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
  secuencia del paso simple consta de 4 pasos, que se indican dentro del mismo bucle for.*/
  for(int i = 0; i < 512; i++){
    //Bucle for que recorre la matriz bidimensional pasoSimple, que describe ese paso.
    for(int j = 0; j < 4; j++){
      /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
      específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
      constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
      digitalWrite(IN1, pasoDoble[j][0]); //Con esta línea de código se mandan los bits que activan la bobina A.
      digitalWrite(IN2, pasoDoble[j][1]); //Con esta línea de código se mandan los bits que activan la bobina B.
      digitalWrite(IN3, pasoDoble[j][2]); //Con esta línea de código se mandan los bits que activan la bobina C.
      digitalWrite(IN4, pasoDoble[j][3]); //Con esta línea de código se mandan los bits que activan la bobina D.
      /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
      delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
    }
  }

  //Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
  digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
  digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
  digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
  digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
  delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}

```

## Código Arduino - Paso Medio:

```

/*30.3.1.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En este código se le aplica el Paso Medio.*/
//3_Paso medio: Se activa primero una fase y luego dos a la vez en el motor para accionar su rotación,
//dando como resultado la mejor precisión posible de 4,096 pasos y un torque mediocre que soporta
//alrededor de 63 gramos, no se puede saber a ciencia cierta ya que este motor no está hecho para
//este tipo de paso.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 2; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
    INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
  pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
  pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
  pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
  torque, por lo que el motor debe dar un total de 4,096 pasos para dar una vuelta completa,
  esto solo se aplica en el paso medio debido a que:
  - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un

```



```

giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
necesitan 64 pasos, dando un giro de 5.625° por paso al eje original, ya que a fuerza se
deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.
Esta situación aunada a que:
- Un ciclo del paso simple consta de 8 pasos, da como resultado que un giro completo del rotor
se conforme de 64/8 = 8 ciclos.
En conclusión, el motor necesita dar:
#pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*64 = 8*8*64 = 4096 pasos para dar una vuelta.
Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
secuencia del paso simple consta de 8 pasos, que se indican dentro del mismo bucle for.*/
for(int i = 0; i < 512; i++){
/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

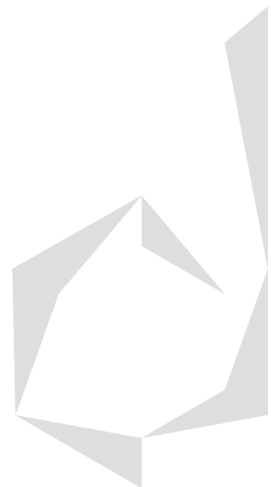
/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina B.
digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
digitalWrite(IN3, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina C.
digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.

/*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
especifico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
digitalWrite(IN1, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, HIGH); //Con esta línea de código se manda un 1 lógico a la bobina D.
/*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
}

//Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.

```



```
digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}
```

## Código Arduino - Paso Medio - Array:

```
/*30.3.2.-Motor a Pasos Unipolar modelo 28BYJ-48 con controlador ULN2003, este cuenta con 4 bobinas
A, B, C y D y una terminal común que se conecta a 5V. En el código anterior la secuencia se realizó
por medio de instrucciones digitalWrite, que mandan un bit, pero en este caso la programación se
realizará por medio de un array bidimensional que guarde los bits que se deben mandar, extraídos de
la tabla que describe el paso simple, incluida en el archivo 12.-Motor a Pasos.*/
//3_Paso medio: Se activa primero una fase y luego dos a la vez en el motor para accionar su rotación,
//dando como resultado la mejor precisión posible de 4,096 pasos y un torque mediocre que soporta
//alrededor de 63 gramos, no se puede saber a ciencia cierta ya que este motor no está hecho para este
//tipo de paso.
int IN1 = 8; //Pin IN1 = Bobina A del motor a pasos conectado al PIN8 digital del Arduino.
int IN2 = 9; //Pin IN2 = Bobina B del motor a pasos conectado al PIN9 digital del Arduino.
int IN3 = 10; //Pin IN2 = Bobina C del motor a pasos conectado al PIN10 digital del Arduino.
int IN4 = 11; //Pin IN2 = Bobina D del motor a pasos conectado al PIN11 digital del Arduino.
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 2; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.
/*Se utiliza la matriz bidimensional pasoMedio[Filas][Columnas] para indicar los bits que se debe
mandar al motor a pasos unipolar para realizar este tipo de paso, recordemos que el tamaño se cuenta
de forma normal desde 1, pero sus coordenadas se indican desde 0 en un array.*/
int pasoMedio[8][4] = {
    {1, 0, 0, 0},
    {1, 1, 0, 0},
    {0, 1, 0, 0},
    {0, 1, 1, 0},
    {0, 0, 1, 0},
    {0, 0, 1, 1},
    {0, 0, 0, 1},
    {1, 0, 0, 1}
};

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
    /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
    de la comunicación serial*/
    /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
    - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
    - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
    INPUT para indicar que el pin es una entrada.
    El número del pin que recibe este método como primer parámetro se puede declarar directamente
    como un número o se puede declarar al inicio del programa como una variable.*/
    pinMode(IN1, OUTPUT); //El pin 8 es una salida digital.
    pinMode(IN2, OUTPUT); //El pin 9 es una salida digital.
    pinMode(IN3, OUTPUT); //El pin 10 es una salida digital.
    pinMode(IN4, OUTPUT); //El pin 11 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
    /*El motor a pasos cuenta con un sistema de engranes que reducen su velocidad y aumentan su
    torque, por lo que el motor debe dar un total de 4,096 pasos para dar una vuelta completa,
    esto solo se aplica en el paso medio debido a que:
    - 64 vueltas completas del rotor original corresponden a 1 vuelta del eje del motor, dando un
    giro de 5.625° por paso al eje final, pero para que el rotor original gire una vuelta se
    necesitan 64 pasos, dando un giro de 5.625° por paso al eje original, ya que a fuerza se
    deben completar 8 ciclos de los pasos para que el rotor original dé 1 vuelta.
    Esta situación aunada a que:
    - Un ciclo del paso simple consta de 8 pasos, da como resultado que un giro completo del rotor
    se conforme de 64/8 = 8 ciclos.
    En conclusión, el motor necesita dar:
    #pasosSecuencia*#Ciclos*#reduccionRotorFinal = 32*64 = 8*8*64 = 4096 pasos para dar una vuelta.
    Esto en código se transfiere a dar #Ciclos*#reduccionRotorFinal = 8*64 = 512 ciclos, porque la
    secuencia del paso simple consta de 8 pasos, que se indican dentro del mismo bucle for. */
    for(int i = 0; i < 512; i++){
        //Bucle for que recorre la matriz bidimensional pasoSimple, que describe ese paso.
        for(int j = 0; j < 8; j++){
            /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
            específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
            constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada. */
            digitalWrite(IN1, pasoMedio[j][0]); //Se mandan los bits que activan la bobina A.
            digitalWrite(IN2, pasoMedio[j][1]); //Se mandan los bits que activan la bobina B.
            digitalWrite(IN3, pasoMedio[j][2]); //Se mandan los bits que activan la bobina C.
            digitalWrite(IN4, pasoMedio[j][3]); //Se mandan los bits que activan la bobina D.
            /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
            delay(demora); //Detiene T = 1/100 = 20 ms el código antes de volver a ejecutarse.
        }
    }

    //Al finalizar de dar una vuelta, el programa se detiene 5 segundos antes de dar otra.
    digitalWrite(IN1, LOW); //Con esta línea de código se manda un 0 lógico a la bobina A.
    digitalWrite(IN2, LOW); //Con esta línea de código se manda un 0 lógico a la bobina B.
    digitalWrite(IN3, LOW); //Con esta línea de código se manda un 0 lógico a la bobina C.
    digitalWrite(IN4, LOW); //Con esta línea de código se manda un 0 lógico a la bobina D.
    delay(5000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}
```



## TLD Motor a Pasos Unipolar: Paso Simple, Doble y Medio en Verilog y VHDL

Se realizarán 3 diseños TLD (Top Level Design) en programas individuales de Verilog y VHDL donde se utilicen las 3 secuencias de pasos respectivamente, las características más importantes de cada paso son:

- **Paso Simple:** Precisión **media** (2,048 pasos para girar el eje del motor 1 vuelta completa) y un **torque bajo** que aguanta un peso máximo de **63 gramos de carga** (0.03006 [N·m]).
- **Paso Doble:** Precisión **media** (2,048 pasos para girar el eje del motor 1 vuelta completa) y alcanza el **torque máximo** del motor, aguantando un peso de **126 gramos de carga** (0.03804 [N·m]).
- **Medio Paso:** **Precisión máxima** (4,096 pasos para girar el eje del motor 1 vuelta completa) y un **torque bajo** que aguanta un peso máximo de **63 gramos de carga** (0.03006 [N·m]).

Paso Simple				Paso Doble				Medio Paso			
1	0	0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	1	0	1	1	0
0	0	0	1	1	0	0	1	0	0	1	1

Las bobinas **A**, **B**, **C** y **D** serán conectadas directamente a los puertos **JA1**, **JA2**, **JA3** y **JA4** de la Nexys 2 y la alimentación del controlador **ULN2003** será conectada a los puertos **+JA6 o +JA12** y **-JA5 o -JA11**.

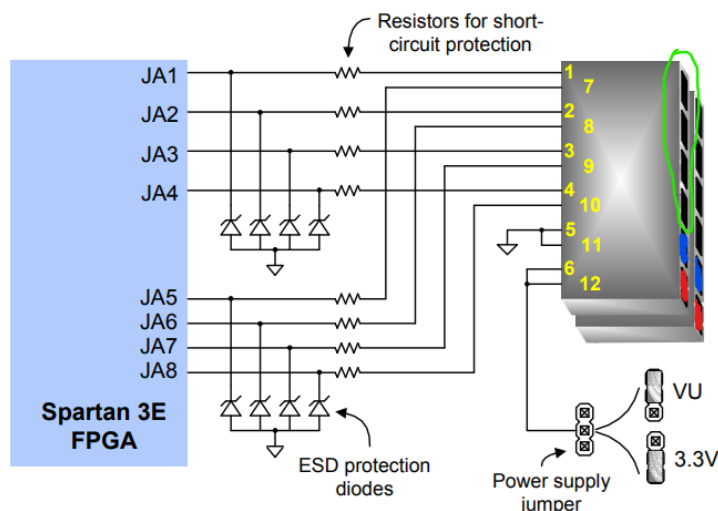


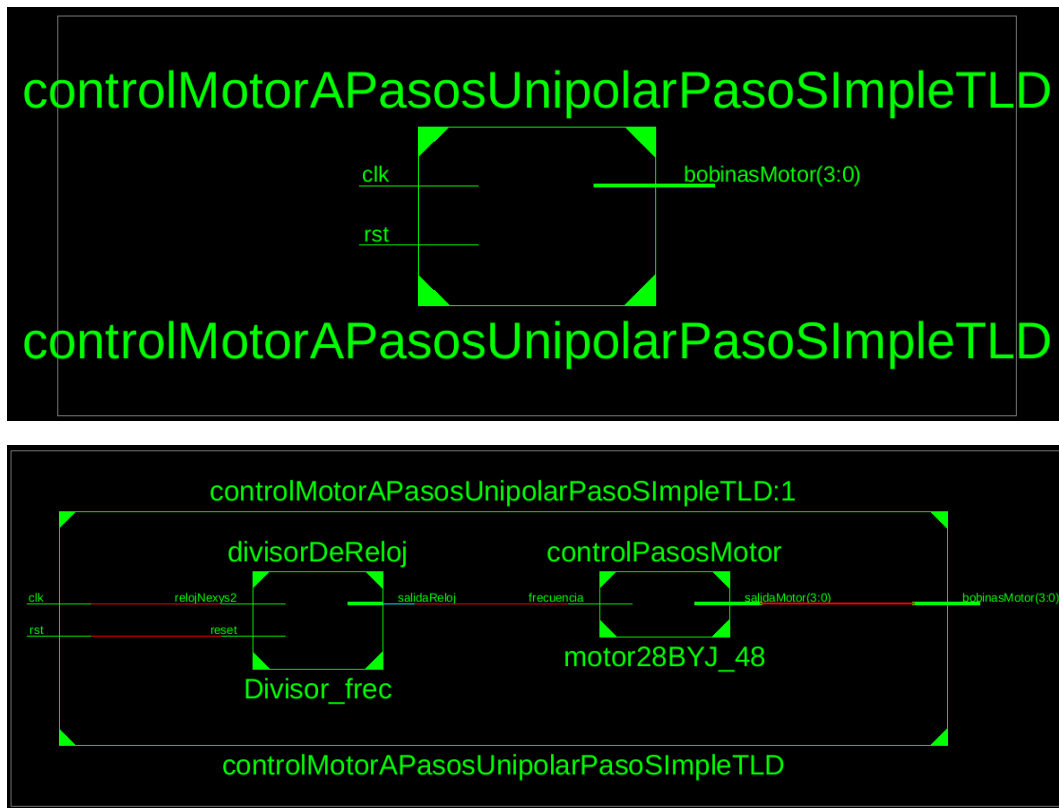
Figure 23: Nexys2 Pmod connector circuits

Table 3: Nexys2 Pmod Connector Pin Assignments							
Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14 <sup>1</sup>
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15 <sup>2</sup>
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15 <sup>3</sup>
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4: P18	JD10: J14 <sup>4</sup>

Notes: <sup>1</sup> shared with LD3 <sup>2</sup> shared with LD3 <sup>3</sup> shared with LD3 <sup>4</sup> shared with LD3

## Código Verilog – Paso Simple:

### Diagrama TLD:



### Divisor de Reloj:

```
//1.-DIVISOR DE RELOJ:
//Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

module divisorDeReloj(
    input relojNexys2, //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input reset,       //Botón de reset.
    output salidaReloj //Reloj que quiero con una frecuencia menor a 50MHz.
);

//REG: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo sirve
//para almacenar y usar valores que sobrevivirán durante la ejecución del código y que además se deben usar dentro de
//un condicional o bucle.
reg [24:0] divisorDeReloj;
//Este reg sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj,
//dependiendo de la coordenada que elijamos, se tomara del vector para asignársela a la salida.

//POSEDGE: La instrucción posedge() solo puede tener una entrada o reg dentro de su paréntesis y a fuerza se debe
//declarar en el paréntesis del always@(), además hace que los condicionales o bucles que estén dentro del always@()
//se ejecuten por si solos cuando ocurra un flanco de subida en la entrada que tiene posedge() dentro de su
//paréntesis, el flanco de subida ocurre cuando la entrada pasa de valer 0 lógico a valer 1 lógico y el hecho de
//que la instrucción posedge() haga que el código se ejecute por si solo, significa que yo directamente no debo
//indicarlo con una operación lógica en el paréntesis de los condicionales o bucles, si lo hago me dará error,
//aunque si quiero que se ejecute una acción en especifico cuando se dé el flanco de subida en solo una de las
//entradas que usan posedge(), debo meter el nombre de esa entrada en el paréntesis del condicional o bucle.
//Si uso posedge() en el paréntesis de un always@(), todas las entradas de ese always@() deben ser activadas igual
//por un posedge().
always@(posedge(relojNexys2), posedge(reset))
begin
    if(reset)
        //En VHDL se puede declarar a un número hexadecimal para evitar poner muchos bits de un numero
        //binario grande, pero en Verilog si hago esto el programa se confunde en el tipo de dato que está
        //recibiendo y como consecuencia obtendremos un error.
        divisorDeReloj = 25'b00000000000000000000000000000000;
    else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado el Reset.
        //No debo poner el caso cuando if(relojNexys2) porque eso ya lo está haciendo la instrucción
        //always@(posedge(relojNexys2), ...) por si sola.
        divisorDeReloj = divisorDeReloj + 1;
end
```

```

//Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
//en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
//En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
assign salidaReloj = divisorDeReloj[16];
//El motor a pasos unipolar 28BYJ-48 con el paso simple puede recibir una frecuencia mínima de 1 Hz y máxima de 400Hz,
//esto nos da la posibilidad de elegir de la coordenada 16 (381.47 Hz) a la coordenada 24 (1.49 Hz) del divisor.
endmodule

```

## Control Motor a Pasos Unipolar:

```

//2.-CONTROL DE MOTOR A PASOS:
//Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
//28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
//para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
//principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
//Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
//A continuación se describen las características de cada una.

//PASO SIMPLE O WAVE DRIVE: En este paso solo se activa una fase a la vez, su secuencia consta de 4 acciones
//por lo cual se necesitan 32 pasos (8 ciclos de 4 secuencias) para que el rotor original de una vuelta
//completa, en total se necesitan 64 vueltas de 32 pasos cada una, osea 2048 pasos para que el eje final de
//una vuelta, como resultado la rotación no es muy suave y se tiene un torque mediocre, que soporta 63 gramos.
module controlPasosMotor(
    input frecuencia,
    input direccionGiro,
    output reg [3:0] salidaMotor, //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg ledDireccion
);

//REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o
//bucle, solo sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta
//de desarrollo y se le asignan valores con el símbolo =
reg [1:0] paso = 2'b00; //Se le da un valor inicial de 0 al vector.
//2'b00 está indicando que el valor es de dos (2) números (b) binarios (b) con valor (00), se usa este
//número porque el paso simple solo tiene 4 acciones, por eso se cuenta del 0 al 3 en binario:
//00, 01, 10 y 11.

//always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se
//deben poner las entradas que usara y además tiene su propio begin y end.
always@(posedge(frecuencia))
//La instrucción posedge() hace que este condicional se ejecute solamente cuando ocurra un flanco de subida
//en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la instrucción
//posedge() hará que el código se ejecute por sí solo, sin que yo directamente tenga que indicarlo con una
//operación lógica.
begin
    if(direccionGiro == 1'b1) begin
        ledDireccion = 1'b1;
        //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
        //paréntesis y asignar una salida correspondiente a cada caso.
        case(paso)
            //Rotación con dirección en el sentido de las manecillas del reloj si el motor a pasos
            //esta boca arriba.
            2'b00 : salidaMotor = 4'b1000;
            2'b01 : salidaMotor = 4'b0100;
            2'b10 : salidaMotor = 4'b0010;
            2'b11 : salidaMotor = 4'b0001;
            //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
            //anteriores uso default.
            default : salidaMotor = 4'b0000;
        endcase
    end else begin
        ledDireccion = 1'b0;
        //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
        //paréntesis y asignar una salida correspondiente a cada caso.
        case(paso)
            //Rotación con dirección contraria al sentido de las manecillas del reloj si el motor a
            //pasos esta boca arriba.
            2'b00 : salidaMotor = 4'b0001;
            2'b01 : salidaMotor = 4'b0010;
            2'b10 : salidaMotor = 4'b0100;
            2'b11 : salidaMotor = 4'b1000;
            //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
            //anteriores uso default.
            default : salidaMotor = 4'b0000;
        endcase
    end
    paso = paso + 2'b01;
end
endmodule

```

## Módulo TLD:

```

//3.-TLD (Top Level Design) controlMotorAPasosUnipolarPasoSimpleTLD:
//Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de paso simple a un motor a pasos
//unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
//se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
//el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
//donde se ejecutan las 4 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
//dirigidas al motor a pasos.
module MotorAPasosUnipolarPasoSimpleTLD(
    //Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto

```

```

//se ve en el documento 12.-Motor a Pasos.
input direcc,
input clk,
input rst,
output [3:0] bobinasMotor,
output ledDirecc
);

//wire: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
//existe durante la ejecución del código y sirve para poder usar algún valor internamente sin tener que estarlo
//actualizando como los datos tipo reg.
wire alambre_int;

//INSTANCIAS:
//Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
//un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
//módulo separadas por comas una de la otra, esto hará que lo que entre o salga del otro módulo, entre, salga o se
//guarde en este. La sintaxis que debemos usar es la siguiente:

//Nombre_Del_Modulo_Que_Queremos_Instanciar      NombreInstancia      (
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Signal_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Signal_En_Este_Modulo)
//);

//INSTANCIA DEL MODULO divisorDeRelej
divisorDeRelej Divisor_frec(
    .relojNexys2(clk),
    //La entrada relojNexys2 del divisorDeRelej se asigna a la entrada clk de este módulo.
    .reset(rst),
    //La entrada reset del divisorDeRelej se asigna a la entrada rst de este módulo.
    .salidaRelej(alambre_int)
    //La salida salidaRelej del divisorDeRelej se asigna a la wire alambre_int de este módulo.
);

//INSTANCIA DEL MODULO encenderApagarLed
controlPasosMotor motor28BYJ_48(
    .direccionGiro(direcc),
    //La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    .frecuencia(alambre_int),
    //La wire alambre_int de este módulo se asigna a la entrada frecuencia del módulo controlPasosMotor.
    .salidaMotor(bobinasMotor),
    //La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
    .ledDireccion(ledDirecc)
    //La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);

endmodule

```

## Código UCF:

```

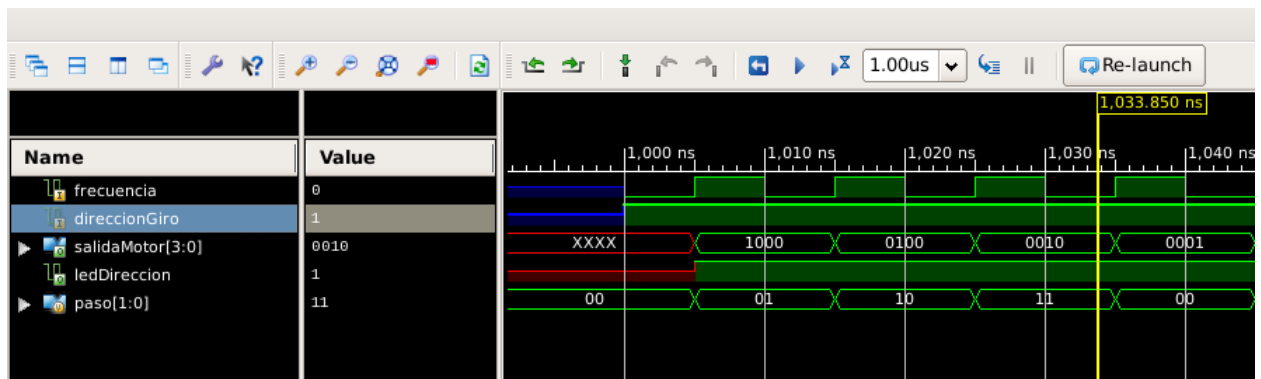
//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18"; //Entrada dirección de giro asignada al switch SW0 en el puerto G18.

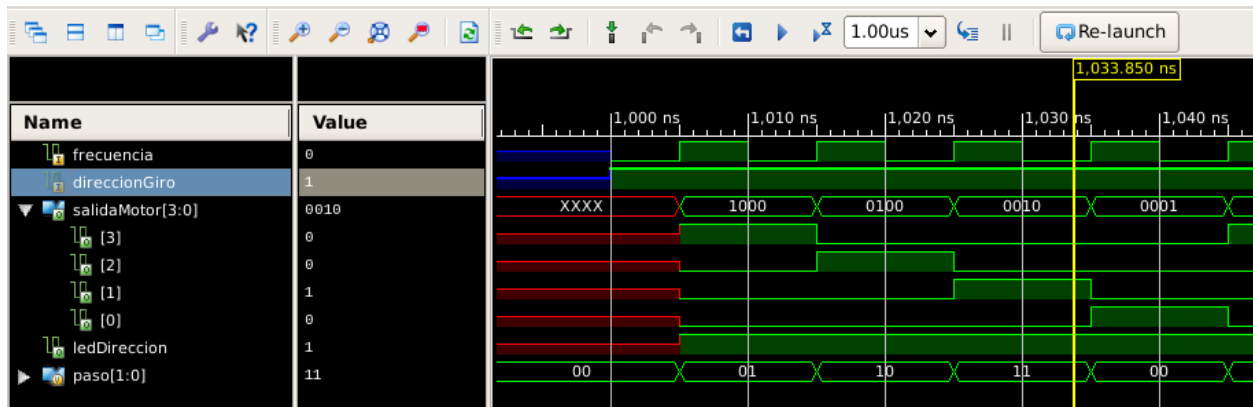
//SALIDAS:
net "bobinasMotor[3]" loc = "L15"; //Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12"; //Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17"; //Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15"; //Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

net "ledDirecc" loc = "J14"; //Salida ledDirecc asignada al led LD0 en el puerto J14.

```

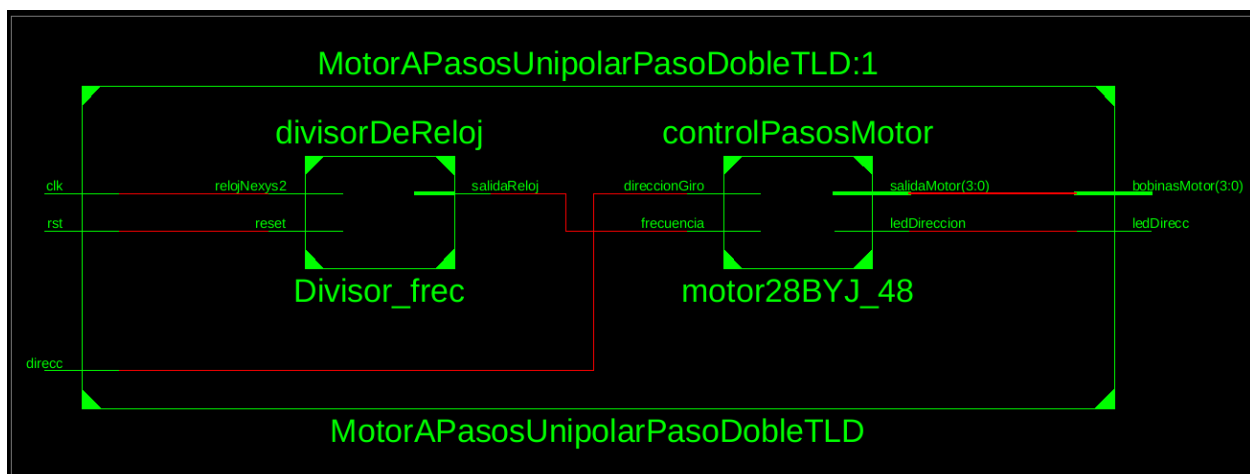
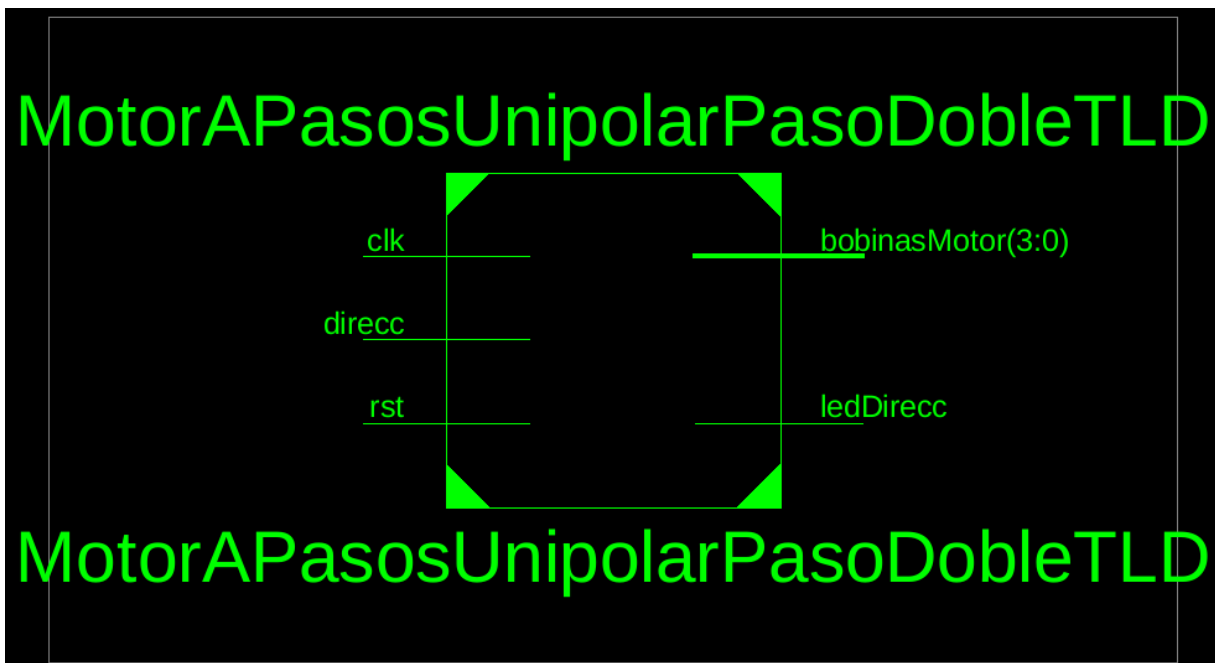
## Simulación Paso Simple:





Código Verilog – Paso Doble:

Diagrama TLD:





## Divisor de Reloj:

```
//1.-DIVISOR DE RELOJ:
//Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

module divisorDeReloj(
    input relojNexys2, //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input reset,        //Botón de reset.
    output salidaReloj //Reloj que quiero con una frecuencia menor a 50MHz.
);

    //REG: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo sirve
    //para almacenar y usar valores que sobrevivirán durante la ejecución del código y que además se deben usar dentro de
    //un condicional o bucle.
    reg [24:0]divisorDeReloj;
    //Este reg sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj,
    //dependiendo de la coordenada que elijamos, se tomara del vector para asignársela a la salida.

    //POSEDGE: La instrucción posedge() solo puede tener una entrada o reg dentro de su paréntesis y a fuerza se debe
    //declarar en el paréntesis del always@(), además hace que los condicionales o bucles que estén dentro del always@()
    //se ejecuten por si solos cuando ocurra un flanco de subida en la entrada que tiene posedge() dentro de su
    //paréntesis, el flanco de subida ocurre cuando la entrada pasa de valer 0 lógico a valer 1 lógico y el hecho de
    //que la instrucción posedge() haga que el código se ejecute por si solo, significa que yo directamente no debo
    //indicarlo con una operación lógica en el paréntesis de los condicionales o bucles, si lo hago me dará error,
    //aunque si quiero que se ejecute una acción en específico cuando se dé el flanco de subida en solo una de las
    //entradas que usan posedge(), debo meter el nombre de esa entrada en el paréntesis del condicional o bucle.
    //Si uso posedge() en el paréntesis de un always@(), todas las entradas de ese always@() deben ser activadas igual
    //por un posedge().
    always@(posedge(relojNexys2), posedge(reset))
    begin
        if(reset)
            //En VHDL se puede declarar a un número hexadecimal para evitar poner muchos bits de un numero
            //binario grande, pero en Verilog si hago esto el programa se confunde en el tipo de dato que está
            //recibiendo y como consecuencia obtendremos un error.
            divisorDeReloj = 25'b00000000000000000000000000000000;
        else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado el Reset.
            //No debo poner el caso cuando if(relojNexys2) porque eso ya lo está haciendo la instrucción
            //always@(posedge(relojNexys2),...) por si sola.
            divisorDeReloj = divisorDeReloj + 1;
        end
        //Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
        //en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
        //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
        assign salidaReloj = divisorDeReloj[16];
        //El motor a pasos unipolar 28BYJ-48 con el paso doble puede recibir una frecuencia mínima de 1 Hz y máxima de 400Hz,
        //esto nos da la posibilidad de elegir de la coordenada 16 (381.47 Hz) a la coordenada 24 (1.49 Hz) del divisor.
    endmodule
```

## Control Motor a Pasos Unipolar:

```
//2.-CONTROL DE MOTOR A PASOS:
//Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
//28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
//para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
//principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
//Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
//A continuación se describen las características de cada una.

//PASO DOBLE O COMPLETO: En este paso se activan dos fases a la vez, su secuencia consta de 4 acciones por lo
//cual se necesitan 32 pasos (8 ciclos de 4 secuencias) para que el rotor original de una vuelta completa, en
//total se necesitan 64 vueltas de 32 pasos cada una, osea 2048 pasos para que el eje final de una vuelta, como
//resultado la rotación no es muy suave, pero se obtiene el máximo torque, que soporta 126 gramos.

module controlPasosMotor(
    input frecuencia,
    input direccionGiro,
    output reg [3:0] salidaMotor, //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg ledDireccion
);

    //REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o
    //bucle, solo sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta
    //de desarrollo y se le asignan valores con el símbolo =
    reg [1:0] paso = 2'b00; //Se le da un valor inicial de 0 al vector.
    //2'b00 está indicando que el valor es de dos (2) números (') binarios (b) con valor (00), se usa este
    //número porque el paso simple solo tiene 4 acciones, por eso se cuenta del 0 al 3 en binario:
    //00, 01, 10 y 11.

    //always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se
    //deben poner las entradas que usara y además tiene su propio begin y end.
    always@(posedge(frecuencia))
    //La instrucción posedge() hace que este condicional se ejecute solamente cuando ocurra un flanco de subida
    //en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la instrucción
    //posedge() hará que el código se ejecute por si solo, sin que yo directamente tenga que indicarlo con una
    //operación lógica.
    begin
        if(direccionGiro == 1'b1) begin
            ledDireccion = 1'b1;
            //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
            //paréntesis y asignar una salida correspondiente a cada caso.
            case(paso)
                //Rotación con dirección en el sentido de las manecillas del reloj si el motor a pasos
                //esta boca arriba.
                2'b00 : salidaMotor = 4'b1100;
            endcase
        end
    end
```

```

                2'b01 : salidaMotor = 4'b0110;
                2'b10 : salidaMotor = 4'b0011;
                2'b11 : salidaMotor = 4'b1001;
                //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
                //anteriores uso default.
                default : salidaMotor = 4'b0000;
            endcase
        end else begin
            ledDireccion = 1'b0;
            //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
            //paréntesis y asignar una salida correspondiente a cada caso.
            case(paso)
                //Rotación con dirección contraria al sentido de las manecillas del reloj si el motor a
                //pasos esta boca arriba.
                2'b00 : salidaMotor = 4'b0011;
                2'b01 : salidaMotor = 4'b0110;
                2'b10 : salidaMotor = 4'b1100;
                2'b11 : salidaMotor = 4'b1001;
                //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
                //anteriores uso default.
                default : salidaMotor = 4'b0000;
            endcase

            end
            paso = paso + 2'b01;
        end
    endmodule

```

## Módulo TLD:

```

//3.-TLD (Top Level Design) controlMotorAPasosUnipolarPasoSimpleTLD:
//Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de paso doble a un motor a pasos
//unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
//se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
//el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
//donde se ejecutan las 4 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
//dirigidas al motor a pasos.

module MotorAPasosUnipolarPasoDobleTLD(
    //Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto
    //se ve en el documento 12.-Motor a Pasos.
    input direcc,
    input clk,
    input rst,
    output [3:0] bobinasMotor,
    output ledDirecc
);

//wire: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
//existe durante la ejecución del código y sirve para poder usar algún valor internamente sin tener que estarlo
//actualizando como los datos tipo reg.
wire alambre_int;

//INSTANCIAS:
//Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
//un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
//módulo separadas por comas una de la otra, esto hará que lo que entre o salga del otro módulo, entre, salga o se
//guarde en este. La sintaxis que debemos usar es la siguiente:

//Nombre_Del_Modulo_Que_Queremos_Instanciar      NombreInstancia      (
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//
//      .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Signal_En_Este_Modulo),
//      .Nombre_De_La_Salida_Del_Modulo_Instanciado(Signal_En_Este_Modulo)
//);

//INSTANCIA DEL MODULO divisorDeReloj
divisorDeReloj Divisor_frec(
    .relojNexys2(clk),
    //La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
    .reset(rst),
    //La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
    .salidaReloj(alambre_int)
    //La salida salidaReloj del divisorDeReloj se asigna a la wire alambre_int de este módulo.
);

//INSTANCIA DEL MODULO encenderApagarLed
controlPasosMotor motor28BYJ_48(
    .direccionGiro(direcc),
    //La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    .frecuencia(alambre_int),
    //La wire alambre_int de este módulo se asigna a la entrada frecuencia del módulo controlPasosMotor.
    .salidaMotor(bobinasMotor),
    //La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
    .ledDireccion(ledDirecc)
    //La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);

endmodule

```

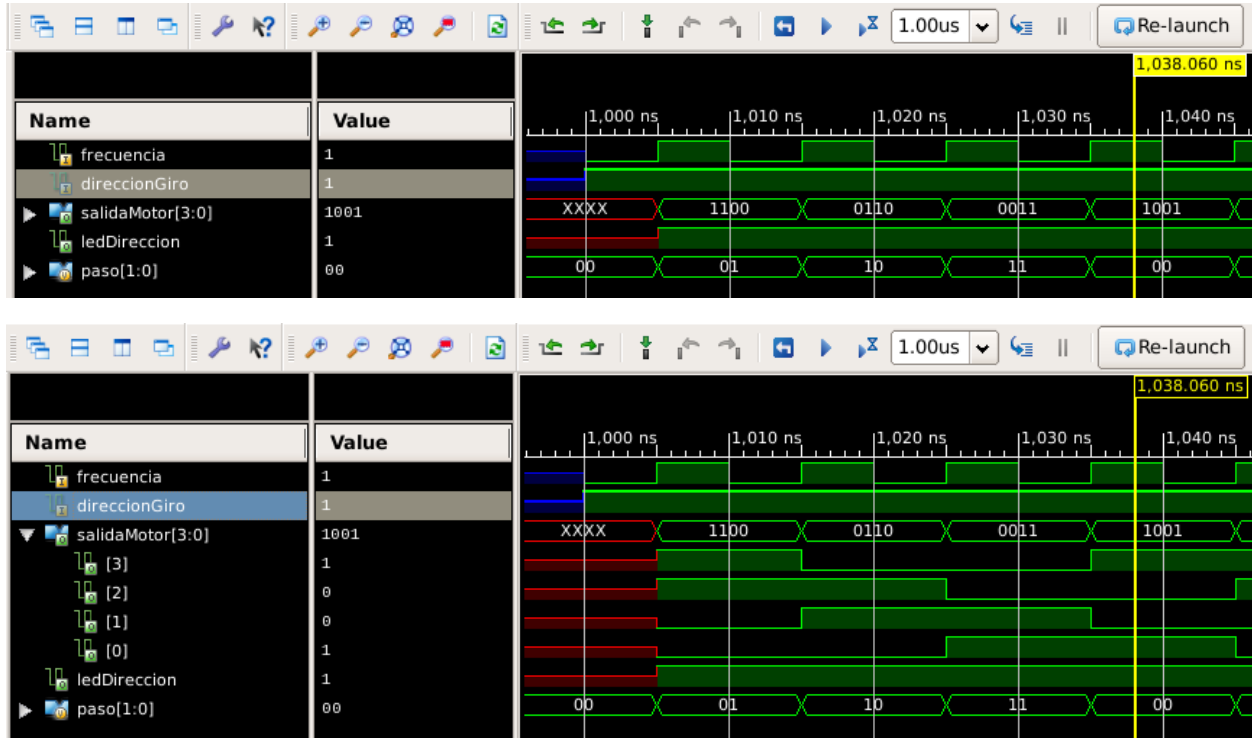
## Código UCF:

```
//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18"; //Entrada direccion de giro asignada al switch SW0 en el puerto G18.

//SALIDAS:
net "bobinasMotor[3]" loc = "L15"; //Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12"; //Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17"; //Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15"; //Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

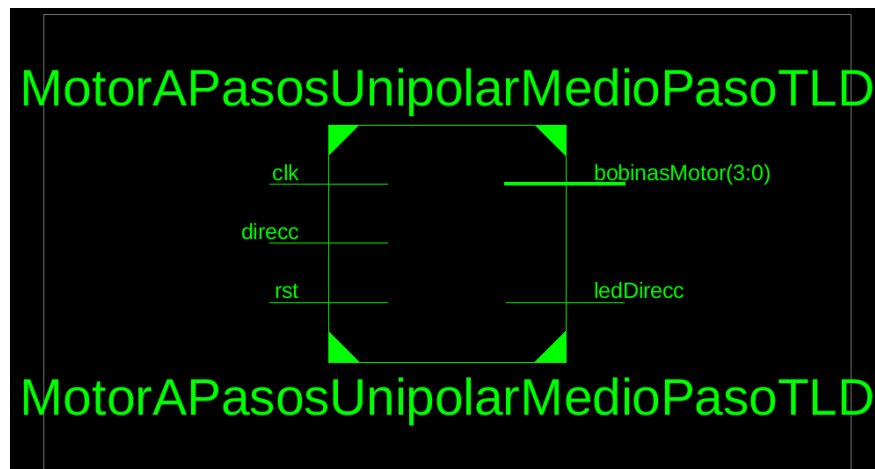
net "ledDirecc" loc = "J14"; //Salida ledDirecc asignada al led LD0 en el puerto J14.
```

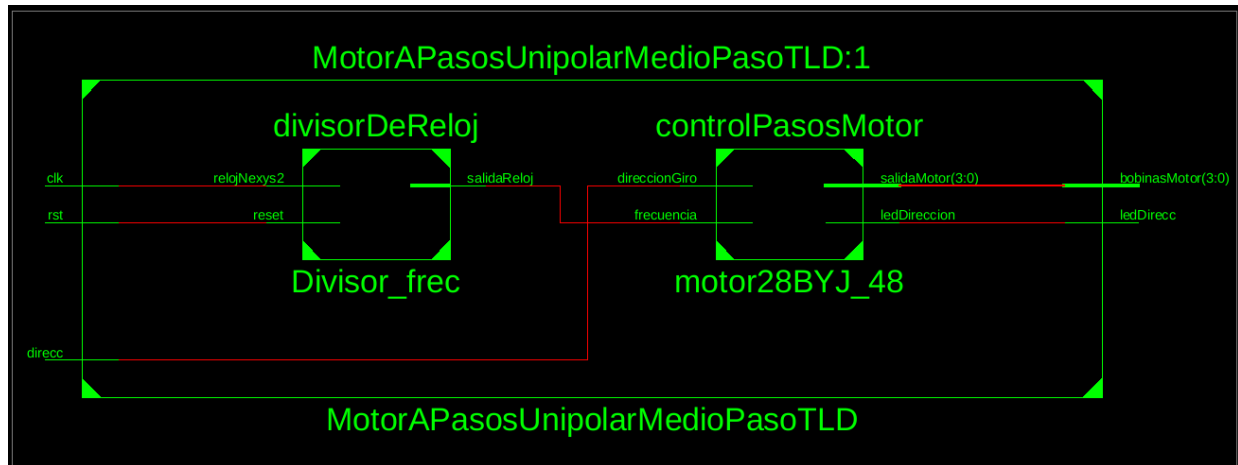
## Simulación Paso Doble:



## Código Verilog – Medio Paso:

### Diagrama TLD:





## Divisor de Reloj:

```
//1.-DIVISOR DE RELOJ:
//Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

module divisorDeReloj(
    input relojNexys2, //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input reset,        //Botón de reset.
    output salidaReloj //Reloj que quiero con una frecuencia menor a 50MHz.
);

    //REG: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo sirve
    //para almacenar y usar valores que sobrevivirán durante la ejecución del código y que además se deben usar dentro de
    //un condicional o bucle.
    reg [24:0]divisorDeReloj;
    //Este reg sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj,
    //dependiendo de la coordenada que elijamos, se tomara del vector para asignársela a la salida.

    //POSEDGE: La instrucción posedge() solo puede tener una entrada o reg dentro de su paréntesis y a fuerza se debe
    //declarar en el paréntesis del always@(), además hace que los condicionales o bucles que estén dentro del always@()
    //se ejecuten por si solos cuando ocurra un flanco de subida en la entrada que tiene posedge() dentro de su
    //paréntesis, el flanco de subida ocurre cuando la entrada pasa de valer 0 lógico a valer 1 lógico y el hecho de
    //que la instrucción posedge() haga que el código se ejecute por si solo, significa que yo directamente no debo
    //indicarlo con una operación lógica en el paréntesis de los condicionales o bucles, si lo hago me dará error,
    //aunque si quiero que se ejecute una acción en específico cuando se dé el flanco de subida en solo una de las
    //entradas que usan posedge(), debo meter el nombre de esa entrada en el paréntesis del condicional o bucle.
    //Si uso posedge() en el paréntesis de un always@(), todas las entradas de ese always@() deben ser activadas igual
    //por un posedge().
    always@(posedge(relojNexys2), posedge(reset))
    begin
        if(reset)
            //En VHDL se puede declarar a un número hexadecimal para evitar poner muchos bits de un numero
            //binario grande, pero en Verilog si hago esto el programa se confunde en el tipo de dato que está
            //recibiendo y como consecuencia obtendremos un error.
            divisorDeReloj = 25'b000000000000000000000000000000;
        else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado el Reset.
            //No debo poner el caso cuando if(relojNexys2) porque eso ya lo está haciendo la instrucción
            //always@(posedge(relojNexys2),...) por si sola.
            divisorDeReloj = divisorDeReloj + 1;
        end

        //Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
        //en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
        //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
        assign salidaReloj = divisorDeReloj[15];
        //El motor a pasos unipolar 28BYJ-48 con el medio paso puede recibir una frecuencia mínima de 1 Hz y máxima de 800Hz,
        //esto nos da la posibilidad de elegir de la coordenada 15 (762.94 Hz) a la coordenada 24 (1.49 Hz) del divisor de
        reloj.
    endmodule
```

## Control Motor a Pasos Unipolar:

```
//2.-CONTROL DE MOTOR A PASOS:
//Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
//28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
//para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
//principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
//Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
//A continuación se describen las características de cada una.

//PASO MEDIO O MEDIO PASO: En este paso primero se activa una fase y luego dos a la vez, por lo cual su
//secuencia consta de 8 acciones y se necesitan 64 pasos (8 ciclos de 8 secuencias) para que el rotor original
//de una vuelta completa, en total se necesitan 64 vueltas de 64 pasos cada una, osea 4096 pasos para que el
//eje final de una vuelta, como resultado la rotación es muy suave, obteniendo la máxima precisión, pero se
//tiene un torque mediocre, que soporta 63 gramos aproximadamente, no se puede saber a ciencia cierta, porque
```

```

//este motor no está hecho para soportar este paso.

module controlPasosMotor(
    input frecuencia,
    input direccionGiro,
    output reg [3:0] salidaMotor, //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg ledDireccion
);

//REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o
//bucle, solo sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta
//de desarrollo y se le asignan valores con el símbolo =
reg [2:0] paso = 3'b000; //Se le da un valor inicial de 0 al vector.
//3'b000 está indicando que el valor es de tres (3) números (') binarios (b) con valor (000), se usa este
//número porque el paso simple tiene 8 acciones, por eso se cuenta del 0 al 7 en binario:
//000, 001, 010, 011, 100, 101, 110, 111.

//always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se
//deben poner las entradas que usara y además tiene su propio begin y end.
always@(posedge(frecuencia))
//La instrucción posedge() hace que este condicional se ejecute solamente cuando ocurra un flanco de subida
//en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la instrucción
//posedge() hará que el código se ejecute por sí solo, sin que yo directamente tenga que indicarlo con una
//operación lógica.
begin
    if(direccionGiro == 1'b1) begin
        ledDireccion = 1'b1;
        //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
        //paréntesis y asignar una salida correspondiente a cada caso.
        case(paso)
            //Rotación con dirección en el sentido de las manecillas del reloj si el motor a pasos
            //esta boca arriba.
            3'b000 : salidaMotor = 4'b1000;
            3'b001 : salidaMotor = 4'b1100;
            3'b010 : salidaMotor = 4'b0100;
            3'b011 : salidaMotor = 4'b0110;
            3'b100 : salidaMotor = 4'b0010;
            3'b101 : salidaMotor = 4'b0011;
            3'b110 : salidaMotor = 4'b0001;
            3'b111 : salidaMotor = 4'b1001;
            //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
            //anteriores uso default.
            default : salidaMotor = 4'b0000;
        endcase
    end else begin
        ledDireccion = 1'b0;
        //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
        //paréntesis y asignar una salida correspondiente a cada caso.
        case(paso)
            //Rotación con dirección contraria al sentido de las manecillas del reloj si el motor a
            //pasos esta boca arriba.
            3'b000 : salidaMotor = 4'b0001;
            3'b001 : salidaMotor = 4'b0011;
            3'b010 : salidaMotor = 4'b0010;
            3'b011 : salidaMotor = 4'b0110;
            3'b100 : salidaMotor = 4'b0100;
            3'b101 : salidaMotor = 4'b1100;
            3'b110 : salidaMotor = 4'b1000;
            3'b111 : salidaMotor = 4'b1001;
            //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones
            //anteriores uso default.
            default : salidaMotor = 4'b0000;
        endcase
    end
    paso = paso + 3'b001;
end
endmodule

```

## Módulo TLD:

```

//3.-TLD (Top Level Design) controlMotorAPasosUnipolarPasoSimpleTLD:
//Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de medio paso a un motor a pasos
//unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
//se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
//el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
//donde se ejecutan las 8 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
//dirigidas al motor a pasos.

module MotorAPasosUnipolarMedioPasoTLD(
    //Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto
    //se ve en el documento 12.-Motor a Pasos.
    input direcc,
    input clk,
    input rst,
    output [3:0] bobinasMotor,
    output ledDirecc
);

//wire: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
//existe durante la ejecución del código y sirve para poder usar algún valor internamente sin tener que estarlo
//actualizando como los datos tipo reg.
wire alambre_int;

```

```

//INSTANCIAS:
//Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
//un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
//modulo separadas por comas una de la otra, esto hará que lo que entre o salga del otro modulo, entre, salga o se
//guarde en este. La sintaxis que debemos usar es la siguiente:
//Nombre_Del_Modulo_Que_Queremos_Instanciar      NombreInstancia
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Signal_En_Este_Modulo),
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Signal_En_Este_Modulo)
//);

//INSTANCIA DEL MODULO divisorDeReloj
divisorDeReloj Divisor_frec(
    .relojNexys2(clk),
    //La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
    .reset(rst),
    //La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
    .salidaReloj(alambre_int)
    //La salida salidaReloj del divisorDeReloj se asigna a la wire alambre_int de este módulo.
);

//INSTANCIA DEL MODULO encenderApagarLed
controlPasosMotor motor28BYJ_48(
    .direccionGiro(direcc),
    //La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    .frecuencia(alambre_int),
    //La wire alambre_int de este módulo se asigna a la entrada frecuencia del módulo controlPasosMotor.
    .salidaMotor(bobinasMotor),
    //La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
    .ledDireccion(ledDirecc)
    //La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);

endmodule

```

## Código UCF:

```

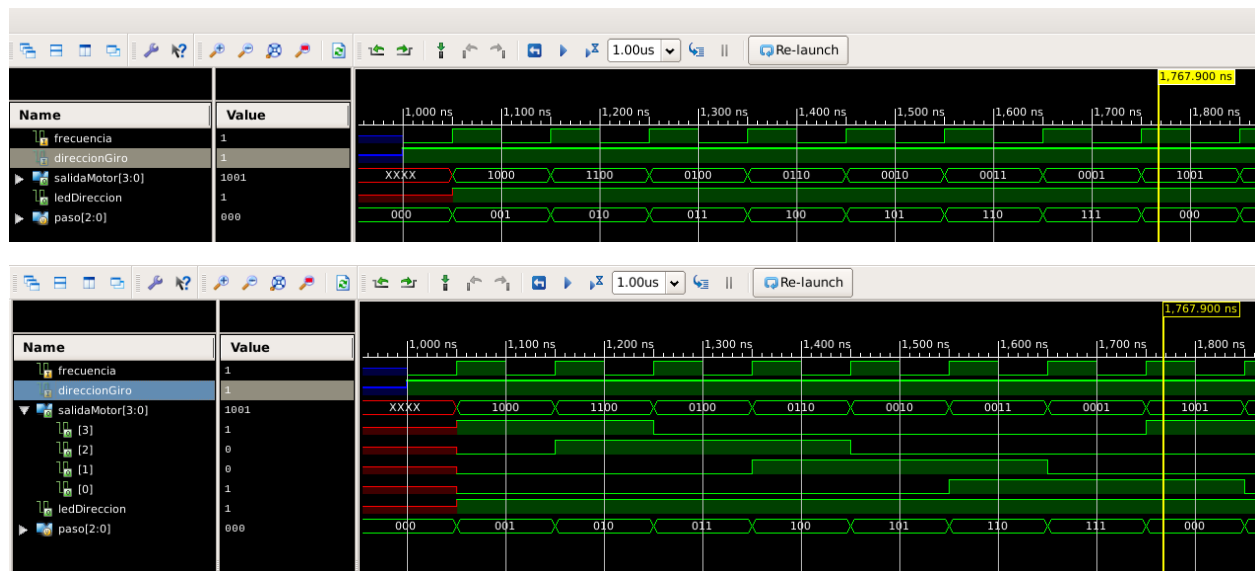
//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18"; //Entrada direccion de giro asignada al switch SW0 en el puerto G18.

//SALIDAS:
net "bobinasMotor[3]" loc = "L15"; //Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12"; //Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17"; //Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15"; //Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

net "ledDirecc" loc = "J14"; //Salida ledDirecc asignada al led LD0 en el puerto J14.

```

## Simulación Medio Paso:



## Código VHDL – Paso Simple:

### Divisor de Reloj:

```
--1.-DIVISOR DE RELOJ:
--Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.

entity divisorDeReloj is
    Port ( relojNexys2 : in  STD_LOGIC;    --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
          reset : in  STD_LOGIC;          --Botón de reset.
          salidaReloj : out STD_LOGIC);    --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí se hará el divisor de reloj haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeReloj is
    --SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
    --existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
    --arquitectura y antes de su begin, se le asignan valores con el símbolo :=
    signal divisorDeReloj : STD_LOGIC_VECTOR (23 downto 0);
    --Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
    --dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.
begin
    process(relojNexys2, reset)
    begin
        if(reset = '1') then--Cuando el Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciara.
            --NUMEROS HEXADECIMALES EN VHDL: 1 dígito hexadecimal equivale a 4 bits en binario, esto nos puede
            --servir para poner un numero binario grande sin tener la necesidad de poner un valor de muchos
            --bits, como cuando debo llenar un vector de puros ceros, declaro un numero hexadecimal poniendo
            --X"numero hexadecimal".
            divisorDeReloj <= X"00000000";
            --Poner X"000000" equivale a poner "000000000000000000000000".
        elsif(rising_edge(relojNexys2)) then
            --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de
            --subida en la señal de reloj clkNexys2 proveniente de la NEXYS 2.
            divisorDeReloj <= divisorDeReloj + 1;--Esto crea al divisor de reloj.
        end if;
    end process;

    --Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
    --en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
    salidaReloj <= divisorDeReloj(16);--En la tabla se ve que la coordenada 16 proporciona una frecuencia de 381.47Hz.
    --El motor a pasos unipolar 28BYJ-48 con el paso simple puede recibir una frecuencia mínima de 1 Hz y máximo de 400Hz,
    --esto nos da la posibilidad de elegir de la coordenada 16 (381.47 Hz) a la coordenada 24 (1.49 Hz) del divisor de
    --reloj.
end frecuenciaNueva;
```

### Control Motor a Pasos Unipolar:

```
--2.-CONTROL DE MOTOR A PASOS:
--Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
--28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
--para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
--principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
--Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
--A continuación, se describen las características de cada una.

--PASO SIMPLE O WAVE DRIVE: En este paso solo se activa una fase a la vez, su secuencia consta de 4 acciones
--por lo cual se necesitan 32 pasos (8 ciclos de 4 secuencias) para que el rotor original de una vuelta
--completa, en total se necesitan 64 vueltas de 32 pasos cada una, osea 2048 pasos para que el eje final de
--una vuelta, como resultado la rotación no es muy suave y se tiene un torque mediocre, que soporta 63 gramos.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: Aquí se declaran las entradas y salidas.
entity controlPasosMotor is
    Port ( frecuencia : in  STD_LOGIC;--Frecuencia de conmutación de las bobinas obtenida del divisor de reloj.
          direccionGiro : in  STD_LOGIC;--Switch que indica la dirección del giro del motor a pasos.
          salidaMotor : out STD_LOGIC_VECTOR (3 downto 0);--Señales dirigidas a las 4 bobinas A, B, C y D.
          ledDireccion : out STD_LOGIC);--Led indicativo de la dirección del giro del motor a pasos.
end controlPasosMotor;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end
--nombreArquitectura;
architecture Behavioral of controlPasosMotor is

    --SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del
    --programa.
    signal paso : STD_LOGIC_VECTOR (1 downto 0) := "00"; --Se le da un valor inicial de 0 al vector.
    --2'b00 está indicando que el valor es de dos (2) números (1) binarios (b) con valor (00), se usa este numero
    --porque el paso simple solo tiene 4 acciones, por eso se cuenta del 0 al 3 en binario:
    --00, 01, 10 y 11.
```

```

begin
--process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
--Las diferentes entradas se separan entre sí por comas.
process(frecuencia, paso) --Se usa para usar condicionales o bucles y tiene su propio begin y end process;
begin
    --La instrucción rising_edge() hace que este condicional se ejecute solamente cuando ocurra un flanco de
    --subida en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la
    --instrucción rising_edge() hará que el código se ejecute por sí solo, sin que yo directamente tenga que
    --indicarlo con una operación lógica.
    if(rising_edge(frecuencia)) then
        if(direccionGiro = '1') then
            ledDireccion <= '1'; --Encender led.
            --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
            case (paso) is
                when "00" => salidaMotor <= "1000"; --Salida de 4 bits dirigida a las bobinas.
                when "01" => salidaMotor <= "0100"; --Salida de 4 bits dirigida a las bobinas.
                when "10" => salidaMotor <= "0010"; --Salida de 4 bits dirigida a las bobinas.
                when "11" => salidaMotor <= "0001"; --Salida de 4 bits dirigida a las bobinas.
                --Para la última condición siempre debo usar la instrucción when others
                --perteneiente al condicional case.
                when others => salidaMotor <= "0000";
            end case; --Al final del end case debo poner punto y coma ;
        else
            ledDireccion <= '0'; --Apagar led.
            --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
            case (paso) is
                when "00" => salidaMotor <= "0001"; --Salida de 4 bits dirigida a las bobinas.
                when "01" => salidaMotor <= "0010"; --Salida de 4 bits dirigida a las bobinas.
                when "10" => salidaMotor <= "0100"; --Salida de 4 bits dirigida a las bobinas.
                when "11" => salidaMotor <= "1000"; --Salida de 4 bits dirigida a las bobinas.
                --Para la última condición siempre debo usar la instrucción when others
                --perteneiente al condicional case.
                when others => salidaMotor <= "0000";
            end case; --Al final del end case debo poner punto y coma ;
        end if;
        --Se usa una signal en vez de hacer el contador directo con la salida contador porque a las salidas
        --solo se les puede asignar un valor, no se les puede leer.
        paso <= paso + "01";
    end if;
end process;
end Behavioral;

```

## Módulo TLD:

```

--3.-TLD (Top Level Design) controlMotorAPasosUnipolarPasoSimpleTLD:
--Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de paso simple a un motor a pasos
--unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
--se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
--el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
--donde se ejecutan las 4 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
--dirigidas al motor a pasos.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.

--Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto se ve
--en el documento 8.2.-Señal de reloj CLK Solamente en VHDL.
entity MotorAPasosUnipolarPasoSimpleTLD is
    Port ( direcc : in STD_LOGIC;
          clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          bobinasMotor : out STD_LOGIC_VECTOR (3 downto 0); --Señales dirigidas a las 4 bobinas A, B, C y D.
          ledDirecc : out STD_LOGIC);
end MotorAPasosUnipolarPasoSimpleTLD;

--Arquitectura: Aquí voy a declarar todas las instancias de mis demás módulos para poderlos usar y sus signal.
architecture Behavioral of MotorAPasosUnipolarPasoSimpleTLD is
    --Signal: Es un tipo de dato que sirve para almacenar un valor que no salga del bloque global TLD.
    signal alambre_int : STD_LOGIC;
    --Las signal se declaran antes del begin de la arquitectura.
begin
    --INSTANCIAS:
    --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
    --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
    --módulo instanciado una entrada, salida o signal de este módulo, separadas por comas una de la otra, esto hará que
    --lo que entre o salga del otro módulo entre, salga o se guarde en este.
    --La sintaxis que debemos usar es la siguiente:

    --NombreInstancia : entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar port map(
    --    Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Salida_En_Este_Modulo,

    --    Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,

    --    Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Signal_En_Este_Modulo
    --);

    --INSTANCIA DEL MODULO divisorDeReloj
    Divisor_frec: entity work.divisorDeReloj port map(
        relojNexys2 => clk,

```



```

--La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
reset => rst,
--La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
salidaReloj => alambre_int
--La salida salidaReloj del divisorDeReloj se asigna a la signal alambre_int de este módulo.
);
--INSTANCIA DEL MODULO controlPasosMotor
LED: entity work.controlPasosMotor port map(
    direccionGiro => direcc,
    --La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    frecuencia => alambre_int,
    --A la entrada frecuencia del controlPasosMotor se le asigna el valor de la signal alambre_int de
    --este módulo.
    salidaMotor => bobinasMotor,
    --La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
    ledDireccion => ledDirecc
    --La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);
end Behavioral;

```

## Código UCF:

```

//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8";//Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13";//Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18";//Entrada direccion de giro asignada al switch SW0 en el puerto G18.

//SALIDAS:
net "bobinasMotor[3]" loc = "L15";//Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12";//Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17";//Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15";//Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

net "ledDirecc" loc = "J14";//Salida ledDirecc asignada al led LD0 en el puerto J14.

```

## Código VHDL – Paso Doble:

### Divisor de Reloj:

```

--1.-DIVISOR DE RELOJ:
--Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.

entity divisorDeReloj is
    Port ( relojNexys2 : in STD_LOGIC; --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
          reset : in STD_LOGIC; --Botón de reset.
          salidaReloj : out STD_LOGIC); --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí se hará el divisor de reloj haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeReloj is
    --SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
    --existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
    --arquitectura y antes de su begin, se le asignan valores con el simbolo :=
    signal divisorDeReloj : STD_LOGIC_VECTOR (24 downto 0);
    --Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
    --dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.
begin
    process(relojNexys2, reset)
    begin
        if(reset = '1') then--Cuando el Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciara.
            --NUMEROS HEXADECIMALES EN VHDL: 1 dígito hexadecimal equivale a 4 bits en binario, esto nos puede
            --servir para poner un numero binario grande sin tener la necesidad de poner un valor de muchos
            --bits, como cuando debo llenar un vector de puros ceros, declaro un numero hexadecimal poniendo
            --X"numero hexadecimal".
            divisorDeReloj <= "00000000000000000000000000000000";
            --Poner X"000000" equivale a poner "00000000000000000000000000000000".
        elsif(rising_edge(relojNexys2)) then
            --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de
            --subida en la señal de reloj clkNexys2 proveniente de la NEXYS 2.
            divisorDeReloj <= divisorDeReloj + 1; --Esto crea al divisor de reloj.
        end if;
    end process;

    --Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
    --en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
    salidaReloj <= divisorDeReloj(16);--En la tabla se ve que la coordenada 16 proporciona una frecuencia de 381.47Hz.
    --El motor a pasos unipolar 28BYJ-48 con el paso doble puede recibir una frecuencia mínima de 1 Hz y máxima de 400Hz,
    --esto nos da la posibilidad de elegir de la coordenada 16 (381.47 Hz) a la coordenada 24 (1.49 Hz) del divisor
    --de reloj.
end frecuenciaNueva;

```

## Control Motor a Pasos Unipolar:

```
--2.-CONTROL DE MOTOR A PASOS:
--Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
--28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
--para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
--principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
--Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
--A continuación, se describen las características de cada una.

--PASO DOBLE O COMPLETO: En este paso se activan dos fases a la vez, su secuencia consta de 4 acciones por lo
--cual se necesitan 32 pasos (8 ciclos de 4 secuencias) para que el rotor original de una vuelta completa, en
--total se necesitan 64 vueltas de 32 pasos cada una, osea 2048 pasos para que el eje final de una vuelta, como
--resultado la rotación no es muy suave, pero se obtiene el máximo torque, que soporta 126 gramos.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: Aquí se declaran las entradas y salidas.
entity controlPasosMotor is
    Port ( frecuencia : in  STD_LOGIC;--Frecuencia de conmutación de las bobinas obtenida del divisor de reloj.
          direccionGiro : in  STD_LOGIC;--Switch que indica la dirección del giro del motor a pasos.
          salidaMotor : out STD_LOGIC_VECTOR (3 downto 0);--Señales dirigidas a las 4 bobinas A, B, C y D.
          ledDireccion : out STD_LOGIC);--Led indicativo de la dirección del giro del motor a pasos.
end controlPasosMotor;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end
--nombreArquitectura;
architecture Behavioral of controlPasosMotor is

--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del
--programa.
signal paso : STD_LOGIC_VECTOR (1 downto 0) := "00"; --Se le da un valor inicial de 0 al vector.
--2'b00 está indicando que el valor es de dos (2) números (') binarios (b) con valor (00), se usa este numero
--porque el paso simple solo tiene 4 acciones, por eso se cuenta del 0 al 3 en binario:
--00, 01, 10 y 11.

begin
--process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
--Las diferentes entradas se separan entre sí por comas.
process(frecuencia, paso) --Se usa para usar condicionales o bucles y tiene su propio begin y end process;
begin
--La instrucción rising_edge() hace que este condicional se ejecute solamente cuando ocurra un flanco de
--subida en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la
--instrucción rising_edge() hará que el código se ejecute por sí solo, sin que yo directamente tenga que
--indicarlo con una operación lógica.
if(rising_edge(frecuencia)) then
    if(direccionGiro = '1')then
        ledDireccion <= '1'; --Encender led.
        --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
        case (paso) is
            when "00" => salidaMotor <= "1100"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "01" => salidaMotor <= "0110"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "10" => salidaMotor <= "0011"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "11" => salidaMotor <= "1001"; --Salida de 4 bits dirigida a las 4 bobinas.
            --Para la última condición siempre debo usar la instrucción when others
            --perteneiente al condicional case.
            when others => salidaMotor <= "0000";
        end case;--Al final del end case debo poner punto y coma ;
    else
        ledDireccion <= '0'; --Apagar led.
        --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
        case (paso) is
            when "00" => salidaMotor <= "0011"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "01" => salidaMotor <= "0110"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "10" => salidaMotor <= "1100"; --Salida de 4 bits dirigida a las 4 bobinas.
            when "11" => salidaMotor <= "1001"; --Salida de 4 bits dirigida a las 4 bobinas.
            --Para la última condición siempre debo usar la instrucción when others
            --perteneiente al condicional case.
            when others => salidaMotor <= "0000";
        end case;--Al final del end case debo poner punto y coma ;
    end if;
    --Se usa una signal en vez de hacer el contador directo con la salida contador porque a las salidas
    --solo se les puede asignar un valor, no se les puede leer.
    paso <= paso + "01";
end process;
end Behavioral;
```

## Módulo TLD:

```
--3.-TLD (Top Level Design) controlMotorAPasosUnipolarPasoSimpleTLD:
--Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de paso simple a un motor a pasos
--unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
--se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
--el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
--donde se ejecutan las 4 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
--dirigidas al motor a pasos.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
--Librerías para poder usar el lenguaje VHDL.

--Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto se ve
--en el documento 8.2.-Señal de reloj CLK Solamente en VHDL.
entity MotorAPasosUnipolarPasoDobleTLD is
  Port ( direcc : in STD_LOGIC;
         clk : in STD_LOGIC;
         rst : in STD_LOGIC;
         bobinasMotor : out STD_LOGIC_VECTOR (3 downto 0); --Señales dirigidas a las 4 bobinas A, B, C y D.
         ledDirecc : out STD_LOGIC);
end MotorAPasosUnipolarPasoDobleTLD;

--Arquitectura: Aquí voy a declarar todas las instancias de mis demás módulos para poderlos usar y sus signal.
architecture Behavioral of MotorAPasosUnipolarPasoDobleTLD is
  --Signal: Es un tipo de dato que sirve para almacenar un valor que no salga del bloque global TLD.
  signal alambre_int : STD_LOGIC;
  --Las signal se declaran antes del begin de la arquitectura.
begin
  --INSTANCIAS:
  --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
  --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
  --módulo instanciado una entrada, salida o signal de este módulo, separadas por comas una de la otra, esto hará que
  --lo que entre o salga del otro módulo entre, salga o se guarde en este.
  --La sintaxis que debemos usar es la siguiente:

  --NombreInstancia      :      entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar      port map(
  --      Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
  --      Salida_Del_Modulo_Instanciado  => Salida_En_Este_Modulo,

  --      Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
  --      Salida_Del_Modulo_Instanciado  => Entrada_En_Este_Modulo,

  --      Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
  --      Salida_Del_Modulo_Instanciado  => Signal_En_Este_Modulo
  --);

  --INSTANCIA DEL MODULO divisorDeReloj
  Divisor_frec: entity work.divisorDeReloj port map(
      relojNexys2 => clk,
      --La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
      reset => rst,
      --La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
      salidaReloj => alambre_int
      --La salida salidaReloj del divisorDeReloj se asigna a la signal alambre_int de este módulo.
  );

  --INSTANCIA DEL MODULO controlPasosMotor

  motor28BYJ_48: entity work.controlPasosMotor port map(
      direccionGiro => direcc,
      --La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
      frecuencia => alambre_int,
      --A la entrada frecuencia del controlPasosMotor se le asigna el valor de la signal alambre_int de
      --este módulo.
      salidaMotor => bobinasMotor,
      --La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
      ledDireccion => ledDirecc
      --La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
  );
end Behavioral;
```

## Código UCF:

```
//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoDobleTLD
//ENTRADAS:
net "clk" loc = "B8";//Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13";//Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18";//Entrada dirección de giro asignada al switch SW0 en el puerto G18.

//SALIDAS:
net "bobinasMotor[3]" loc = "L15";//Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12";//Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17";//Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15";//Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

net "ledDirecc" loc = "J14";//Salida ledDirecc asignada al led LD0 en el puerto J14.
```

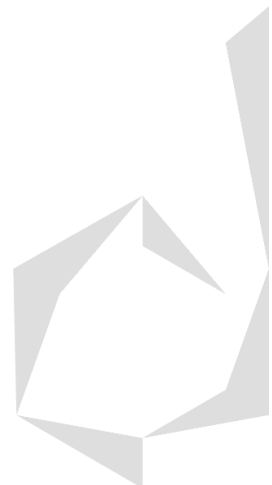
## Código VHDL – Medio Paso:

### Divisor de Reloj:

```
--1.-DIVISOR DE RELOJ:
--Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.

entity divisorDeReloj is
```



```

Port ( relojNexys2 : in STD_LOGIC; --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
      reset : in STD_LOGIC; --Botón de reset.
      salidaReloj : out STD_LOGIC); --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí se hará el divisor de reloj haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeReloj is
--SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
--existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
--arquitectura y antes de su begin, se le asignan valores con el símbolo :=
signal divisorDeReloj : STD_LOGIC_VECTOR (24 downto 0);
--Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
--dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.
begin
    process(relojNexys2, reset)
    begin
        if(reset = '1') then--Cuando el botón Reset sea presionado valdrá 1 lógico y el divisor se reiniciara.
            --NUMEROS HEXADECIMALES EN VHDL: 1 dígito hexadecimal equivale a 4 bits en binario, esto nos puede
            --servir para poner un número binario grande sin tener la necesidad de poner un valor de muchos
            --bits, como cuando debo llenar un vector de puros ceros, declaro un número hexadecimal poniendo
            --X"número hexadecimal".
            divisorDeReloj <= "00000000000000000000000000000000";
            --Poner X"000000" equivale a poner "00000000000000000000000000000000".
        elsif(rising_edge(relojNexys2)) then
            --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de
            --subida en la señal de reloj clkNexys2 proveniente de la NEXYS 2.
            divisorDeReloj <= divisorDeReloj + 1;--Esto crea al divisor de reloj.
        end if;
    end process;

    --Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
    --en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
    salidaReloj <= divisorDeReloj(15);--En la tabla se ve que la coordenada 16 proporciona una frecuencia de 381.47Hz.
    --El motor a pasos unipolar 28BYJ-48 con el medio paso puede recibir una frecuencia mínima de 1 Hz y máxima de 800Hz,
    --esto nos da la posibilidad de elegir de la coordenada 15 (762.94 Hz) a la coordenada 24 (1.49 Hz) del divisor de
    --reloj.
end frecuenciaNueva;

```

## Control Motor a Pasos Unipolar:

```

--2.-CONTROL DE MOTOR A PASOS:
--Modulo que usa el divisor de reloj para prender y apagar las 4 bobinas A, B, C y D del motor a pasos unipolar
--28BYJ-48, este motor cuenta con un sistema de engranes que tiene una reducción de 1/64, esto significa que
--para que el eje final de una vuelta, el eje original debe dar 64 vueltas, aunado a esto para que el rotor
--principal de una vuelta, se deben cumplir 8 ciclos de la secuencia de pasos que se esté utilizando.
--Se cuenta con 3 secuencias de pasos principales: Paso simple, paso doble y paso medio.
--A continuación, se describen las características de cada una.

--PASO MEDIO O MEDIO PASO: En este paso primero se activa una fase y luego dos a la vez, por lo cual su
--secuencia consta de 8 acciones y se necesitan 64 pasos (8 ciclos de 8 secuencias) para que el rotor original
--de una vuelta completa, en total se necesitan 64 vueltas de 64 pasos cada una, osea 4096 pasos para que el
--eje final de una vuelta, como resultado la rotación es muy suave, obteniendo la máxima precisión, pero se
--tiene un torque mediocre, que soporta 63 gramos aproximadamente, no se puede saber a ciencia cierta, porque
--este motor no está hecho para soportar este paso.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: Aquí se declaran las entradas y salidas.
entity controlPasosMotor is
Port ( frecuencia : in STD_LOGIC;--Frecuencia de conmutación de las bobinas obtenida del divisor de reloj.
      direccionGiro : in STD_LOGIC;--Switch que indica la dirección del giro del motor a pasos.
      salidaMotor : out STD_LOGIC_VECTOR (3 downto 0);--Señales dirigidas a las 4 bobinas A, B, C y D.
      ledDireccion : out STD_LOGIC);--Led indicativo de la dirección del giro del motor a pasos.
end controlPasosMotor;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end
--nombreArquitectura;
architecture Behavioral of controlPasosMotor is
--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del
--programa.
signal paso : STD_LOGIC_VECTOR (2 downto 0) := "000"; --Se le da un valor inicial de 0 al vector.
--3'b000 está indicando que el valor es de tres (3) números (') binarios (b) con valor (000), se usa este
--número porque el paso simple tiene 8 acciones, por eso se cuenta del 0 al 7 en binario:
--000, 001, 010, 011, 100, 101, 110, 111.
begin
    --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
    --Las diferentes entradas se separan entre sí por comas.
    process(frecuencia, paso) --Se usa para usar condicionales o bucles y tiene su propio begin y end process;
    begin
        --La instrucción rising_edge() hace que este condicional se ejecute solamente cuando ocurra un flanco de
        --subida en la entrada frecuencia, osea cuando pase de valer 0 lógico a valer 1 lógico, además la
        --instrucción rising_edge() hará que el código se ejecute por sí solo, sin que yo directamente tenga que
        --indicarlo con una operación lógica.
        if(rising_edge(frecuencia)) then
            if(direccionGiro = '1') then
                ledDireccion <= '1'; --Encender led.
                --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
                case (paso) is
                    when "000" => salidaMotor <= "1000"; --Salida de 4 bits dirigida a las bobinas.

```

```

        when "001" => salidaMotor <= "1100"; --Salida de 4 bits dirigida a las bobinas.
        when "010" => salidaMotor <= "0100"; --Salida de 4 bits dirigida a las bobinas.
        when "011" => salidaMotor <= "0110"; --Salida de 4 bits dirigida a las bobinas.
        when "100" => salidaMotor <= "0010"; --Salida de 4 bits dirigida a las bobinas.
        when "101" => salidaMotor <= "0011"; --Salida de 4 bits dirigida a las bobinas.
        when "110" => salidaMotor <= "0001"; --Salida de 4 bits dirigida a las bobinas.
        when "111" => salidaMotor <= "1001"; --Salida de 4 bits dirigida a las bobinas.
        --Para la última condición siempre debo usar la instrucción when others
        --perteneciente al condicional case.
        when others => salidaMotor <= "0000";
    end case; --Al final del end case debo poner punto y coma ;

else
    ledDireccion <= '0'; --Apagar led.
    --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable.
    case (paso) is
        when "000" => salidaMotor <= "0001"; --Salida de 4 bits dirigida a las bobinas.
        when "001" => salidaMotor <= "0011"; --Salida de 4 bits dirigida a las bobinas.
        when "010" => salidaMotor <= "0010"; --Salida de 4 bits dirigida a las bobinas.
        when "011" => salidaMotor <= "0110"; --Salida de 4 bits dirigida a las bobinas.
        when "100" => salidaMotor <= "0100"; --Salida de 4 bits dirigida a las bobinas.
        when "101" => salidaMotor <= "1100"; --Salida de 4 bits dirigida a las bobinas.
        when "110" => salidaMotor <= "1000"; --Salida de 4 bits dirigida a las bobinas.
        when "111" => salidaMotor <= "1001"; --Salida de 4 bits dirigida a las bobinas.
        --Para la última condición siempre debo usar la instrucción when others
        --perteneciente al condicional case.
        when others => salidaMotor <= "0000";
    end case; --Al final del end case debo poner punto y coma ;

end if;
--Se usa una signal en vez de hacer el contador directo con la salida contador porque a las salidas
--solo se les puede asignar un valor, no se les puede leer.
paso <= paso + "001";

end if;
end process;
end Behavioral;

```

## Módulo TLD:

```

--3.-TLD (Top Level Design) MotorAPasosUnipolarMedioPasoTLD:
--Este código sirve para unir los 2 módulos anteriores y poder aplicar la configuración de paso simple a un motor a pasos
--unipolar modelo 28BYJ-48, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
--se encienden y apagan las 4 bobinas A, B, C y D del motor, esto indicará su velocidad, que estará siempre limitada por
--el sistema de engranajes reductor que tiene en su salida, finalmente en el módulo controlMotor se inicia un contador en
--donde se ejecutan las 4 acciones de la secuencia de paso simple y se asigna las señales pertinentes a las 4 salidas
--dirigidas al motor a pasos.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.

--Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto se ve
--en el documento 8.2.-Señal de reloj CLK Solamente en VHDL.
entity MotorAPasosUnipolarMedioPasoTLD is
    Port ( direcc : in STD_LOGIC;
          clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          bobinasMotor : out STD_LOGIC_VECTOR (3 downto 0); --Señales dirigidas a las 4 bobinas A, B, C y D.
          ledDirecc : out STD_LOGIC);
end MotorAPasosUnipolarMedioPasoTLD;

--Arquitectura: Aquí voy a declarar todas las instancias de mis demás módulos para poderlos usar y sus signal.
architecture Behavioral of MotorAPasosUnipolarMedioPasoTLD is
    --Signal: Es un tipo de dato que sirve para almacenar un valor que no salga del bloque global TLD.
    signal alambre_int : STD_LOGIC;
    --Las signal se declaran antes del begin de la arquitectura.
begin
    --INSTANCIAS:
    --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
    --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
    --módulo instanciado una entrada, salida o signal de este módulo, separadas por comas una de la otra, esto hará que
    --lo que entre o salga del otro módulo entre, salga o se guarde en este.
    --La sintaxis que debemos usar es la siguiente:

    --NombreInstancia : entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar port map(
    --    Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Salida_En_Este_Modulo,

    --    Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,

    --    Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
    --    Salida_Del_Modulo_Instanciado => Signal_En_Este_Modulo
    --);

    --INSTANCIA DEL MODULO divisorDeReloj
    Divisor_frec: entity work.divisorDeReloj port map(
        relojNexys2 => clk,
        --La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
        reset => rst,
        --La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
        salidaReloj => alambre_int
        --La salida salidaReloj del divisorDeReloj se asigna a la signal alambre_int de este módulo.
    );

    --INSTANCIA DEL MODULO controlPasosMotor

```

```

motor28BYJ_48: entity work.controlPasosMotor port map(
    direccionGiro => direcc,
    --La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    frecuencia => alambre_int,
    --A la entrada frecuencia del controlPasosMotor se le asigna el valor de la signal alambre_int de
    --este módulo.
    salidaMotor => bobinasMotor,
    --La salida salidaMotor del controlPasosMotor se asigna a la salida bobinasMotor de este módulo.
    ledDireccion => ledDirecc
    --La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);
end Behavioral;

```

## Código UCF:

```

//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoDobleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "G18"; //Entrada dirección de giro asignada al switch SW0 en el puerto G18.

//SALIDAS:
net "bobinasMotor[3]" loc = "L15"; //Bobina A conectada al puerto JA1 perteneciente a los puertos JA.
net "bobinasMotor[2]" loc = "K12"; //Bobina B conectada al puerto JA2 perteneciente a los puertos JA.
net "bobinasMotor[1]" loc = "L17"; //Bobina C conectada al puerto JA3 perteneciente a los puertos JA.
net "bobinasMotor[0]" loc = "M15"; //Bobina D conectada al puerto JA4 perteneciente a los puertos JA.

net "ledDirecc" loc = "J14"; //Salida ledDirecc asignada al led LD0 en el puerto J14.

```

Al realizar la comparación de los programas de control de un motor a pasos unipolar realizados con Arduino y VHDL o Verilog, podremos ver que su mayor diferencia es la frecuencia de operación alcanzada:

- Paso Simple y Doble:
  - **Arduino:** Frecuencias de 2 a 500 Hz.
  - **Verilog y VHDL:** Frecuencias de 1 a 400 Hz (por la tabla del divisor de reloj).
- Medio Paso:
  - **Arduino:** Frecuencias de 2 a 500 Hz.
  - **Verilog y VHDL:** Frecuencias de 1 a 800 Hz (por la tabla del divisor de reloj).

En conclusión, la variación de frecuencias es mínima, aunque si se puede llegar a observar que se realiza de mejor manera el manejo de los tiempos de encendido y apagado de las bobinas con la frecuencia entregada por el FPGA, es más precisa que el Arduino.

## Motor a Pasos Bipolar NEMA 17 y su Controlador A4988

Los motores a pasos **bipolares** más comunes son de tipo **NEMA 17**, este no es un modelo específico de un motor paso a paso, sino que corresponde a las siglas de un estándar industrial que establece ciertas cuestiones mecánicas, para que de esta forma sean compatibles con motores de otros fabricantes:

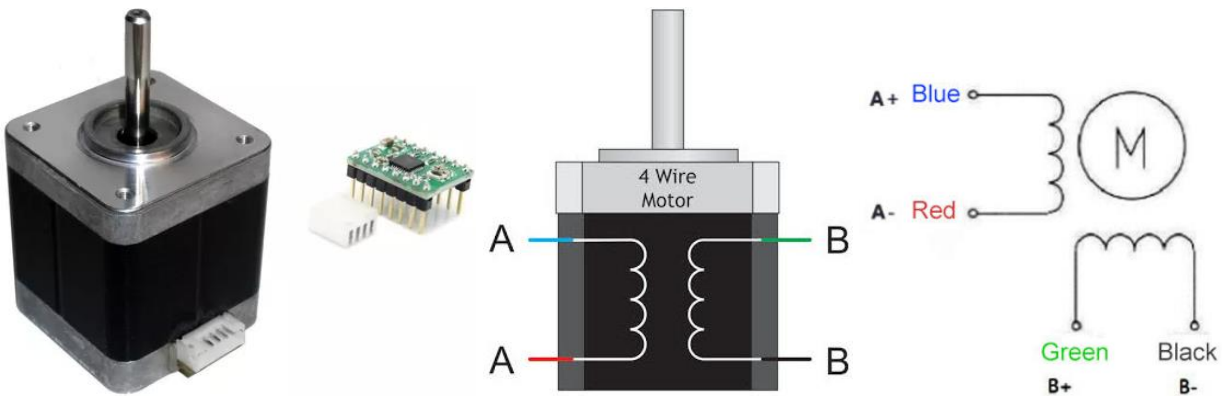
- **N.E.M.A. = National Electrical Manufacturers Association.**
- **17 = 1.7 pulgadas de longitud del motor ≈ 40 mm.**
  - Es importante tener en cuenta que, exceptuando al tamaño físico, las especificaciones como el par, la corriente nominal y la precisión pueden variar entre modelos de motores.

La gran ventaja de los motores bipolares es el mayor torque que entregan en comparación con los motores unipolares, estos cuentan con solamente 2 bobinas o fases A y B que no tienen un punto en común, por lo que el motor posee 4 cables (dos por cada bobina), manejadas por medio del controlador **A4988** o **Pololu** que es una **configuración de puente H** cuya función es conmutar la dirección de la corriente en las bobinas (polaridad de sus electroimanes). Las características del motor y controlador son:



### Motor a pasos NEMA 17:

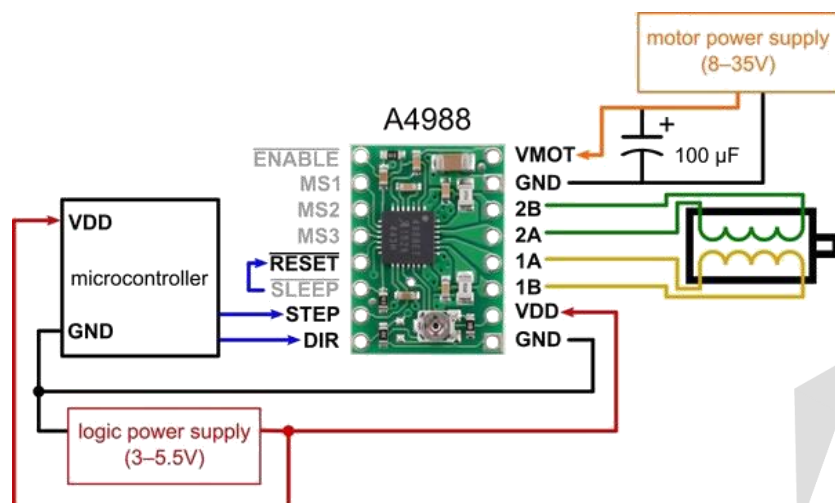
- El motor tipo NEMA 17 recibe de 3 a 35 V de alimentación que no se proporcionan de forma directa, sino a través de su **controlador A4988**, la tensión de entrada depende enteramente del modelo, a continuación, se muestra la tensión de alimentación de 2 motores bipolares distintos:
  - Modelo 17hs4401 de tipo NEMA 17:** Tensión de alimentación de 3 a 12 V.
  - Modelo YISQIAA de tipo NEMA 17:** Tensión de alimentación de 12 a 36 V.
- Cuenta con 2 fases **A** y **B**, y 4 cables **A+**, **A-**, **B+** y **B-** que tienen una resistencia de aproximadamente 1.65  $\Omega$  por bobina y máximo consumen 1.68 A cada una.
- La velocidad de rotación del motor** se controla por medio de una frecuencia mínima de 1 Hz y máxima de 1000 Hz. *Usar frecuencias menores a 100Hz por mucho tiempo calienta el motor.*
- Dependiendo del modelo exacto del motor bipolar NEMA 17 contará con una alguna de las dos precisiones descritas a continuación, que será afectada también por el tipo de paso elegido:
  - 200 pasos para una vuelta completa:** Dando un giro de 1.8° por paso.
  - 400 pasos para una vuelta completa:** Dando un giro de 0.9° por paso.



### Controlador de Motor A4988 "Pololu":

- El controlador recibe de 3 a 5V de alimentación en sus pines inferiores derechos **VDD** y **GND**.
- El controlador además recibe de 3 a 35V para alimentar al motor a pasos en sus pines superiores derechos **VMOT** y **GND**.
- Cuenta con una serie de pines, los cuales tienen funciones específicas:
  - Pines del lado izquierdo - Control:** Los que sean nombrados con una línea superior se activan con un 0 lógico (nivel bajo).
    - ENABLE:** Este pin sirve para **habilitar** el controlador A4988 con un **0 lógico** o **deshabilitarlo** con un **1 lógico**.
    - MS1, MS2 y MS3:** Con los 3 pines MS (de Micro Stepping) se establece el tipo de paso del motor, para así alcanzar una **mayor precisión o torque**, dependiendo de para qué se esté utilizando el motor.
      - Paso doble o completo:** Precisión media de 200 o 400 pasos, **torque máximo**.
      - Medio Paso (1/2 paso):** **Precisión 2 veces mayor**, torque medio.
      - 1/4 paso, 1/8 paso y 1/16 paso:** **Mientras más se reduzca el paso, mayor es la resolución de la precisión en el movimiento**, pero el torque se reduce proporcionalmente.

- **RESET:** Este pin sirve para **restablecer (reiniciar)** el controlador A4988 a su **estado inicial** con un **0 lógico** o **dejarlo correr en su estado actual** con un **1 lógico**.
- **SLEEP:** Este pin sirve para **habilitar el modo ahorrador de energía** que pone a dormir al **controlador A4988** con un **0 lógico** o **deshabilitar ese modo** con un **1 lógico**.
  - Usualmente los pines RESET y SLEEP se conectan entre sí porque como dentro tienen **resistencias pull-up**, se mantienen con un **1 lógico** para no interrumpir el funcionamiento del controlador Pololu.
- **STEP:** Este pin se utiliza para controlar el movimiento del motor, cada que reciba un pulso, el motor se mueve un paso con la configuración de pasos indicada en los pines de micro stepping **MS1, MS2 y MS3**. La frecuencia de dicho pulso controla la velocidad de rotación del motor a pasos bipolar.
- **DIR:** Con este pin se indica el sentido de giro del motor.
  - **1 lógico:** Giro hacia el sentido de las manecillas del reloj (derecha), si se ve cuando el motor está boca arriba.
  - **0 lógico:** Giro contrario al sentido de las manecillas del reloj (izquierda), si se ve cuando el motor está boca arriba.
- **Pines del lado derecho - Alimentación:**
  - **VMOT y GND:** Pines de alimentación del motor, que reciben de 3 a 35V en sus pines superiores derechos **VMOT** y **GND**.
    - **El motor NEMA puede recibir como alimentación de 3 a 35 V y de 1 a 2A.**
    - Por recomendación del fabricante se coloca un capacitor de 100µF/50V entre las terminales de alimentación del motor para filtrar la tensión de entrada. Su conexión se realiza teniendo muy en cuenta las terminales  $\pm$  del capacitor electrolítico, sino este puede explotar.
  - **2B:** Pin conectado a la terminal **B-** del motor a pasos bipolar.
  - **2A:** Pin conectado a la terminal **B+** del motor a pasos bipolar.
  - **1A:** Pin conectado a la terminal **A+** del motor a pasos bipolar.
  - **1B:** Pin conectado a la terminal **A-** del motor a pasos bipolar.
  - **VDD y GND:** Pines de alimentación del controlador, que reciben de 3 a 5V en sus pines inferiores derechos **VDD** y **GND**.

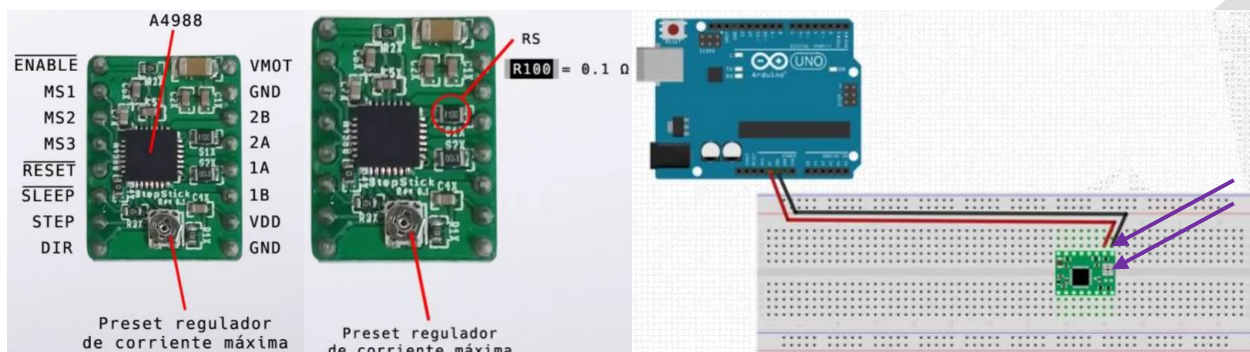


- El controlador viene con un **disipador de calor** que se puede pegar o no sobre el circuito integrado que realiza la conmutación del puente H dependiendo de las siguientes condiciones:
  - **No se coloca el disipador de calor:** Cuando la **corriente máxima** que consumen las bobinas del motor bipolar es de **1A**.
  - **Se coloca el disipador de calor:** Cuando la **corriente** que consumen las fases del motor bipolar va de **1 a 2A**.



- En su parte inferior cuenta con un potenciómetro de seguridad llamado Preset, el cual sirve para establecer la corriente máxima que pueden recibir las bobinas **A** y **B**, si por alguna razón se excede dicha corriente, el módulo interrumpe la alimentación del motor para proteger sus bobinas, evitando que se fundan por exceso de corriente, se debe establecer la tensión del potenciómetro a través de la siguiente ecuación, donde:
  - ***I<sub>max</sub>***: Representa el límite de corriente permitida que puede fluir a través de las bobinas, por buenas prácticas se elige una corriente un poco menor al verdadero límite, en este caso como el límite real es de 1.65 A, el límite en la ecuación será de 1.5 A.
  - ***R<sub>s</sub>***: Resistencia de censado que se encuentra alado del chip A4988, específicamente en su lado superior derecho cuando el preset se vea en su parte inferior. Su valor está indicado sobre la resistencia SMD (Surface Mount Device) de montaje superficial, que es más chiquita y no tiene código de colores para saber su resistencia, solo un número.
    - El valor de las resistencias SMD se divide entre 1000, si dice R100, en realidad es de 0.1 Ω, a veces la resistencia puede indicarse directamente con su valor de 0.1.
  - ***V<sub>ref</sub>***: El potenciómetro preset se mueve hasta que se alcanza la tensión calculada entre el pin **GND** del A4988 y el **metal del tornillo del preset**, para ello el controlador debe ser alimentado con 5V en sus pines inferiores derechos **VDD** y **GND**.

$$V_{ref} = I_{max}(8 * R_s) = 1.5(8 * 0.1) = 1.2V$$



## Pasos del Motor a Pasos Bipolar: Completo, Medio, 1/4, 1/8 y 1/16 de Paso en Arduino.

Existen varias maneras para **controlar** el **motor a pasos bipolar**, se describirán 5 maneras, donde:

- Dependiendo de la **precisión** del movimiento y el **torque** que se quiera obtener se usa un tipo de paso diferente:
  - **Paso Doble**: Precisión **media** de 200 pasos para girar el eje del motor 1 vuelta completa y se alcanza el **torque máximo**, que aguanta un peso de **4 kilogramos de carga** (0.4 [N·m]).
  - **Medio paso, 1/4 paso, 1/8 paso y 1/16 paso**: **Mientras más se reduzca el paso, mayor es la resolución de la precisión en el movimiento**, pero el torque se reduce proporcionalmente.
- **Este motor puede detenerse en un punto deseado y mantener esa posición.**
- Se controla la **velocidad de rotación** del motor en función de la **frecuencia con la que se realizan sus pasos en el pin STEP**. La velocidad del motor bipolar es mucho mayor que la del unipolar.

Tipo de Paso	Pin MS1	Pin MS2	Pin MS3
<b>Paso Completo</b>	0	0	0
<b>Medio paso</b>	1	0	0
<b>1/4 de paso</b>	0	1	0
<b>1/8 de paso</b>	1	1	0
<b>1/16 de paso</b>	1	1	1

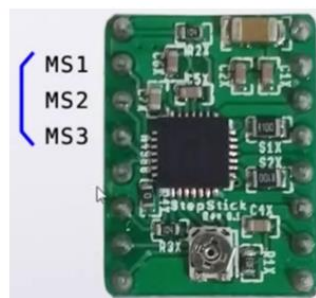
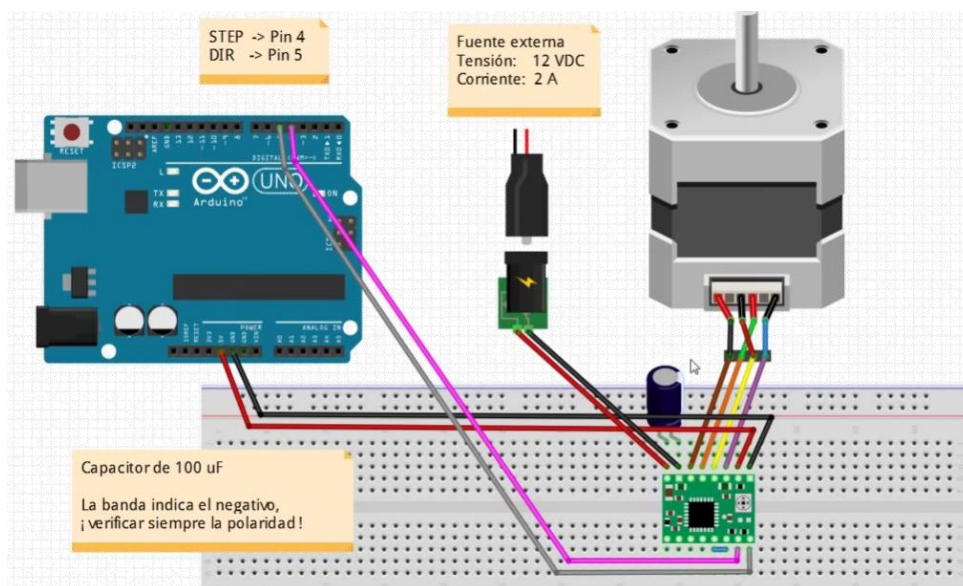


Diagrama de conexión del controlador (Arduino o FPGA), el controlador de motor y el motor a pasos:

Los pines MS1, MS2 y MS3 pueden conectarse o no, si no se conectan a nada, se estará eligiendo el **paso completo**, obteniendo una **precisión media** (giro de 360° al realizar 200 o 400 pasos) y el **torque máximo que aguanta 4 kg**. Además, el Arduino si alcanza a alimentar directamente al controlador A4988 y al motor sin una fuente externa conectada a los pines **VDD**, **VMOT** y **GND**, pero no se debe hacer por mucho tiempo, sino el chip del controlador se empieza a calentar.



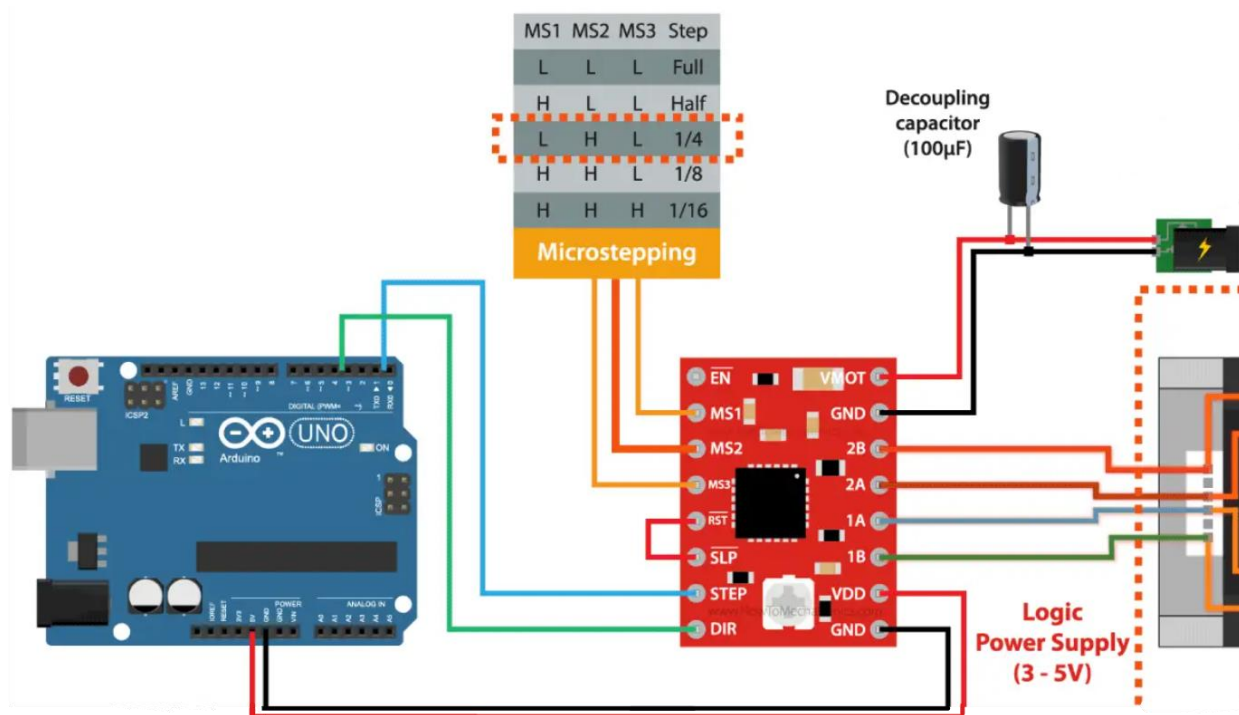


Si se conectan los pines MS1, MS2 y MS3 a algunos puertos digitales del Arduino se puede elegir el paso que se quiere utilizar, es importante remarcar que **mientras más se reduzca el paso, mayor es la precisión, pero menor es el torque de forma proporcional**, siguiendo la fórmula descrita a continuación:

$$\# \text{ Pasos 1 vuelta paso completo} = 200 \text{ o } 400 \left[ \frac{\text{pasos}}{1 \text{ revolucion}} \right]$$

$$\# \text{ Pasos 1 vuelta demás pasos} = \# \text{ Pasos 1 vuelta paso completo} * \frac{1}{\text{división de paso}}$$

$$\# \text{ Pasos 1 vuelta } \frac{1}{4} \text{ de paso} = \# \text{ Pasos 1 vuelta paso completo} * \frac{1}{\text{división de paso}} = 200 * \frac{1}{\frac{1}{4}} = 200 * 4 = 800 \left[ \frac{\text{pasos}}{\text{rev}} \right]$$



En esta configuración se asocian de la siguiente manera los pines del controlador con el Arduino:

- **MS1 → Pin 7, MS2 → Pin 6 y MS3 → Pin 5:** Con estos 3 pines MS (de Micro Stepping) se establece el tipo de paso del motor para lograr una **mayor precisión o torque**, se elige una configuración dependiendo de para qué se esté utilizando el motor.
- **STEP → Pin 4:** Este pin se utiliza para controlar el movimiento del motor, cada que reciba un pulso, el motor se mueve un paso con la configuración de pasos indicada en los pines de micro stepping *MS1*, *MS2* y *MS3*. La frecuencia de dicho pulso controla la velocidad de rotación del motor a pasos bipolar, que es mucho mayor que la velocidad del motor a pasos unipolar.
- **DIR → Pin 3:** Con este pin se indica el sentido de giro del motor, si se le envía un 1 lógico el eje del motor gira hacia un lado y con 0 lógico gira hacia el otro lado.
  - **1 lógico:** Giro hacia el sentido de las manecillas del reloj (derecha), si se ve cuando el motor está boca arriba.
  - **0 lógico:** Giro contrario al sentido de las manecillas del reloj (izquierda), si se ve cuando el motor está boca arriba.

## Código Arduino - Paso Doble o Completo:

```
/*32.1.-Motor a Pasos Bipolar tipo NEMA 17 con controlador A4988, este cuenta con 2 bobinas
A y B, su gran ventaja en comparación con el motor a pasos unipolar es que este aguanta mucho
más carga, soportando hasta 4kg de peso y tiene una mayor velocidad de rotación.
Si no se les indica ningún bit a los pines MS1, MS2 y MS3 el paso elegido es el doble o completo,
que resiste la carga máxima y necesita dar 200 pasos para completar 1 giro completo del eje.*/
/*#define: Instrucción utilizada para crear constantes globales en el programa de Arduino
siguiendo la sintaxis descrita a continuación:
#define nombreConstante valorConstante*/
#define STEP 4
#define DIR 3
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 1; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(STEP, OUTPUT); //La constante STEP asignada al pin digital 4 es una salida.
  pinMode(DIR, OUTPUT); //La constante DIR asignada al pin digital 5 es una salida.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(DIR, HIGH); //DIR = 1 lógico: Giro en sentido de las manecillas del reloj.
  /*El motor a pasos NEMA 17 puede constar de 200 o 400 pasos:
  - Si consta de 200 pasos, tiene una rotación de 1.8° por paso.
  - Si consta de 400 pasos, tiene una rotación de 0.9° por paso.*/
  for(int i = 0; i < 200; i++){ //Giro de 360°, avanzando a 1.8° por paso hacia la derecha.
    digitalWrite(STEP, HIGH);
    /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
    delay(demora);
    digitalWrite(STEP, LOW);
    delay(demora);
  }

  digitalWrite(DIR, LOW); //DIR = 0 lógico: Giro en sentido contrario de las manecillas del reloj.
  for(int i = 0; i < 200; i++){ //Giro de 360°, avanzando a 1.8° por paso hacia la izquierda.
    digitalWrite(STEP, HIGH);
    delay(demora);
    digitalWrite(STEP, LOW);
    delay(demora);
  }

  /*Al finalizar de dar una vuelta de un lado y otra vuelta hacia el otro lado, el programa se detiene
  5 segundos antes de dar otra vez las dos vueltas.*/
  delay(2000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}
```

## Código Arduino – 1/2 Paso, 1/4 de Paso, 1/8 de Paso y 1/16 de Paso:

```
/*32.2.-Motor a Pasos Bipolar tipo NEMA 17 con controlador A4988, este cuenta con 2 bobinas
A y B, su gran ventaja en comparación con el motor a pasos unipolar es que este aguanta mucho
más carga, soportando hasta 4kg de peso y tiene una mayor velocidad de rotación.
En este programa por medio de los pines MS1, MS2 y MS3 se elige el tipo de paso que se quiere
utilizar, pero si no es el paso completo, se reduce la carga que puede resistir el motor 1/2,
1/4, 1/8 y 1/16 veces proporcionalmente.*/
/*#define: Instrucción utilizada para crear constantes globales en el programa de Arduino
siguiendo la sintaxis descrita a continuación:
#define nombreConstante valorConstante*/
#define MS1 7
#define MS2 6
#define MS3 5
#define STEP 4
#define DIR 3
/*El motor a pasos recibe una frecuencia mínima de 2 Hz y máxima de 500 Hz, la variable demora
indicará esa frecuencia, pero en milisegundos del periodo, que es el inverso de la frecuencia:
T = 1/f = 1/2 = 0.5 segundos = 500 milisegundos; El movimiento más lento.
T = 1/f = 1/500 = 0.002 = 2 milisegundos; El movimiento más rápido.*/
int demora = 1; //Cambiando esta frecuencia se cambia la velocidad de rotación del motor.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(STEP, OUTPUT); //La constante STEP asignada al pin digital 4 es una salida.
  pinMode(DIR, OUTPUT); //La constante DIR asignada al pin digital 5 es una salida.
  pinMode(MS1, OUTPUT); //La constante MS1 asignada al pin digital 7 es una salida.
  pinMode(MS2, OUTPUT); //La constante MS2 asignada al pin digital 6 es una salida.
  pinMode(MS3, OUTPUT); //La constante MS3 asignada al pin digital 5 es una salida.
}
```





```

El número del pin que recibe este método como primer parámetro se puede declarar directamente
como un número o se puede declarar al inicio del programa como una variable.*/
pinMode(STEP, OUTPUT); //La constante STEP asignada al pin digital 4 es una salida.
pinMode(DIR, OUTPUT); //La constante DIR asignada al pin digital 5 es una salida.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*Con los 3 pines MS1, MS2 Y MS3 de Micro Stepping se establece el tipo de paso del motor para lograr
  una mayor precisión o torque, se elige una de las siguientes opciones dependiendo de para qué se esté
  utilizando el motor:
      MS1    MS2    MS3
  Paso Completo = 0      0      0
  Paso Medio    = 1      0      0
  1/4 de Paso   = 0      1      0
  1/8 de Paso    = 1      1      0
  1/16 de Paso  = 1      1      1
  Mientras más se reduzca el paso, mayor es la resolución de la precisión en el movimiento, pero el torque
  se reduce proporcionalmente:
      # Pasos 1 vuelta paso completo = 200 [pasos/1 revolucion]
      # Pasos 1 vuelta = # Pasos 1 vuelta paso completo * 1/(división de paso)
      # Pasos 1 vuelta 1/4 de paso = 200 * 1/(1/4) = 200 * 4 = 800 [pasos/rev].*/
  //Paso Completo: Necesita dar 200 pasos para que el eje del motor dé 1 vuelta completa.
  digitalWrite(MS1, LOW); //MS1 = 0.
  digitalWrite(MS2, HIGH); //MS2 = 1.
  digitalWrite(MS3, LOW); //MS3 = 0; 1/4 de paso.

  /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
  específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
  constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada.*/
  digitalWrite(DIR, HIGH); //DIR = 1 lógico: Giro en sentido de las manecillas del reloj.
  /*El motor a pasos NEMA 17 puede constar de 200 o 400 pasos:
  - Si consta de 200 pasos, tiene una rotación de 1.8° por paso.
  - Si consta de 400 pasos, tiene una rotación de 0.9° por paso.*/
  for(int i = 0; i < 800; i++){ //Giro de 360°, avanzando a 1.8° por paso.
    digitalWrite(STEP, HIGH);
    /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
    delay(demora);
    digitalWrite(STEP, LOW);
    delay(demora);
  }
  digitalWrite(DIR, LOW); //DIR = 0 lógico: Giro en sentido contrario de las manecillas del reloj.
  /*El motor a pasos NEMA 17 puede constar de 200 o 400 pasos:
  - Si consta de 200 pasos, tiene una rotación de 1.8° por paso.
  - Si consta de 400 pasos, tiene una rotación de 0.9° por paso.*/
  for(int i = 0; i < 800; i++){ //Giro de 180°, avanzando a 1.8° por paso.
    digitalWrite(STEP, HIGH);
    /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
    delay(demora);
    digitalWrite(STEP, LOW);
    delay(demora);
  }

  /*Al finalizar de dar una vuelta de un lado y otra vuelta hacia el otro lado, el programa se detiene
  5 segundos antes de dar otra vez las dos vueltas.*/
  delay(1000); //Detiene el código 5 segundos antes de dar otra vuelta al motor.
}

```

## TLD Motor a Pasos Bipolar: Paso Completo, Medio, 1/4, 1/8 y 1/16 en Verilog y VHDL

Se realizará 1 diseño TLD (Top Level Design) donde se utilizarán las 5 secuencias de pasos del motor bipolar en programas individuales de Verilog y VHDL, pudiendo controlar cual secuencia se elige por medio de switches en la Nexys 2, las características más importantes de cada paso son:

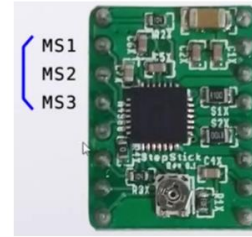
- **Paso Doble:** Precisión **media** (200 pasos para girar el eje del motor 1 vuelta completa) y el **torque máximo**, que aguanta un peso de **4 kilogramos de carga (0.4 [N·m])**.
- **Medio paso, 1/4 paso, 1/8 paso y 1/16 paso:** **Mientras más se reduzca el paso, mayor es la resolución de la precisión en el movimiento**, pero el torque se reduce proporcionalmente.

$$\# \text{ Pasos 1 vuelta paso completo} = 200 \left[ \frac{\text{pasos}}{1 \text{ revolucion}} \right]$$

$$\# \text{ Pasos 1 vuelta demás pasos} = \# \text{ Pasos 1 vuelta paso completo} * \frac{1}{\text{división de paso}}$$

$$\# \text{ Pasos 1 vuelta } \frac{1}{4} \text{ de paso} = \# \text{ Pasos 1 vuelta paso completo} * \frac{1}{\text{división de paso}} = 200 * \frac{1}{\frac{1}{4}} = 200 * 4 = 800 \left[ \frac{\text{pasos}}{\text{rev}} \right]$$

Tipo de Paso	Pin MS1	Pin MS2	Pin MS3
Paso Completo	0	0	0
Medio paso	1	0	0
1/4 de paso	0	1	0
1/8 de paso	1	1	0
1/16 de paso	1	1	1



Los pines **STEP**, **DIR**, **MS1**, **MS2** y **MS3** del módulo **A4988** sirven para controlar el motor bipolar e indicar la configuración de su paso, estos no se pueden conectar directamente a los puertos **JA1**, **JA2**, **JA3**, **JA4**, **JA7**, **JA8**, **JA9** o **JA10** de la Nexys 2, los pines de alimentación **VDD**, **VMOT** y **GND** tampoco pueden ser conectados a los puertos **+JA6** o **+JA12** y **-JA5** o **-JA11**, esto se debe a que como la FPGA entrega una alimentación de 3.3 V en sus pines, no es suficiente para activar el driver, se necesita usar un **convertidor CC boost intermedio** para que eleve la tensión de alimentación y se pueda controlar el motor bipolar, utilizar una fuente externa o separar el circuito en una **etapa de potencia** y otra **de control**.

- La corriente y tensión de alimentación que entrega la tarjeta Nexys 2 (FPGA) es de: **3.3V** y **100 mA**.
  - La corriente que entregan sus **pin**es es de **8mA a 10mA**.
- La alimentación que el Arduino UNO entrega cuando está conectado por USB es de: **5V** y **200 mA**.
  - La corriente que entregan sus **pin**es es de **20mA**.

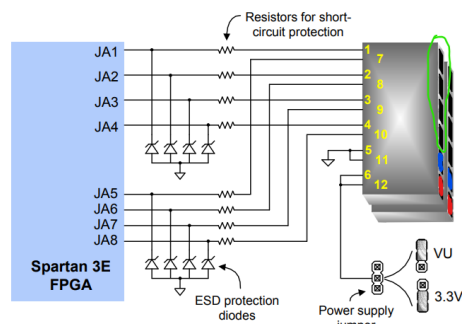
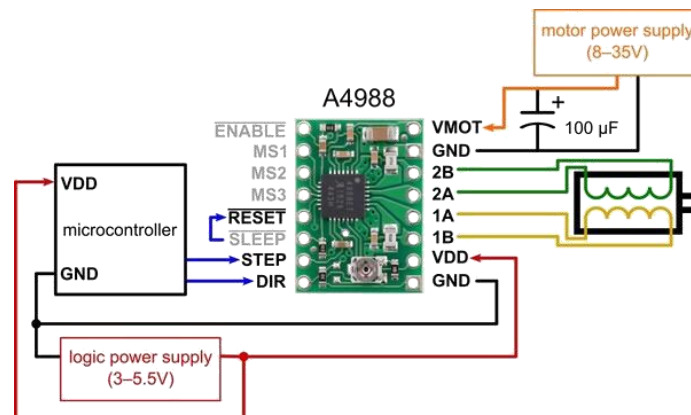


Figure 23: Nexys2 Pmod connector circuits

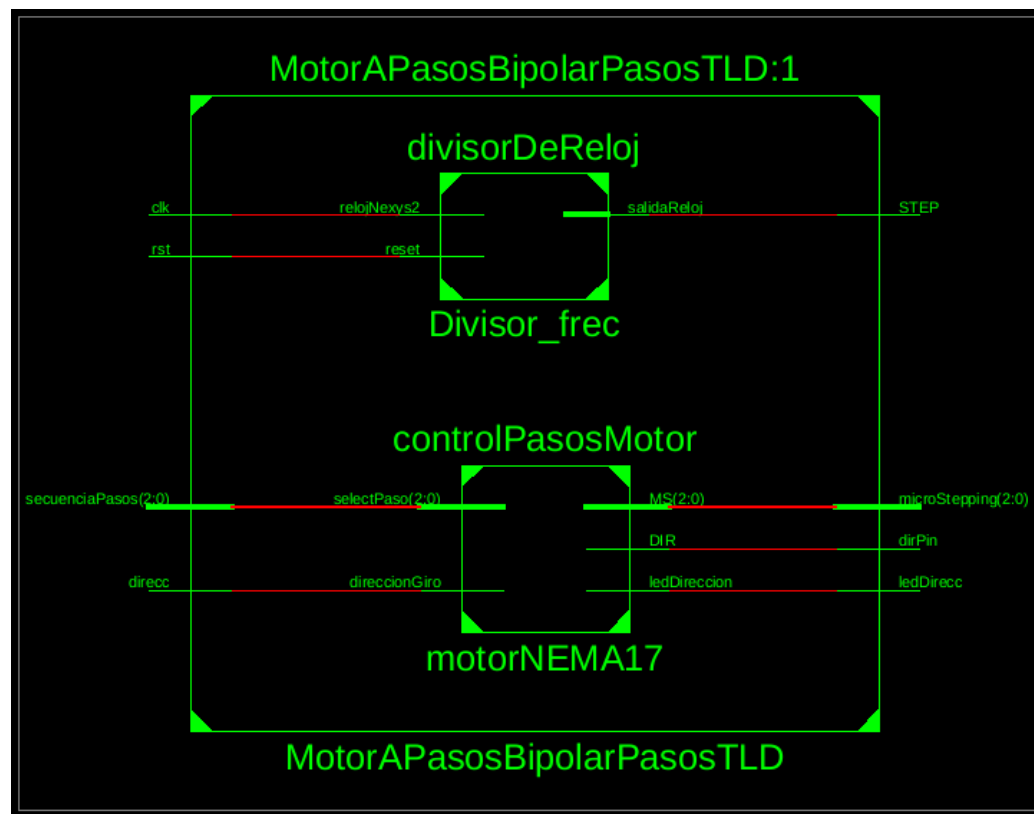
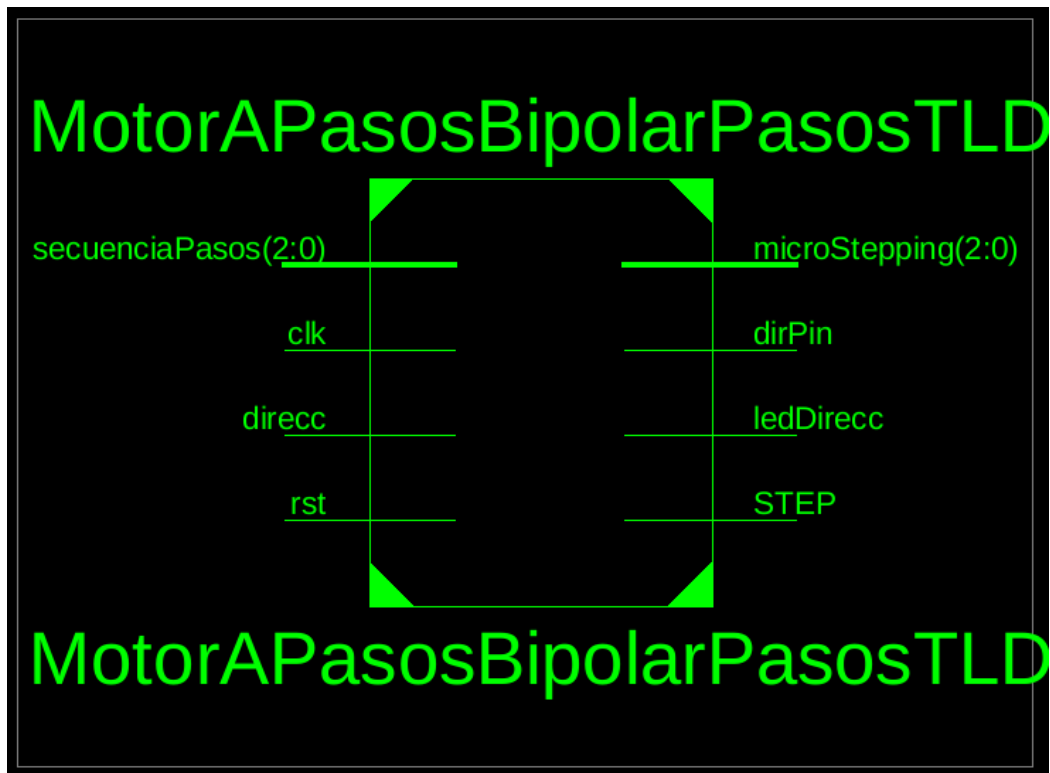
Table 3: Nexys2 Pmod Connector Pin Assignments							
Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14 <sup>1</sup>
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15 <sup>2</sup>
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15 <sup>3</sup>
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4: P18	JD10: J14 <sup>4</sup>

Notes: <sup>1</sup> shared with LD3 <sup>2</sup> shared with LD3 <sup>3</sup> shared with LD3 <sup>4</sup> shared with LD3



Código Verilog - Paso Doble, 1/2, 1/4, 1/8 y 1/16:

Diagrama TLD:



## Divisor de Reloj:

```
//1.-DIVISOR DE RELOJ:
//Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

module divisorDeReloj(
    input relojNexys2, //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input reset,       //Botón de reset.
    output salidaReloj //Reloj que quiero con una frecuencia menor a 50MHz.
);

    //REG: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo sirve
    //para almacenar y usar valores que sobrevivirán durante la ejecución del código y que además se deben usar dentro de
    //un condicional o bucle.
    reg [24:0] divisorDeReloj;
    //Este reg sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj,
    //dependiendo de la coordenada que elijamos, se tomara del vector para asignársela a la salida.

    //POSEDGE: La instrucción posedge() solo puede tener una entrada o reg dentro de su paréntesis y a fuerza se debe
    //declarar en el paréntesis del always@(), además hace que los condicionales o bucles que estén dentro del always@()
    //se ejecuten por si solos cuando ocurra un flanco de subida en la entrada que tiene posedge() dentro de su
    //paréntesis, el flanco de subida ocurre cuando la entrada pasa de valer 0 lógico a valer 1 lógico y el hecho de
    //que la instrucción posedge() haga que el código se ejecute por si solo, significa que yo directamente no debo
    //indicarlo con una operación lógica en el paréntesis de los condicionales o bucles, si lo hago me dará error,
    //aunque si quiero que se ejecute una acción en específico cuando se dé el flanco de subida en solo una de las
    //entradas que usan posedge(), debo meter el nombre de esa entrada en el paréntesis del condicional o bucle.
    //Si uso posedge() en el paréntesis de un always@(), todas las entradas de ese always@() deben ser activadas igual
    //por un posedge().
    always@(posedge(relojNexys2), posedge(reset))
    begin
        if(reset)
            //En VHDL se puede declarar a un número hexadecimal para evitar poner muchos bits de un numero
            //binario grande, pero en Verilog si hago esto el programa se confunde en el tipo de dato que esta
            //recibiendo y como consecuencia obtendremos un error.
            divisorDeReloj = 25'b00000000000000000000000000000000;
        else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado el Reset.
            //No debo poner el caso cuando if(relojNexys2) porque eso ya lo está haciendo la instrucción
            //always@(posedge(relojNexys2),...) por si sola.
            divisorDeReloj = divisorDeReloj + 1;
    end

    //Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
    //en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
    //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
    assign salidaReloj = divisorDeReloj[10];
    //El motor a pasos bipolar NEMA 17 con el paso completo puede recibir una frecuencia mínima de 1 Hz y máxima de 1000Hz,
    //esto nos da la posibilidad de elegir de la coordenada 15 (762.94 Hz) a la coordenada 24 (1.49 Hz) del divisor
    //de reloj.
endmodule
```

## Control Motor a Pasos Bipolar:

```
//2.-CONTROL DE MOTOR A PASOS:
//Modulo que usa el divisor de reloj para prender y apagar las 2 bobinas A y B del motor a pasos bipolar
//de tipo NEMA 17, este motor cuenta con un controlador A4988 y a través de sus pines STEP Y DIR se puede
//controlar su movimiento y sentido de giro, cada vez que el pin STEP reciba un pulso, dará un paso, la
//mayoría de motores bipolares dan pasos 200 pasos para dar una vuelta completa, osea que dan un giro de
//1.8 grados, algunos dan giros de 400 pasos, osea 0.9 grados de rotación por paso.
//En el motor bipolar se cuenta con 5 secuencias de pasos principales: Paso completo, medio paso, 1/4 de
//paso, 1/8 de paso y 1/16 de paso, mientras más se disminuya el paso, mejor será la precisión de su
//movimiento, pero su torque se reducirá proporcionalmente, el torque máximo del motor se alcanza en el
//paso completo, donde aguanta hasta 4kg de peso en una carga.

module controlPasosMotor(
    input direccionGiro,
    input [2:0] selectPaso,
    //Se pueden elegir 5 opciones de pasos: Paso Completo, 1/2 paso, 1/4 de paso, 1/8 de paso y 1/16 de paso.
    //Mientras sea menor el paso, mayor será su precisión, pero se reduce su torque de forma proporcional.
    output reg [2:0] MS, //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg [2:0] ledMS,
    output reg DIR,
    output reg ledDireccion
);

    //always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se
    //deben poner las entradas que usara y además tiene su propio begin y end.
    always@(direccionGiro, selectPaso)
    begin
        //CONDICIONAL CASE: Se usa para evaluar los diferentes valores de la variable que tenga en su
        //paréntesis y asignar una salida correspondiente a cada caso.
        case(selectPaso)
            //Con los 3 pines MS1, MS2 y MS3 de Micro Steping (MS), se establece el tipo de paso del motor para
            //lograr una mayor precisión o torque, se elige una de las siguientes opciones:
            //
            //      MS1    MS2    MS3
            //Paso Completo =    0      0      0
            //Paso Medio =      1      0      0
            //1/4 de Paso =      0      1      0
            //1/8 de Paso =      1      1      0
            //1/16 de Paso =     1      1      1
            //Mientras mas se reduzca el paso, mayor es la resolución de la precisión en su movimiento, pero el
            //torque se reduce proporcionalmente:
            //## Pasos 1 vuelta paso completo = 200 [pasos/revolución]
        endcase
    end
endmodule
```

```

    /// Pasos 1 vuelta cualquier otro paso = # Pasos 1 vuelta paso completo * 1/división de paso
    /// Pasos 1 vuelta de 1/4 de paso = 200 * 1/(1/4) = 200 * 4 = 800 [pasos/revolución]
    3'b000 : begin MS = 3'b000; ledMS = 3'b000; end //Paso completo.
    3'b001 : begin MS = 3'b100; ledMS = 3'b100; end //1/2 Paso.
    3'b010 : begin MS = 3'b010; ledMS = 3'b010; end //1/4 de Paso.
    3'b011 : begin MS = 3'b110; ledMS = 3'b110; end //1/8 de Paso.
    3'b100 : begin MS = 3'b111; ledMS = 3'b111; end //1/16 de Paso.
    //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones anteriores uso
    //default.
    default : begin MS = 3'b000; ledMS = 3'b000; end //Motor a pasos bipolar con paso completo.

endcase
if(direccionGiro == 1'b1) begin
    DIR = 1'b1; //DIR = 1; Giro del motor en sentido de las manecillas del reloj (derecha).
    ledDireccion = 1'b1;
end else begin
    DIR = 1'b0; //DIR = 0; Giro del motor en sentido contrario de las manecillas del reloj (izquierda).
    ledDireccion = 1'b0;
end

end
endmodule

```

## Módulo TLD:

```

//3.-TLD (Top Level Design) MotorAPasosBipolarPasosTLD:
//Este código sirve para unir los 2 módulos anteriores y poder aplicar las 5 configuraciones de pasos a un motor
//bipolar de tipo NEMA 17, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
//se encienden y apagan las 2 bobinas A y B del motor, esto indicara su velocidad por medio del pin STEP, adoptando
//valores de 1 a 1000 Hz, luego en el módulo controlMotor se indica el sentido de giro usando un switch y mandando ese
//bit al pin DIR del controlador, después por medio de switches se elige una secuencia, ya sea de paso completo, 1/2 paso
//1/4 de paso, 1/8 de paso o 1/16 de paso, los 3 bits que sirven para elegir el paso se asignan a los pines de micro
//stepping MS1, MS2 y MS3, dirigidas al controlador A4988.

module MotorAPasosBipolarPasosTLD(
    //Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto
    //se ve en el documento 12.-Motor a Pasos.
    input clk,
    input rst,
    input direcc,
    input [2:0] secuenciaPasos,
    output [2:0] microStepping,
    output [2:0] ledsMicroStepping,
    output ledDirecc,
    output STEP,
    output dirPin
);

//INSTANCIAS:
//Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
//un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
//módulo separadas por comas una de la otra, esto hará que lo que entre o salga del otro módulo, entre, salga o se
//guarde en este. La sintaxis que debemos usar es la siguiente:
//Nombre_Del_Modulo_Que_Queremos_Instanciar NombreInstancia (
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
//
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
//
//.Nombre_De_La_Entrada_Del_Modulo_Instanciado(Signal_En_Este_Modulo),
//.Nombre_De_La_Salida_Del_Modulo_Instanciado(Signal_En_Este_Modulo)
//);

//INSTANCIA DEL MODULO divisorDeReloj
divisorDeReloj Divisor_frec(
    .relojNexys2(clk),
    //La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
    .reset(rst),
    //La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
    .salidaReloj(STEP)
    //La salida salidaReloj del divisorDeReloj se asigna a la salida STEP de este módulo.
);

//INSTANCIA DEL MODULO controlPasosMotor
controlPasosMotor motorNEMA17(
    .direccionGiro(direcc),
    //La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
    .selectPaso(secuenciaPasos),
    //La entrada selectPaso del controlPasosMotor se asigna a la entrada secuenciaPasos de este módulo.
    .MS(microStepping),
    //La salida MS del controlPasosMotor se asigna a la salida microStepping de este módulo.
    .ledMS(ledsMicroStepping),
    //La salida ledMS del controlPasosMotor se asigna a la salida ledsMicroStepping de este módulo.
    .DIR(dirPin),
    //La salida DIR del controlPasosMotor se asigna a la salida dirPin de este módulo.
    .ledDireccion(ledDirecc)
    //La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
);

endmodule

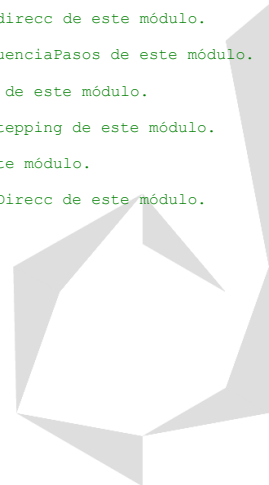
```

## Código UCF:

```

//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.

```



```

net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "R17"; //Entrada dirección de giro asignada al switch SW7 en el puerto R17.
//Bit mas significativo del selector de la secuenciaPasos conectado al switch SW2 en el puerto K18.
net "secuenciaPasos[2]" loc = "K18";
//Bit intermedio del selector de la secuenciaPasos conectado al switch SW1 en el puerto H18.
net "secuenciaPasos[1]" loc = "H18";
//Bit menos significativo del selector de la secuenciaPasos conectado al switch SW0 en el puerto G18.
net "secuenciaPasos[0]" loc = "G18";

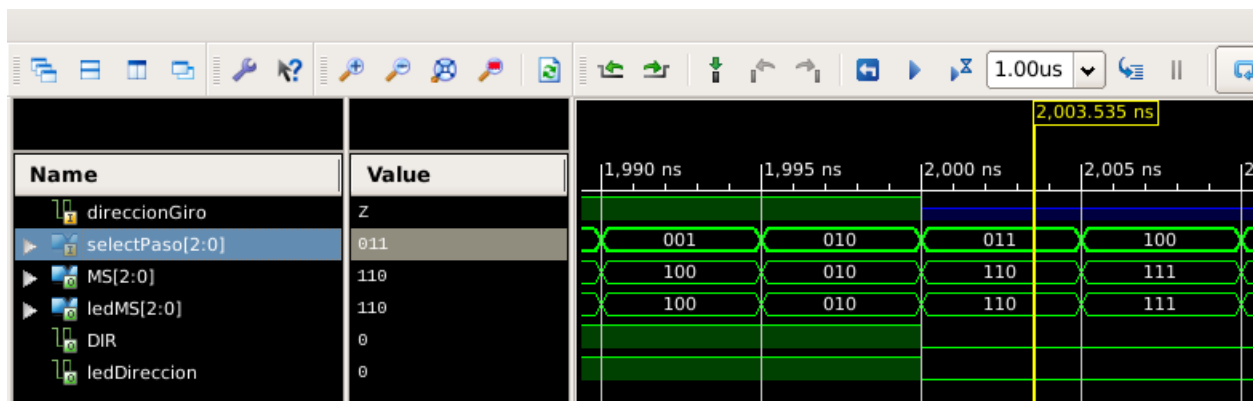
```

```

//SALIDAS:
net "microStepping[2]" loc = "L15"; //Bit MS1 conectado al puerto JA1 perteneciente a los puertos JA.
net "microStepping[1]" loc = "K12"; //Bit MS2 conectado al puerto JA2 perteneciente a los puertos JA.
net "microStepping[0]" loc = "L17"; //Bit MS3 conectado al puerto JA3 perteneciente a los puertos JA.
//Leds Micro Stepping: Pines MS1, MS2 y MS3.
net "ledsMicroStepping[2]" loc = "K15"; //Salida MS3 conectada al led LD2 perteneciente al puerto K15.
net "ledsMicroStepping[1]" loc = "J15"; //Salida MS2 conectada al led LD1 perteneciente al puerto J15.
net "ledsMicroStepping[0]" loc = "J14"; //Salida MS1 conectada al led LD0 perteneciente al puerto J14.
//Salida ledDirecc asignada al led LD7 en el puerto R4.
net "ledDirecc" loc = "R4";
//Salida STEP asignada al puerto JA7 perteneciente a los puertos JA, con código K13.
net "STEP" loc = "K13";
//Salida STEP asignada al puerto JA8 perteneciente a los puertos JA, con código L16.
net "dirPin" loc = "L16";

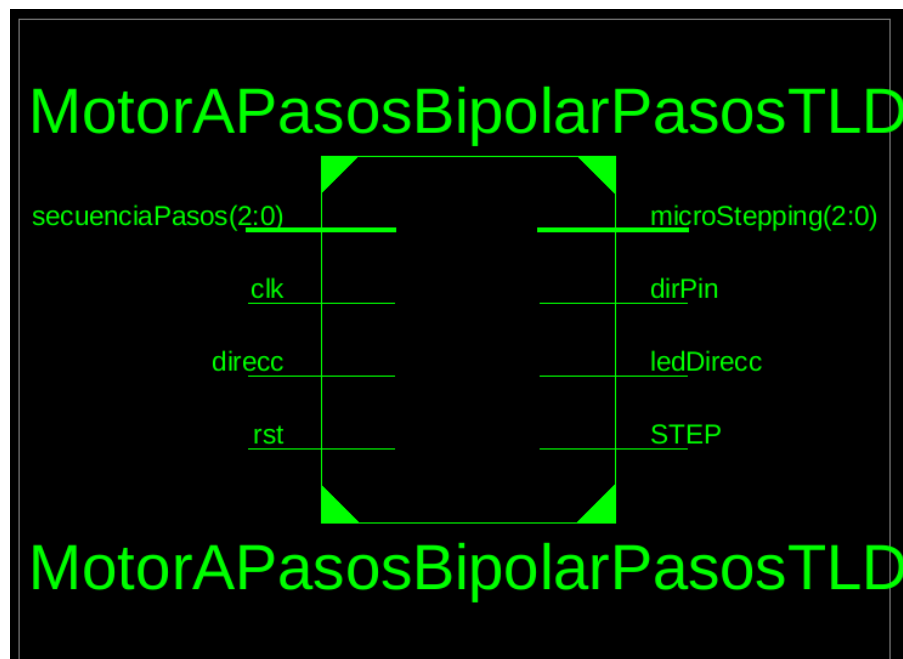
```

Simulación Paso Completo, Medio, 1/4, 1/8 y 1/16:

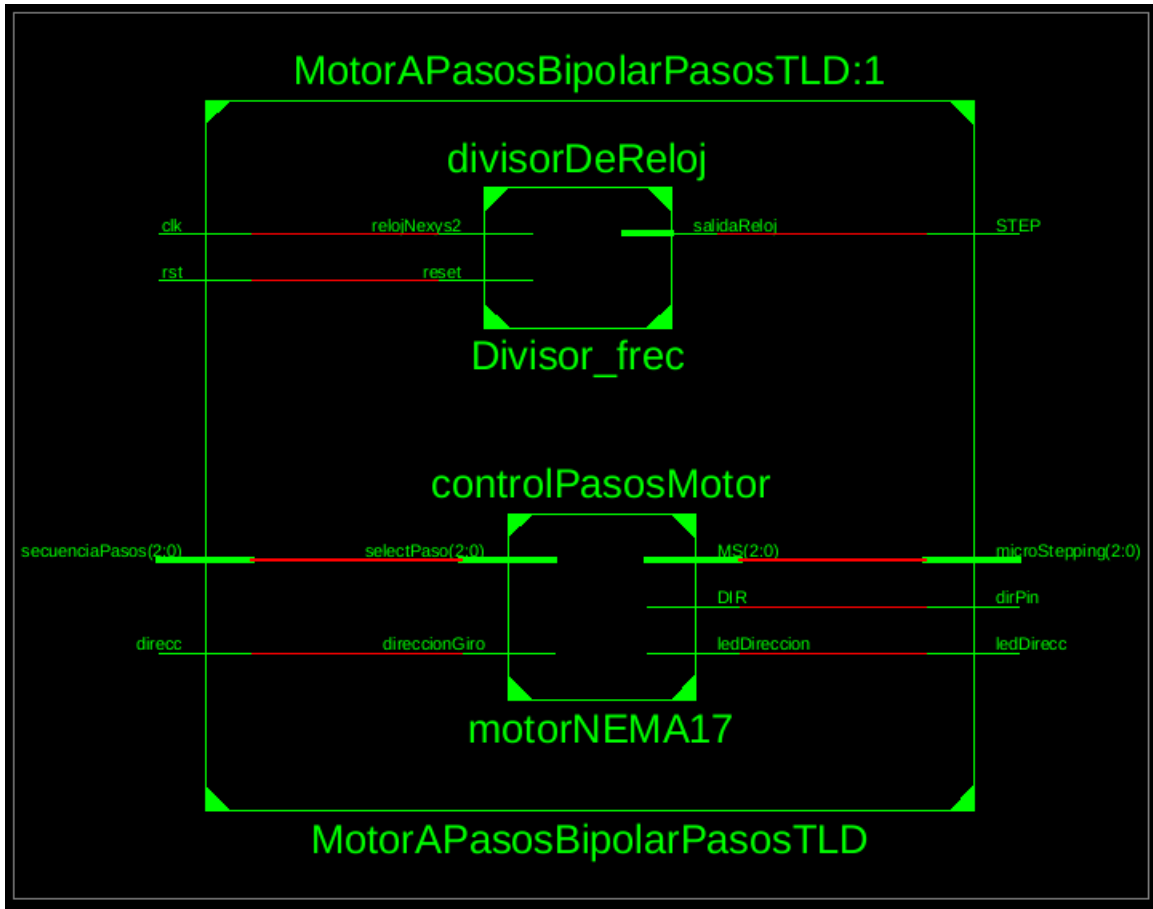


Código VHDL - Paso Doble, 1/2, 1/4, 1/8 y 1/16:

Diagrama TLD:







## Divisor de Reloj:

```
--1.-DIVISOR DE RELOJ:
--Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.

entity divisorDeReloj is
    Port ( relojNexys2 : in  STD_LOGIC;  --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
          reset       : in  STD_LOGIC;   --Botón de reset.
          salidaReloj : out STD_LOGIC);  --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí se hará el divisor de reloj haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeReloj is
    --SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
    --existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
    --arquitectura y antes de su begin, se le asignan valores con el simbolo :=
    signal divisorDeReloj : STD_LOGIC_VECTOR (24 downto 0);
    --Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
    --dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.
begin
    process(relojNexys2, reset)
    begin
        if(reset = '1') then--Cuando el Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciara.
            --NUMEROS HEXADECIMALES EN VHDL: 1 dígito hexadecimal equivale a 4 bits en binario, esto nos puede
            --servir para poner un numero binario grande sin tener la necesidad de poner un valor de muchos
            --bits, como cuando debo llenar un vector de puros ceros, declaro un numero hexadecimal poniendo
            --X"numero hexadecimal".
            divisorDeReloj <= "00000000000000000000000000000000";
            --Poner X"000000" equivale a poner "00000000000000000000000000000000".
        elsif(rising_edge(relojNexys2)) then
            --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de
            --subida en la señal de reloj clkNexys2 proveniente de la NEXYS 2.
            divisorDeReloj <= divisorDeReloj + 1;--Esto crea al divisor de reloj.
        end if;
    end process;
end;
```

```

--Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
--en especifico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
salidaReloj <= divisorDeReloj(15);--En la tabla se ve que la coordenada 16 proporciona una frecuencia de 381.47Hz.
--El motor a pasos bipolar NEMA 17 con el paso completo puede recibir una frecuencia mínima de 1 Hz y máxima de 1000Hz,
--esto nos da la posibilidad de elegir de la coordenada 15 (762.94 Hz) a la coordenada 24 (1.49 Hz) del divisor
--de reloj.
end frecuenciaNueva;

```

## Control Motor a Pasos Bipolar:

```

--2.-CONTROL DE MOTOR A PASOS:
--Modulo que usa el divisor de reloj para prender y apagar las 2 bobinas A y B del motor a pasos bipolar
--de tipo NEMA 17, este motor cuenta con un controlador A4988 y a través de sus pines STEP Y DIR se puede
--controlar su movimiento y sentido de giro, cada vez que el pin STEP reciba un pulso, dará un paso, la
--mayoría de motores bipolares dan pasos 200 pasos para dar una vuelta completa, osea que dan un giro de
--1.8 grados, algunos dan giros de 400 pasos, osea 0.9 grados de rotación por paso.
--En el motor bipolar se cuenta con 5 secuencias de pasos principales: Paso completo, medio paso, 1/4 de
--paso, 1/8 de paso y 1/16 de paso, mientras más se disminuya el paso, mejor será la precisión de su
--movimiento, pero su torque se reducirá proporcionalmente, el torque máximo del motor se alcanza en el
--paso completo, donde aguanta hasta 4kg de peso en una carga.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: Aquí se declaran las entradas y salidas.
entity controlPasosMotor is
    Port ( direccionGiro : in STD_LOGIC;--Switch que indica la dirección del giro del motor a pasos.
          selectPaso : in STD_LOGIC_VECTOR (2 downto 0);--Switches para elegir una secuencia de paso.
          --Se pueden elegir 5 opciones de pasos: Paso Completo, 1/2 paso, 1/4 de paso, 1/8 de paso y
          --1/16 de paso. Mientras sea menor el paso, mayor será su precisión, pero se reduce su torque
          --de forma proporcional.
          M_S : out STD_LOGIC_VECTOR (2 downto 0);--Pines MS1, MS2 y MS3 que eligen el tipo de paso.
          ledMS : out STD_LOGIC_VECTOR (2 downto 0);
          DIR : out STD_LOGIC;
          ledDireccion : out STD_LOGIC);--Led indicativo de la dirección del giro del motor a pasos.
end controlPasosMotor;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end
--nombreArquitectura;
architecture Behavioral of controlPasosMotor is
begin
    --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
    --Las diferentes entradas se separan entre si por comas.
    process(direccionGiro, selectPaso) --Tiene su propio begin y end process;
    begin
        case(selectPaso)is
            --Con los 3 pines MS1, MS2 y MS3 de Micro Stepping (MS), se establece el tipo de paso del motor para
            --lograr una mayor precisión o torque, se elige una de las siguientes opciones:
            --
            --Paso Completo = 0 0 0
            --Paso Medio = 1 0 0
            --1/4 de Paso = 0 1 0
            --1/8 de Paso = 1 1 0
            --1/16 de Paso = 1 1 1
            --Mientras mas se reduzca el paso, mayor es la resolución de la precisión en su movimiento, pero el
            --torque se reduce proporcionalmente:
            --# Pasos 1 vuelta paso completo = 200 [pasos/revolución]
            --# Pasos 1 vuelta cualquier otro paso = # Pasos 1 vuelta paso completo * 1/división de paso
            --# Pasos 1 vuelta de 1/4 de paso = 200 * 1/(1/4) = 200 * 4 = 800 [pasos/revolución]
            when "000" => M_S <= "000"; ledMS <= "000"; --Paso completo.
            when "001" => M_S <= "100"; ledMS <= "100"; --1/2 Paso.
            when "010" => M_S <= "010"; ledMS <= "010"; --1/4 de Paso.
            when "011" => M_S <= "110"; ledMS <= "110"; --1/8 de Paso.
            when "100" => M_S <= "111"; ledMS <= "111"; --1/16 de Paso.
            when others => M_S <= "000"; ledMS <= "000"; --Motor a pasos bipolar con paso completo.
        end case;
        if(direccionGiro = '1') then
            DIR <= '1'; --DIR = 1; Giro del motor en sentido de las manecillas del reloj (derecha).
            ledDireccion <= '1';
        else
            DIR <= '0'; --DIR = 0; Giro del motor en sentido contrario de las manecillas del reloj (izquierda).
            ledDireccion <= '0';
        end if;
    end process;
end Behavioral;

```

## Módulo TLD:

```

--3.-TLD (Top Level Design) MotorAPasosBipolarPasosTLD:
--Este código sirve para unir los 2 módulos anteriores y poder aplicar las 5 configuraciones de pasos a un motor
--bipolar de tipo NEMA 17, para ello primero se ejecuta un divisorDeReloj, del cual se obtiene la frecuencia con la que
--se encienden y apagan las 2 bobinas A y B del motor, esto indicara su velocidad por medio del pin STEP, adoptando
--valores de 1 a 1000 Hz, luego en el módulo controlMotor se indica el sentido de giro usando un switch y mandando ese
--bit al pin DIR del controlador, después por medio de switches se elige una secuencia, ya sea de paso completo, 1/2 paso
--1/4 de paso, 1/8 de paso o 1/16 de paso, los 3 bits que sirven para elegir el paso se asignan a los pines de micro
--stepping MS1, MS2 y MS3, dirigidas al controlador A4988.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.

```

```

--Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto se ve
--en el documento 8.2.-Señal de reloj CLK Solamente en VHDL.
entity MotorAPasosBipolarPasosTLD is
    Port ( clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          direcc : in STD_LOGIC;
          secuenciaPasos : in STD_LOGIC_VECTOR (2 downto 0);
          microStepping : out STD_LOGIC_VECTOR (2 downto 0);
          ledsMicroStepping : out STD_LOGIC_VECTOR (2 downto 0);
          ledDirecc : out STD_LOGIC;
          STEP : out STD_LOGIC;
          dirPin : out STD_LOGIC);
end MotorAPasosBipolarPasosTLD;

--Arquitectura: Aquí voy a declarar todas las instancias de mis demás módulos para poderlos usar y sus signal.
architecture Behavioral of MotorAPasosBipolarPasosTLD is
begin
--INSTANCIAS:
    --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
    --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
    --módulo instanciado una entrada, salida o signal de este módulo, separadas por comas una de la otra, esto hará que
    --lo que entre o salga del otro módulo entre, salga o se guarde en este.
    --La sintaxis que debemos usar es la siguiente:

    --NombreInstancia          :          entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar          port map(
    --          Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
    --          Salida_Del_Modulo_Instanciado  => Salida_En_Este_Modulo,

    --          Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
    --          Salida_Del_Modulo_Instanciado  => Entrada_En_Este_Modulo,

    --          Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
    --          Salida_Del_Modulo_Instanciado  => Signal_En_Este_Modulo
    --);

    --INSTANCIA DEL MODULO divisorDeReloj
    Divisor_frec: entity work.divisorDeReloj port map(
        relojNexys2 => clk,
        --La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clk de este módulo.
        reset => rst,
        --La entrada reset del divisorDeReloj se asigna a la entrada rst de este módulo.
        salidaReloj => STEP
        --La salida salidaReloj del divisorDeReloj se asigna a la salida STEP de este módulo.
    );

    --INSTANCIA DEL MODULO controlPasosMotor
    motorNEMA17: entity work.controlPasosMotor port map(
        direccionGiro => direcc,
        --La entrada direccionGiro del controlPasosMotor se asigna a la entrada direcc de este módulo.
        selectPaso => secuenciaPasos,
        --La entrada selectPaso del controlPasosMotor se asigna a la entrada secuenciaPasos de este módulo.
        M_S => microStepping,
        --La salida M_S del controlPasosMotor se asigna a la salida microStepping de este módulo.
        ledMS => ledsMicroStepping,
        --La salida ledMS del controlPasosMotor se asigna a la salida ledsMicroStepping de este módulo.
        DIR => dirPin,
        --La salida DIR del controlPasosMotor se asigna a la salida dirPin de este módulo.
        ledDireccion => ledDirecc
        --La salida ledDireccion del controlPasosMotor se asigna a la salida ledDirecc de este módulo.
    );
end Behavioral;

```

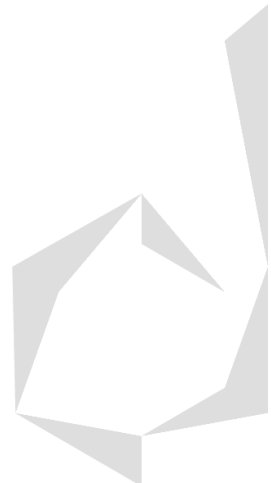
## Código UCF:

```

//ENTRADAS Y SALIDAS DEL archivo Top Level Design controlMotorAPasosUnipolarPasoSimpleTLD
//ENTRADAS:
net "clk" loc = "B8"; //Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13"; //Entrada reset asignada al push button BTN3 en el puerto H13.
net "direcc" loc = "R17"; //Entrada dirección de giro asignada al switch SW7 en el puerto R17.
//Bit mas significativo del selector de la secuenciaPasos conectado al switch SW2 en el puerto K18.
net "secuenciaPasos[2]" loc = "K18";
//Bit intermedio del selector de la secuenciaPasos conectado al switch SW1 en el puerto H18.
net "secuenciaPasos[1]" loc = "H18";
//Bit menos significativo del selector de la secuenciaPasos conectado al switch SW0 en el puerto G18.
net "secuenciaPasos[0]" loc = "G18";

//SALIDAS:
net "microStepping[2]" loc = "L15"; //Bit MS1 conectado al puerto JA1 perteneciente a los puertos JA.
net "microStepping[1]" loc = "K12"; //Bit MS2 conectado al puerto JA2 perteneciente a los puertos JA.
net "microStepping[0]" loc = "L17"; //Bit MS3 conectado al puerto JA3 perteneciente a los puertos JA.
//Leds Micro Stepping: Pines MS1, MS2 y MS3.
net "ledsMicroStepping[2]" loc = "K15"; //Salida MS3 conectada al led LD2 perteneciente al puerto K15.
net "ledsMicroStepping[1]" loc = "J15"; //Salida MS2 conectada al led LD1 perteneciente al puerto J15.
net "ledsMicroStepping[0]" loc = "J14"; //Salida MS1 conectada al led LD0 perteneciente al puerto J14.
//Salida ledDirecc asignada al led LD7 en el puerto R4.
net "ledDirecc" loc = "R4";
//Salida STEP asignada al puerto JA7 perteneciente a los puertos JA, con código K13.
net "STEP" loc = "K13";
//Salida STEP asignada al puerto JA8 perteneciente a los puertos JA, con código L16.
net "dirPin" loc = "L16";

```



## Referencias:

Jared Owen, “¿Cómo funciona un Motor Eléctrico? (Motor de corriente continua)”, 2020 [Online], Available: <https://www.youtube.com/watch?v=CWulQ1ZSE3c>

vt en línea, “Cómo funciona un motor brushless o sin escobillas”, 2019 [Online], Available: <https://www.youtube.com/watch?v=NnUiAgUundw&t=487s>

Mentalidad de Ingeniería, “Motores Paso a Paso”, 2022 [Online], Available: [https://www.youtube.com/watch?v=b\\_-PQCjyRRQ](https://www.youtube.com/watch?v=b_-PQCjyRRQ)

Bitwise Ar, “Arduino desde cero en Español - Capítulo 30 - Paso a Paso (unipolar) 28BYJ-48 con Driver ULN2003”, 2017 [Online], Available: <https://www.youtube.com/watch?v=2-nVV9S7leM>

Bitwise Ar, “Arduino desde cero en Español - Capítulo 33 - Paso a paso bipolar y A4988 controlador (driver)”, 2018 [Online], Available: <https://www.youtube.com/watch?v=u0SG681s8aA&t=16s>

Tecneu, “Nema 17 Motor A Pasos 17hs4401 4kg/cm 1.7a Bipolar”, 2021 [Datasheet Producto], Available: [https://articulo.mercadolibre.com.mx/MLM-943305569-nema-17-motor-a-pasos-17hs4401-4kgcm-17a-bipolar-\\_JM](https://articulo.mercadolibre.com.mx/MLM-943305569-nema-17-motor-a-pasos-17hs4401-4kgcm-17a-bipolar-_JM)

YISQIAA, “Motor A Pasos Fuente 12v Nema 17 Para Impresora 3d Y Cnc”, 2023 [Datasheet Producto], Available: [https://articulo.mercadolibre.com.mx/MLM-2027120528-motor-a-pasos-fuente-12v-nema-17-para-impresora-3d-y-cnc-\\_JM](https://articulo.mercadolibre.com.mx/MLM-2027120528-motor-a-pasos-fuente-12v-nema-17-para-impresora-3d-y-cnc-_JM)

Arduino.CC, “UNO R3”, 2023 [Datasheet Producto], Available: <https://docs.arduino.cc/hardware/uno-rev3>

Digilent, “Digilent Nexys2 Board Reference Manual”, 2008 [Datasheet Producto], Available: [https://digilent.com/reference/\\_media/reference/programmable-logic/nexys-2/nexys2\\_rm.pdf](https://digilent.com/reference/_media/reference/programmable-logic/nexys-2/nexys2_rm.pdf)

