INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

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

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

Servomotor con Arduino, Verilog y VHDL

Contenido

Partes y Tipos de Motores Eléctricos AC/DC

Tipos de Motores Eléctricos AC/DC

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

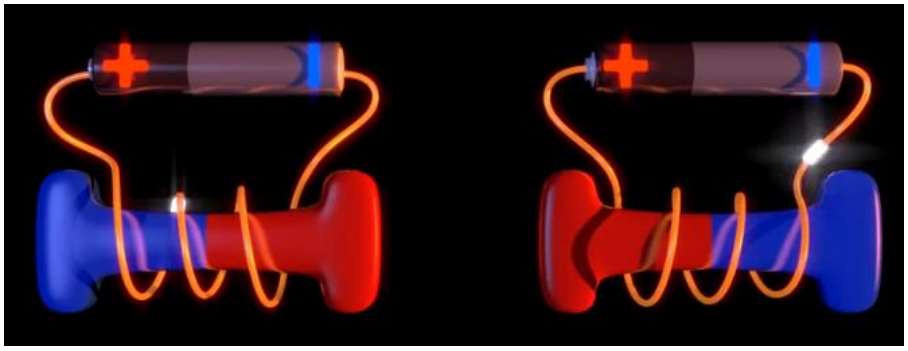
- **Motores DC:** Son impulsados por medio de corriente directa (pilas, rectificadores, etc).
  - **Motores con Escobillas:** Son motores que **realizan su conmutación con unas terminales llamadas escobillas**, que debido a su fricción deterioran el motor a través del tiempo.
    - **Motor DC Sencillo:** **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

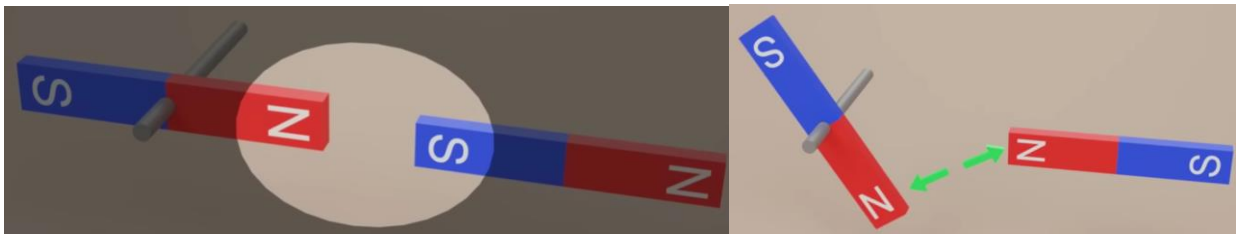
Para comprender el funcionamiento de cualquier motor eléctrico se debe conocer el funcionamiento básico de un electroimán, un electroimán suele estar compuesto de:

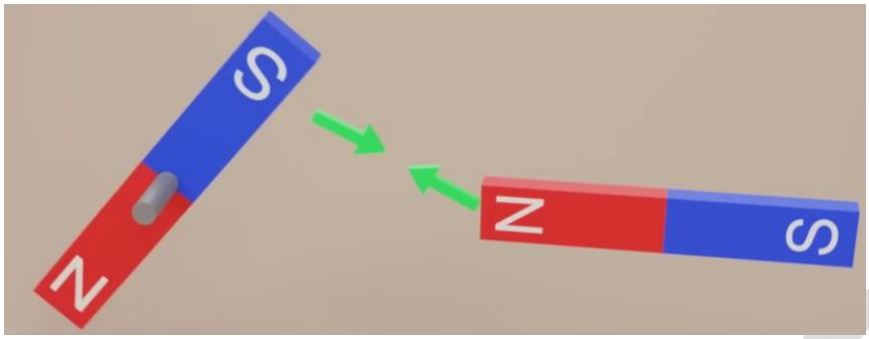
- **Metal ferromagnético:** Es un material que es conductor de campo magnético.
- **Bobina o inductor:** Es un alambre compuesto de un metal conductor eléctrico (usualmente cobre) enrollado alrededor del material ferromagnético mencionado previamente.

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).**



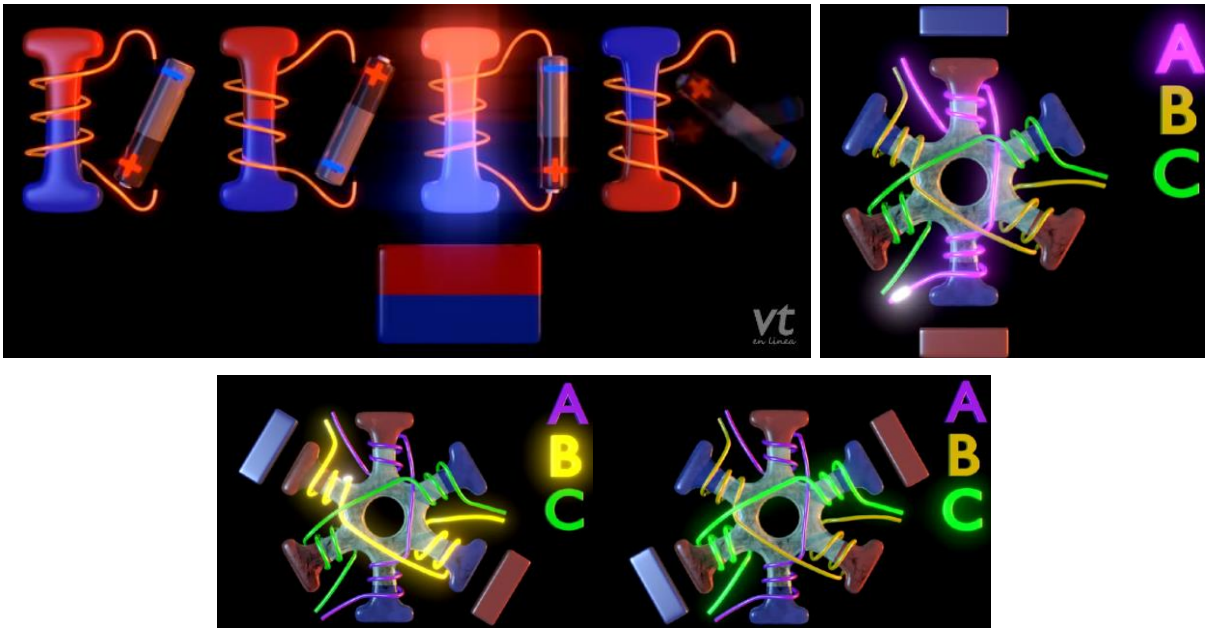
Esta acción convierte un metal ferromagnético cualquiera en un imán, y por medio de un magneto real (imán permanente) u otro electroimán se puede atraer o repeler sus polos (siempre y cuando el electroimán se encuentre encendido), generando así el **movimiento mecánico del motor**, siguiendo el principio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



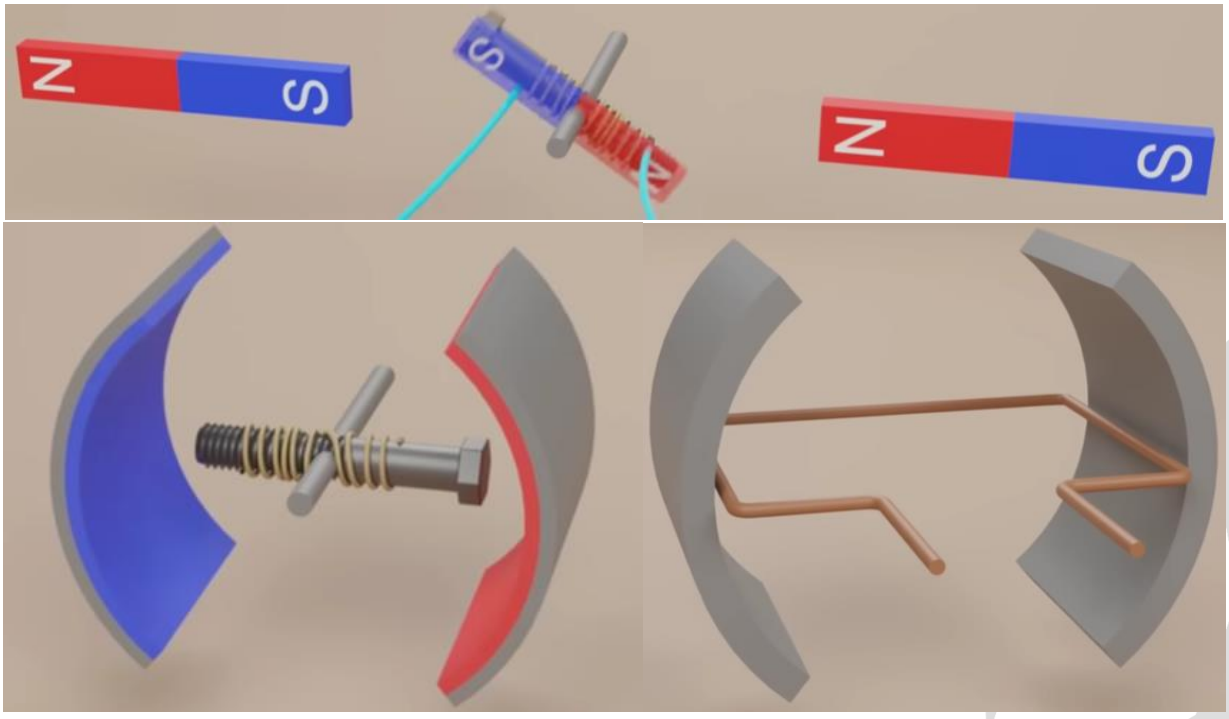


Basándonos en los conceptos explicados previamente, existen dos maneras globales de efectuar el movimiento de un motor eléctrico, que serán explicadas a continuación:

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**.



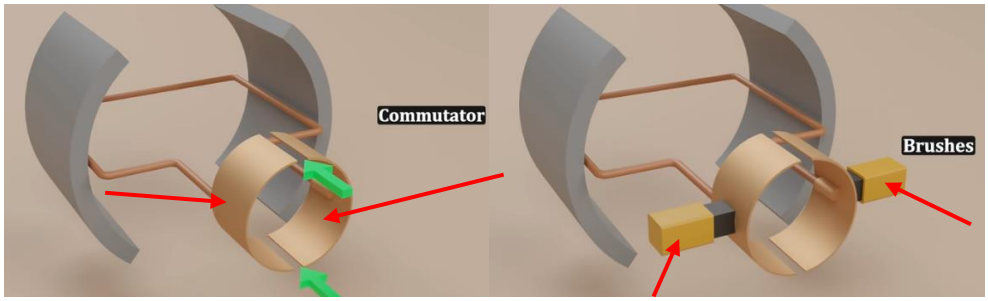
2. **Giro del electroimán:** Se puede contar con dos o más **imanes permanentes estáticos** para **mover el electroimán de un motor**, donde en vez de tener una bobina con un material ferromagnético dentro, se usarán varias bobinas que cambien el sentido de su corriente para invertir el sentido de su 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**.



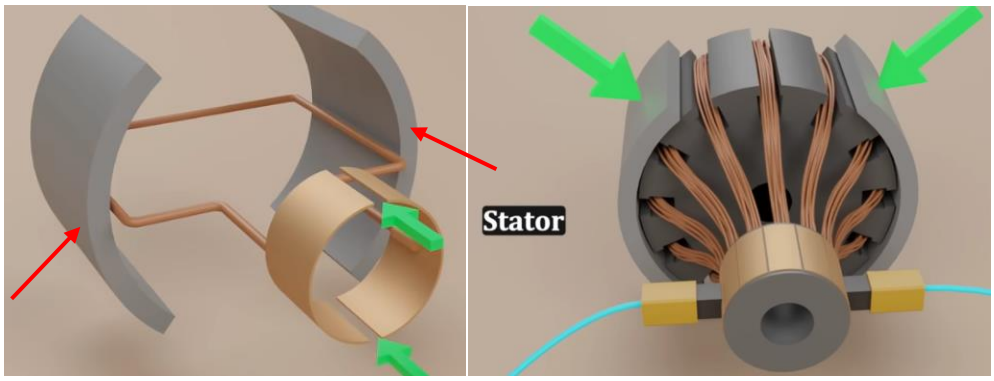
Partes de un Motor DC con Escobillas: Conmutador, Estator, Armadura y Rotor

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

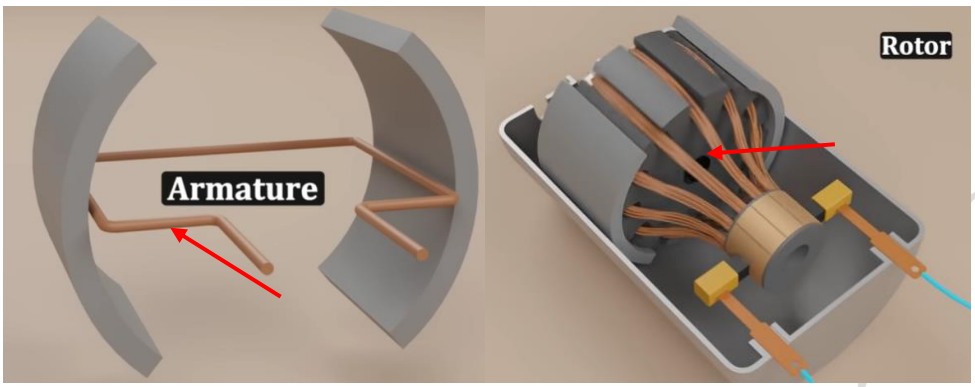
Los motores con escobillas se alimentan con corriente directa y la parte que realiza su conmutación, osea el cambio de dirección en la corriente de las bobinas pertenecientes a los electroimanes para generar su movimiento mecánico, son las mismas **escobillas** (o **brushes**) en conjunto con el **conmutador**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



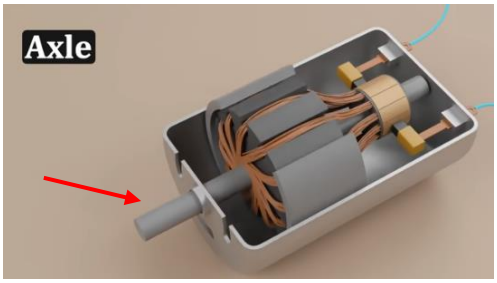
Los imanes permanentes usados para mover los electroimanes que se encienden y apagan en el motor son denominados como **estator** (stator), **siendo la parte que se mantiene estática en el motor**.



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**.



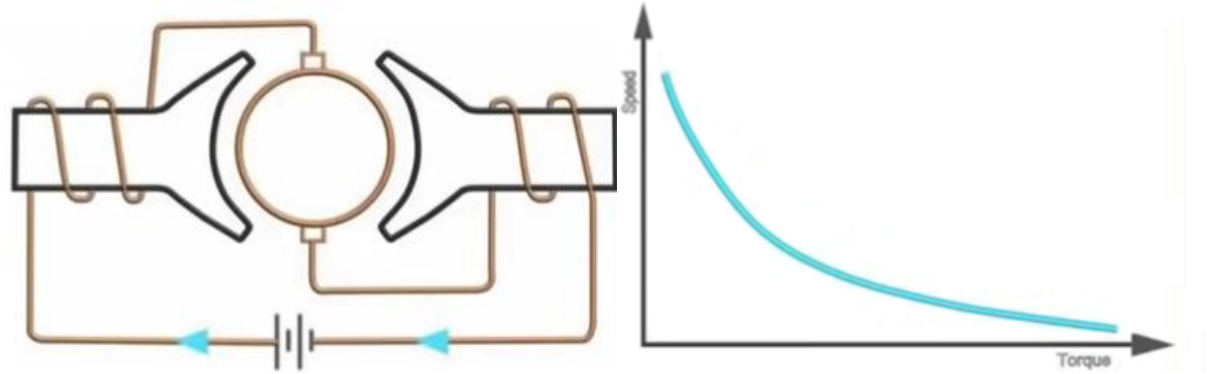
En la parte central del rotor se coloca el **eje del motor** (axle) y con esto finalizamos de nombrar todas las partes importantes de un motor DC con escobillas.



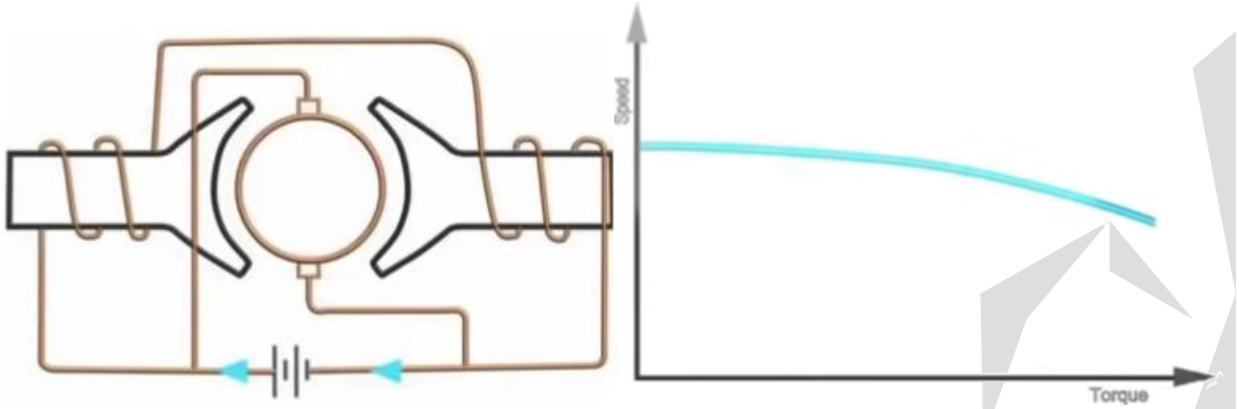
Conexión en Serie o Paralelo del Estator y Rotor de un Motor DC

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:

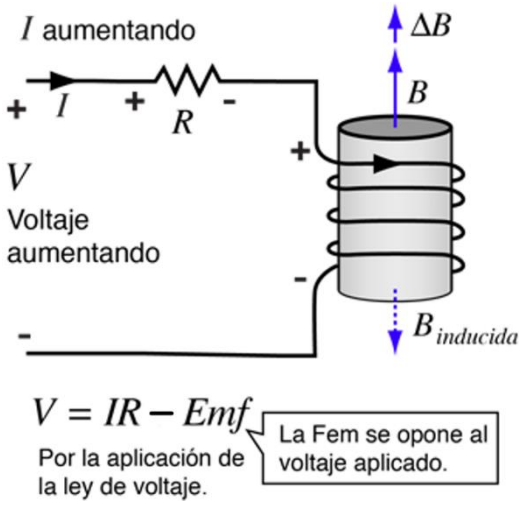
$\boldsymbol{B} = Flujo\ magnético\ inducido\ en\ un\ material\ [Tesla]$

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

$\boldsymbol{L} = Inductancia\ [Henrys]$

$\boldsymbol{i} = Corriente\ cambiante\ del\ circuito\ [Amperes]$

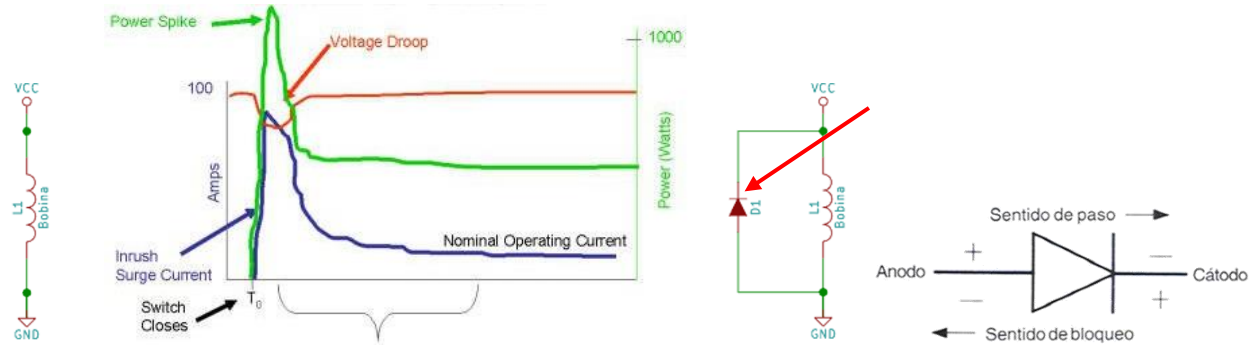
$$V_{fem} = V_{Emf} = L\left(\frac{di}{dt}\right)\ [V]$$



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 a dañar a los demás dispositivos electrónicos.



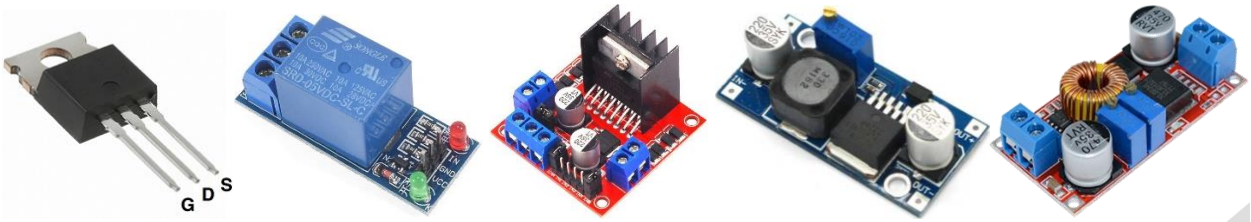
Etapa de Control y Etapa de Potencia en un Circuito con Motores

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.

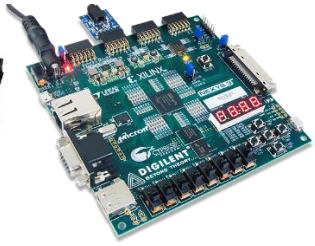
Por esta cuestión es que el circuito se divide en dos partes, una llamada **etapa de potencia** y otra llamada **etapa de lógica o control**:

- **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.

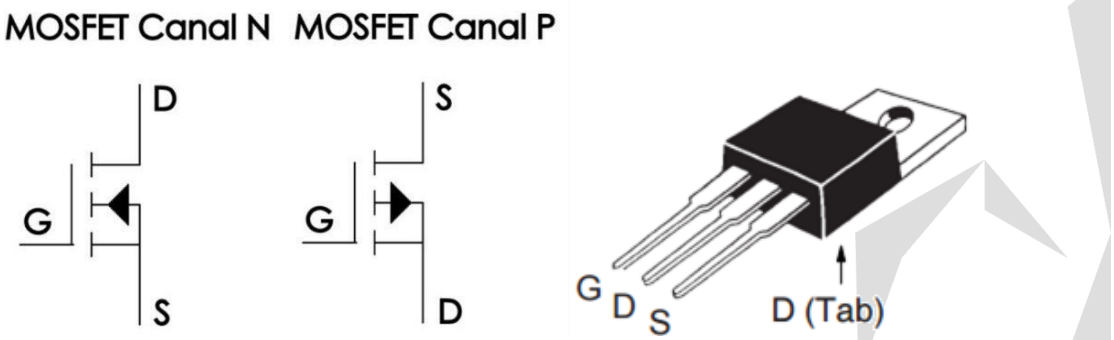



En pocas palabras, la etapa de control es el cerebro del circuito y la etapa de potencia es su músculo.

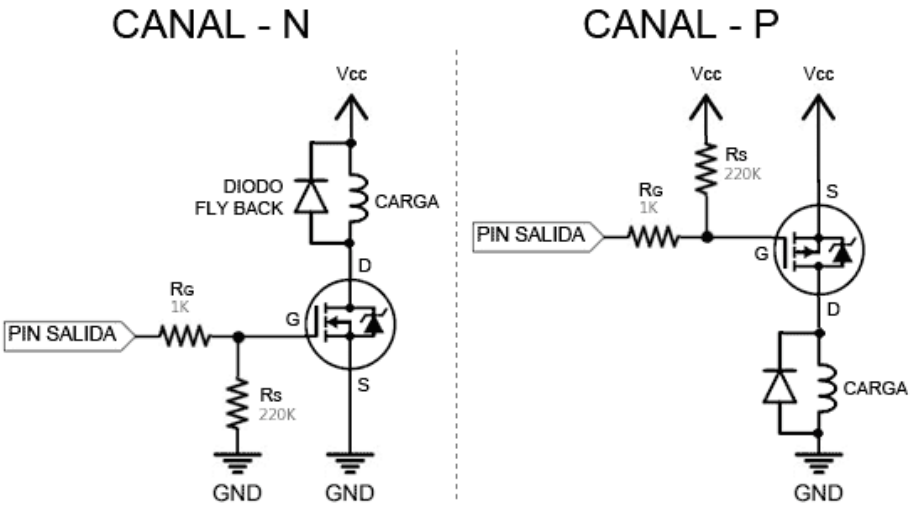
Etapa de Control: Aislamiento de los Circuitos Lógicos

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 usar los 3 siguientes componentes electrónicos para realizar el aislamiento del circuito:

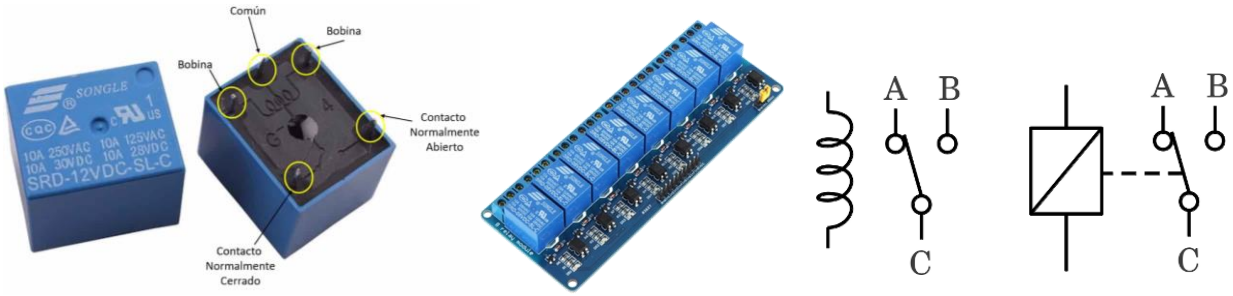
- **Transistor MOSFET:** Es un dispositivo electrónico que puede utilizarse en una configuración de **amplificador** o **interruptor** para separar la etapa de potencia y la etapa lógica.
    - **Amplificador:** Los transistores **pueden amplificar señales de control de bajo voltaje y corriente provenientes 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.



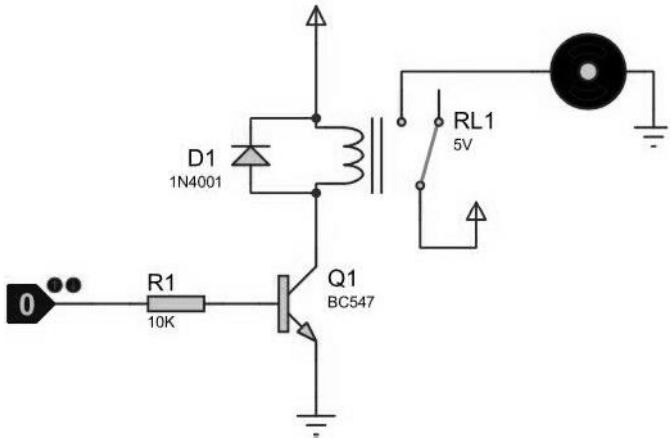
- A continuación, se muestra un ejemplo de conexión que aísla ambas etapas a través de un **transistor MOSFET**:



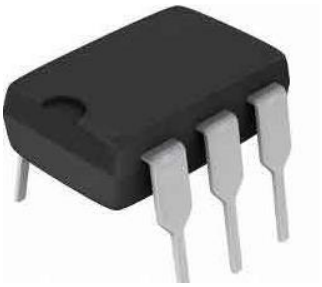
- **Relevador:** Un relevador o relé es un interruptor electromagnético que utiliza un electroimán interno para conectar o desconectar dos contactos mecánicos.
  - Los relés proporcionan un aislamiento eléctrico entre la **etapa de potencia** y la **etapa de control**, ya que no hay conexión eléctrica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

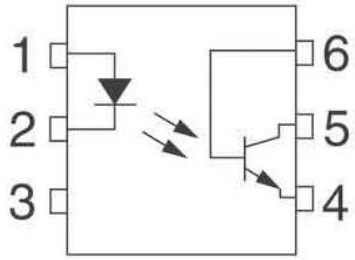


  - A continuación, se muestra un ejemplo de conexión que aísla ambas etapas a través de un **relevador o relé**:

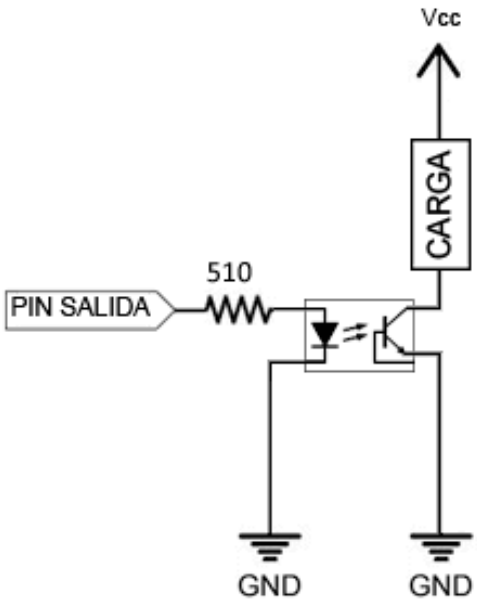


- **Optoacoplador (optoaislador):** Es un dispositivo electrónico que utiliza una combinación de un **emisor de luz** (generalmente un diodo emisor de luz o LED) y un **receptor de luz** (generalmente un fototransistor) para proporcionar un aislamiento eléctrico entre la **etapa de potencia** y la **etapa 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**:

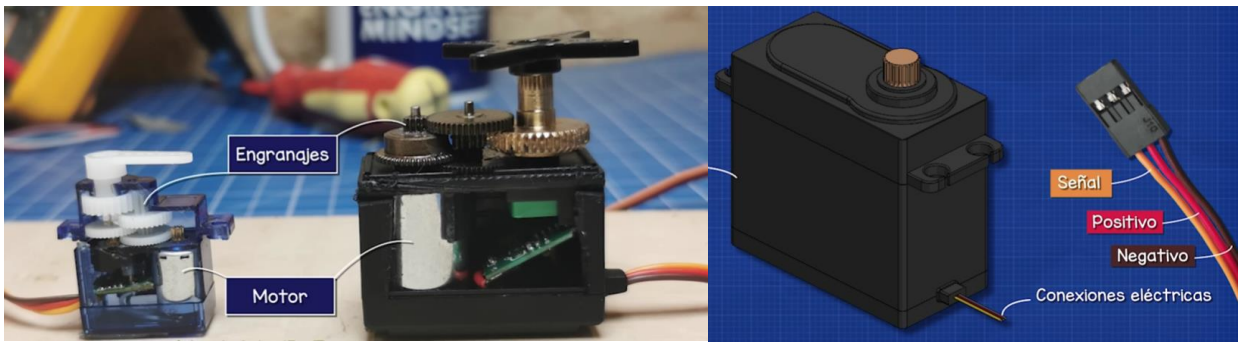


En los ejemplos posteriores donde se explicará el funcionamiento y control de los motores no se separa el circuito en una **etapa de potencia** y **etapa lógica**, esto se realiza por simplicidad en la demostración, pero ya en aplicaciones reales si se debe realizar el aislamiento de ambas partes del 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, por ello es importante también no colocarle ninguna carga al motor si es que se quiere probar su funcionamiento conectando el circuito directamente.

Servomotor

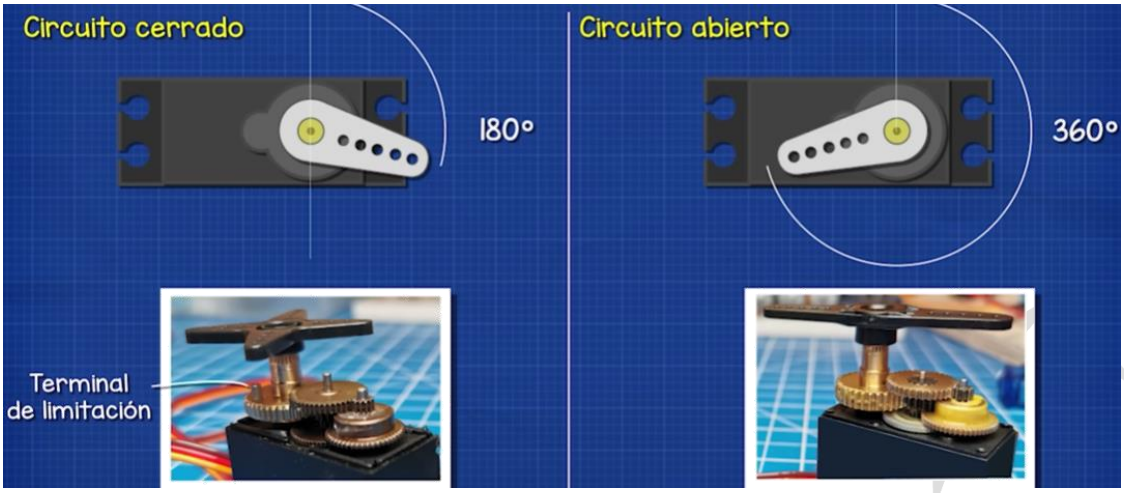
El servomotor es un motor DC con escobillas simple, su **estator se conforma de dos imanes permanentes estáticos que se encuentran en su perímetro** y su **rotor se encuentra en su parte central**. La mayor diferencia que tiene contra los motores convencionales es que **posee la mejor precisión en el control de posición de los motores DC**, **logrando detenerse en un punto deseado y mantener esa posición**, por lo que es ampliamente utilizado en automatización y robótica.

Internamente el servomotor es un simple motor de corriente directa, que **gira de forma continua sin control** cuando se le conecta a una tensión eléctrica, pero el servomotor no se controla de esa manera, además de alimentarlo, forzosamente se le debe mandar una **señal PWM (Pulse Width Module) para controlar su movimiento**, **diciéndole exactamente hasta qué punto debe girar**, esto se realiza por medio de su **electrónica interna**; por último el servomotor también contiene engranajes mecánicos en su punta, que reducen la velocidad de rotación de su eje, aumentando su **torque**.



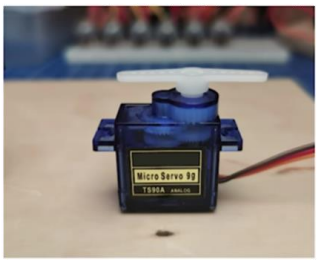
Dependiendo del tipo de giro que se tenga, el servomotor se puede clasificar en 2 categorías:

- **Servomotor de circuito cerrado:** Es el servomotor más común, proporciona el mejor control de posición y solo alcanza a girar de 0 a 180°.
    - *Suele tener un perno en su interior llamado "terminal de limitación" que físicamente evita que el motor siga girando después de cierto punto.*
- **Servomotor de circuito abierto:** Este servomotor alcanza a dar una vuelta completa, girando de 0 a 360°.



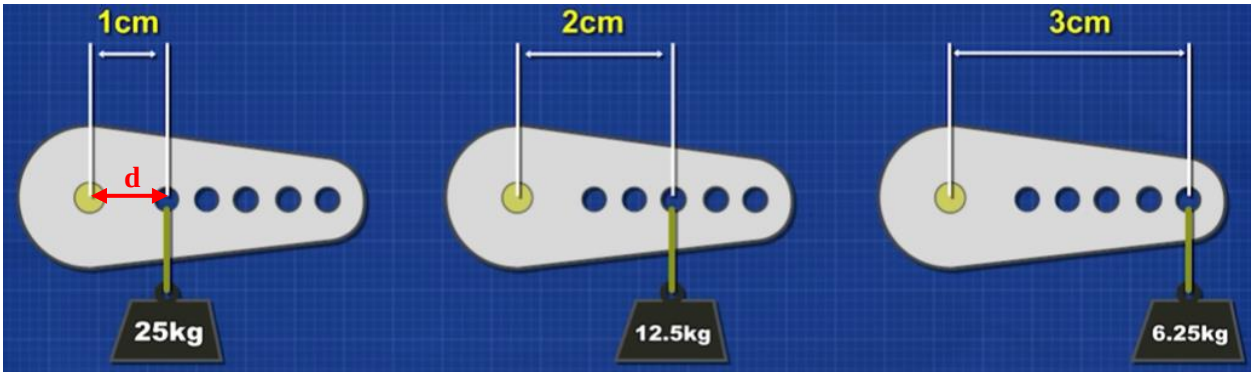
En la etiqueta del servomotor usualmente se suele encontrar un valor de peso, este no es el peso del motor, sino que representa su **torque**, que es la **fuerza de rotación del motor por unidad de cm** [**kg·cm**]; lo cual significa que **el motor podrá levantar cierto peso, si este está siendo aplicado a 1 centímetro de distancia del eje de rotación del motor**, mientras más se aleje la fuerza del eje, menor es el peso que aguanta el motor.

25kg

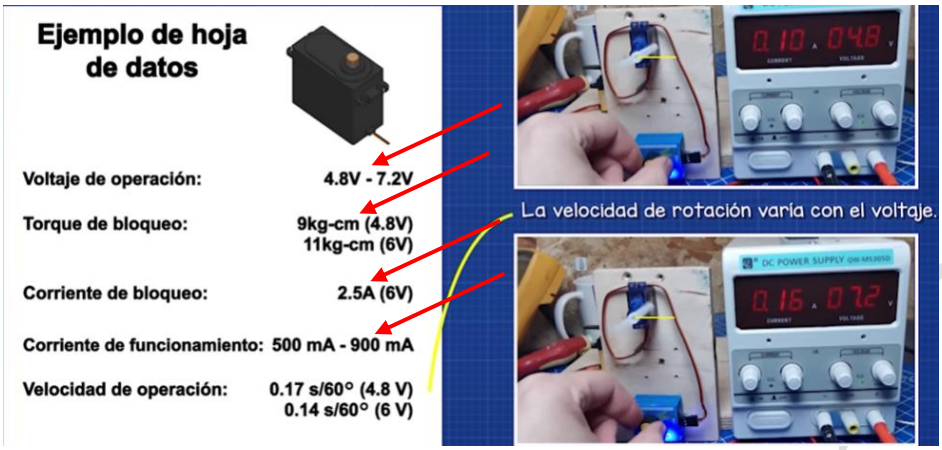


9g (0.009kg)

El torque resultante de la fuerza aplicada que va en sentido contrario al **torque del eje del servomotor** se obtiene al multiplicar el valor del peso por la gravedad y luego multiplicar eso por la distancia del **brazo de palanca**, que es perpendicular a la dirección de la carga: $\boldsymbol{T_c} = W * g * d = Peso * 9.81 \left[\frac{m}{s}\right] * \boldsymbol{d}\ [N \cdot m]$

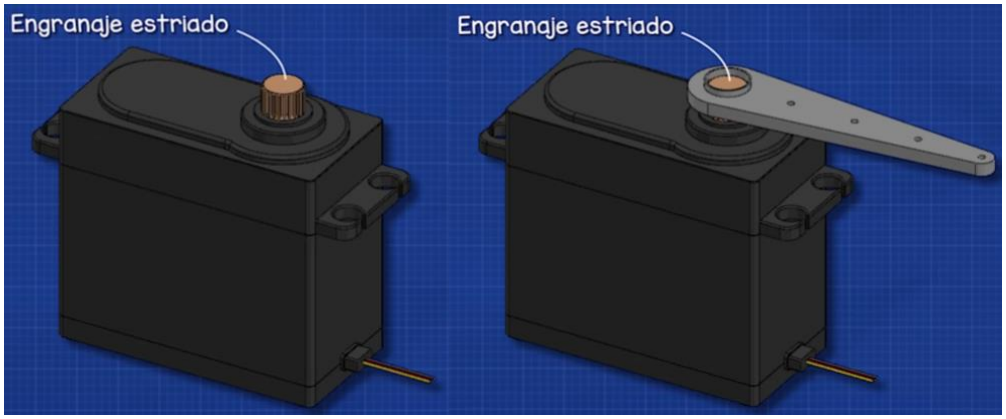


Normalmente los motores dan un rango de tensión de alimentación, cuanto más alto sea él voltaje de alimentación, mayor será el **torque** generado por las bobinas del motor y **llegará más rápidamente a la posición deseada**, pero aun así se tiene un límite en la carga que soporta y **si al motor se le somete a esa fuerza máxima que se opone a su movimiento, se bloqueará, dejando de girar, y al suceder esto, la corriente que demandan las bobinas subirá drásticamente, a esto se le llama corriente de bloqueo**.

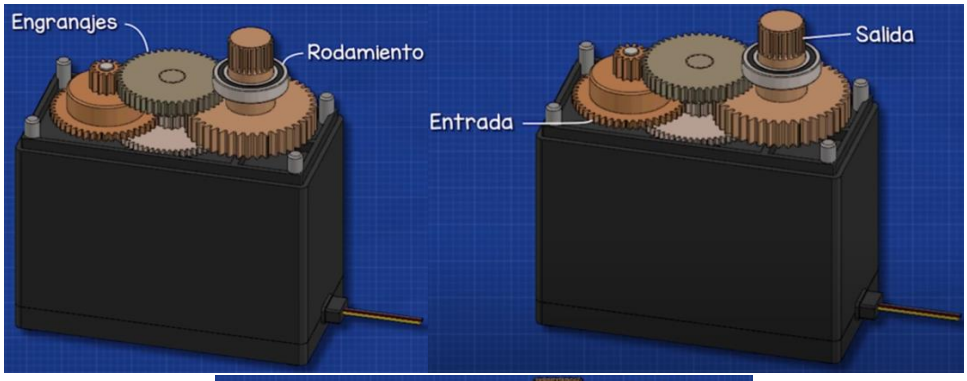


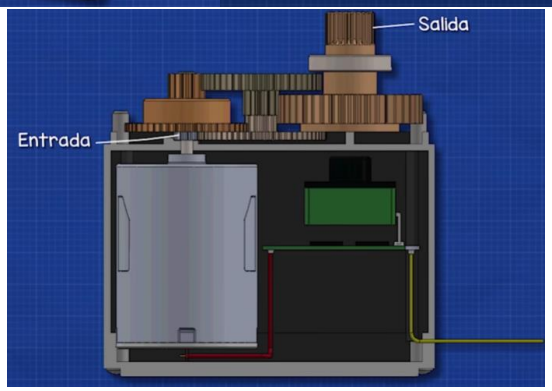
Las partes principales de un servomotor son:

- **Carcasa:** Es la envoltura de plástico que rodea a los elementos del servomotor.
- **3 cables de conexiones eléctricas**: A través de ellos se alimenta el servomotor y se controla su posición.
    - **VCC** y **GND**: Alimentación del motor.
    - **Señal**: Por aquí se manda una señal PWM para controlar la posición del eje del motor.
- **Engranaje estriado en el eje:** Es un pequeño engrane que se encuentra sobre el eje, donde se pueden colocar distintos accesorios para hacer uso de la rotación del motor.

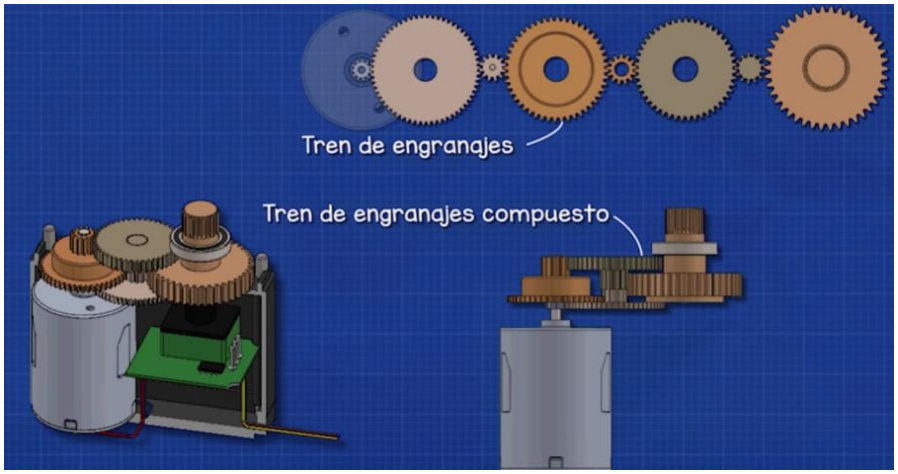


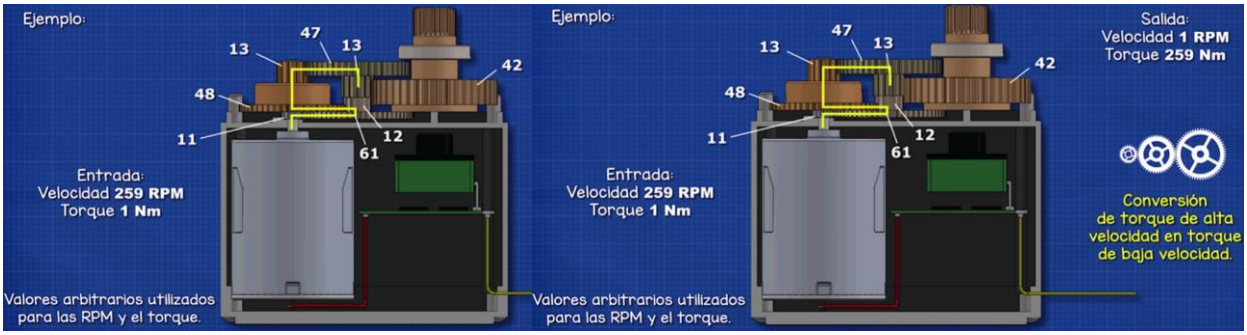
- **Tren de engranes compuesto:** Esta serie de engranes interconectados reducen la velocidad del eje del motor, aumentando su **torque**, ya que la potencia mecánica de un sistema rotatorio se calcula multiplicando su velocidad por su torque y si se aumenta una, se reduce la otra, pero la potencia se mantiene igual: $\boldsymbol{Pot = w * T}\ [Watts]$; $\boldsymbol{w}$: $Velocidad\ angular\ \left[\frac{rad}{s}\right]$; $\boldsymbol{T}$: $Torque[N \cdot m]$



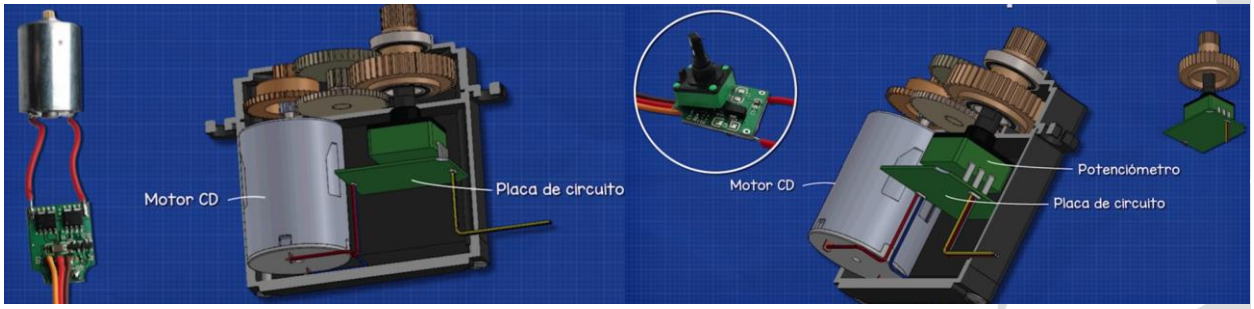


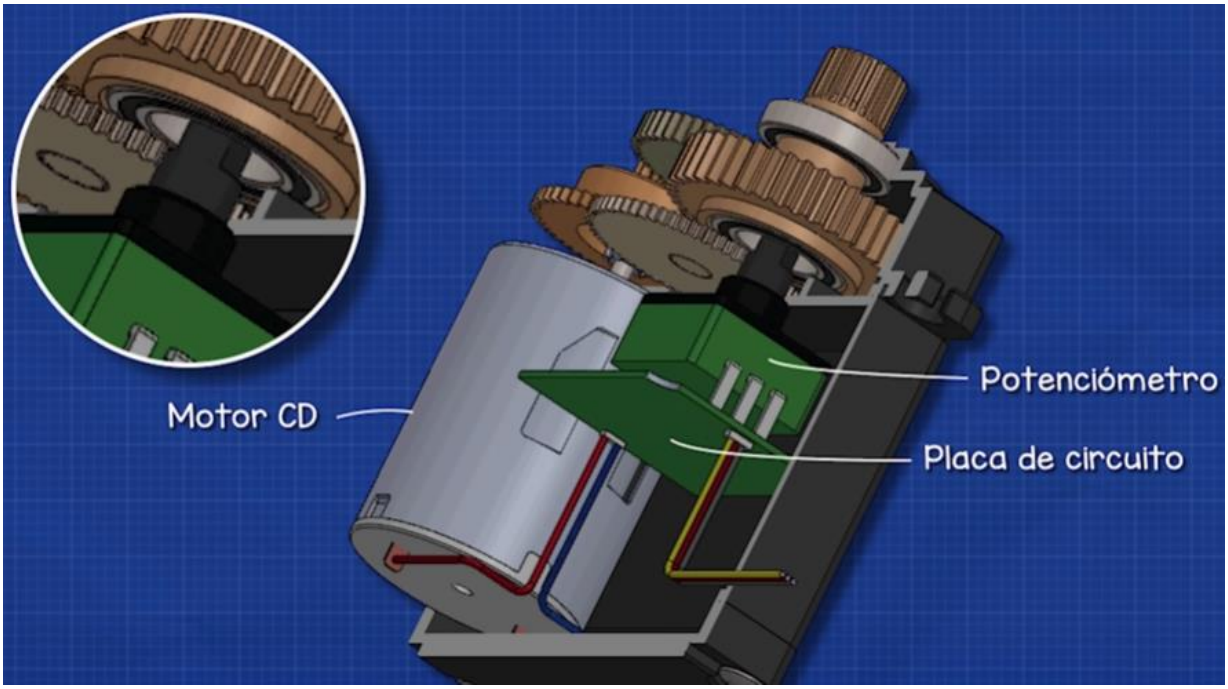
- Inicialmente el motor DC tiene una alta velocidad de rotación, pero bajo torque, razón por la cual, **los engranajes conectados a su eje la convierten en una salida de baja velocidad, pero alto torque**, **esto se logra porque al conectar engranes de distintos diámetros, su velocidad de rotación se ve afectada.** A continuación, se mostrará un diagrama donde se indica el número de dientes de cada engrane para demostrar el camino que se recorre para ocasionar la reducción de su velocidad.



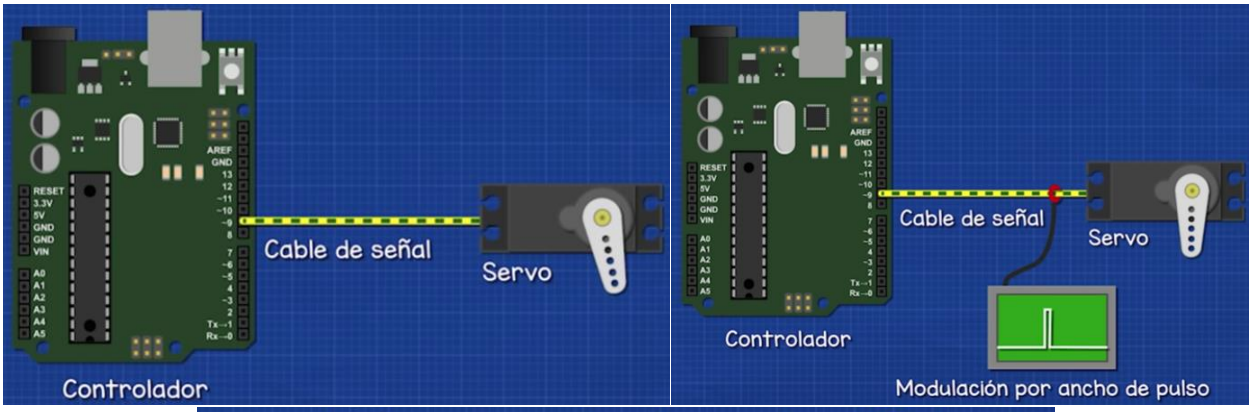


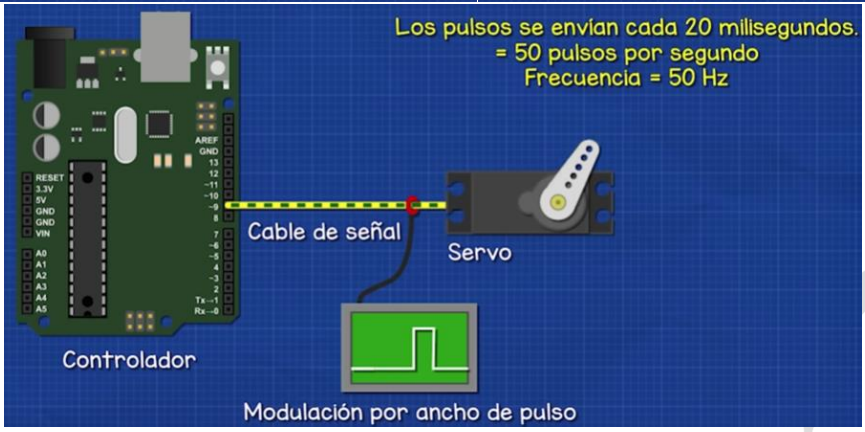
- **Electrónica interna:** El motor de CD está conectado a una pequeña **placa electrónica** dentro de la carcasa del servomotor, esta controla la rotación y sentido de giro (dirección) del eje del motor.
    - **Potenciómetro:** En la **electrónica interna** del servomotor se encuentra un **potenciómetro**, que es un dispositivo cuya resistencia varía al girar su perilla, este a su vez está unido al eje de salida del servomotor, por lo cual cuando su eje gire, la resistencia del potenciómetro variará de la misma forma.
        - La placa que gestiona la **electrónica** del circuito lee el valor de resistencia en el **potenciómetro** para saber en que posición se encuentra exactamente.





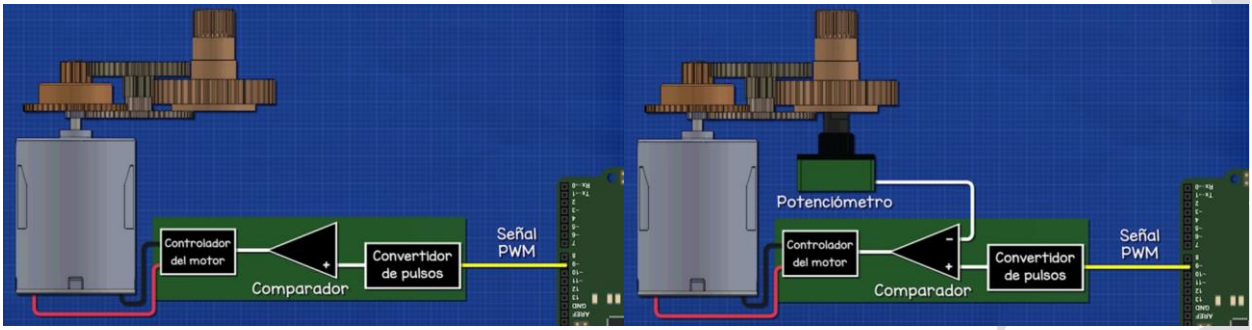
- **Controlador:** El controlador es un **dispositivo externo** que envía una **señal PWM** (**Pulse Width Module**) con una frecuencia de 50 Hz ($T = \frac{1}{f} = \frac{1}{50} = 0.02\ [s] = 20\ [ms]$) al servomotor para indicarle a qué posición debe llegar, **el controlador puede ser un Arduino (microcontrolador), una Nexys 2 (FPGA), una Raspberry Pi (CPU) o un circuito especializado probador de servos**.



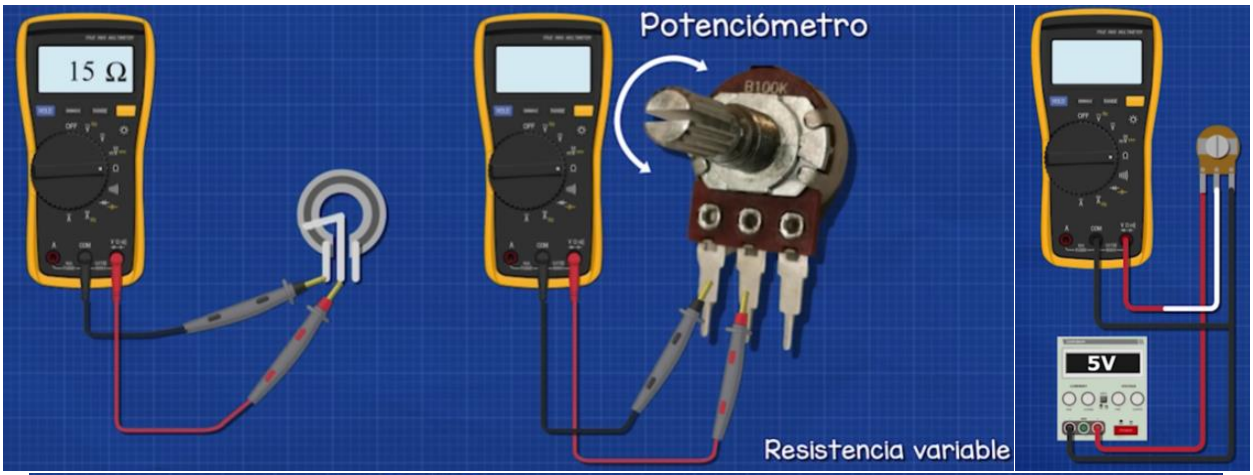


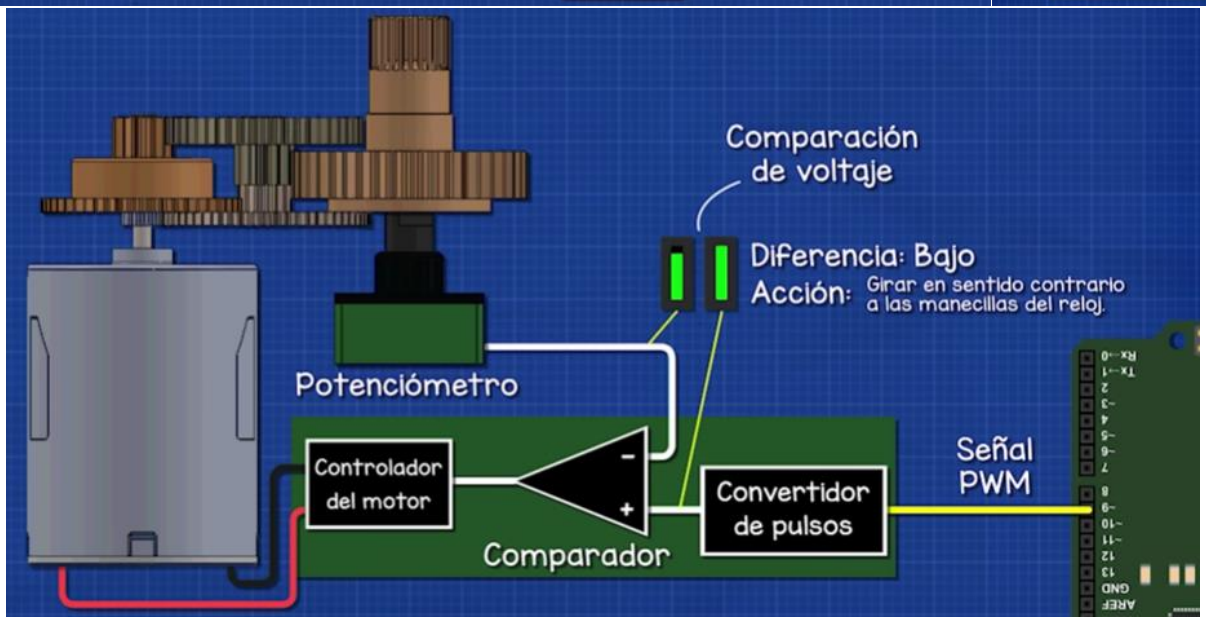
- El ancho del pulso, también llamado porcentaje elevado o duty cycle determina la posición del servomotor:
  - **Si enviamos un pulso ancho**: El servomotor **gira hacia la izquierda**, osea **en sentido contrario a las manecillas del reloj**.
  - **Si enviamos un pulso delgado**: El servomotor **gira hacia la derecha**, osea **en el mismo sentido de las manecillas del reloj**.
    - **Mientras el ancho del pulso se mantenga, el motor mantendrá su posición**, **cuando haya un cambio en el pulso, el motor se moverá**.
      - El motor utiliza **corriente cuando se mueve**, **aumenta la corriente que consume cuando se mueve rápido** y consume muy poca cuando **mantiene su posición**.

  - Al cambiar la altura de la señal (su amplitud), **no se afectará el movimiento del motor**. **Subir la tensión de la fuente solo afecta al torque aplicado en el eje**.
  - Al mover rápidamente el eje del motor, **este demandará más corriente**.
- **Controlador del motor:** Es un **puente H** que se encuentra en la **electrónica interna** del servomotor, este conmutará la corriente entregada a las bobinas del motor CD **para invertir la dirección de su giro, controlando su movimiento**.
  - Una de las terminales del **potenciómetro unido al eje** está conectada al **comparador** de la **electrónica interna**, **donde se compara el valor de la tensión ingresada por la señal PWM** proveniente del controlador externo **con la tensión del potenciómetro**, proporcionándole así **retroalimentación** al circuito y **comprobando que la posición indicada se haya alcanzado**, si la posición no ha sido alcanzada, el **puente H del controlador de motor** cambia el giro del motor hasta que se alcance.



- Se puede leer la tensión del **potenciómetro** porque está conectado a la tensión de alimentación del servomotor, este al girar y cambiar su nivel de resistencia, crea un divisor de tensión, que el **comparador lee y comprueba así la posición del motor**, **proporcionándolo con la mejor precisión**.





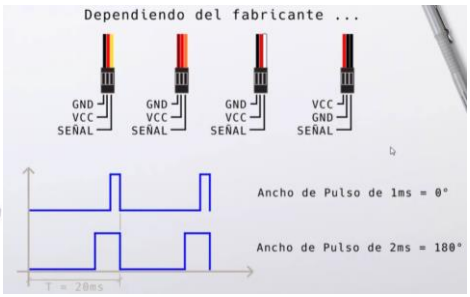
Existen varios tipos de servomotores, donde más que nada varía:

- Su tamaño.
- El rango de tensión de alimentación que reciben en sus pines **VCC** y **GND**.
- La **carga (peso)** que pueden soportar.

Aunque tienen sus diferencias, los servomotores comparten algunas características, como que la mayoría es controlada por una **señal moduladora de pulsos o PWM (Pulse Width Module)** con una **frecuencia de 50 Hz** ($T = \frac{1}{f} = \frac{1}{50} = 0.02\ [s] = 20\ [ms]$), esta frecuencia se refiere al periodo que hay en la señal al enviar un pulso, pero **después de mandar ese pulso, se puede declarar un tiempo de espera** para permitir que el servomotor se mueva gradualmente hacia la posición indicada, **esa frecuencia dictará su velocidad**.

Micro Servomotor SG90

- El servomotor **se alimenta con 4.8 a 6 V** en sus pines **VCC**/**GND**.
- Su posición es controlada por una **señal moduladora de pulsos o PWM** **(Pulse Width Module)** con una **frecuencia de 50 Hz** ($T = \frac{1}{f} = \frac{1}{50} = 0.02\ [s] = 20\ [ms]$).
    - Las posiciones que abarca el servomotor van de 0 a 180°.
- Cuenta con un pequeño **torque** de:
    - **1.2** [**kg·cm**] cuando se alimenta el servomotor con **4.8 V**.
    - **1.8** [**kg·cm**] cuando se alimenta el servomotor con **6 V**.
- **La velocidad de rotación del motor** se controla por medio de un tiempo de espera entre los pulsos de la señal PWM mandados al servomotor, este tiempo de espera tiene una frecuencia máxima de 2.8 Hz, la mínima no está definida, puede ser la que sea. *Usar frecuencias mayores a 2.8 Hz por mucho tiempo calienta el motor y/o causa que no llegue a tiempo a la posición deseada*.



Servomotor MG995

- El servomotor **se alimenta con 4.8 a 7.2 V** en sus pines **VCC**/**GND**.
- Su posición es controlada por una **señal moduladora de pulsos o PWM** **(Pulse Width Module)** con una **frecuencia de 50 Hz** ($T = \frac{1}{f} = \frac{1}{50} = 0.02\ [s] = 20\ [ms]$).
    - Las posiciones que abarca el servomotor van de 0 a 160°.
- Cuenta con un **torque** medio de:
    - **8.5** [**kg·cm**] cuando se alimenta el servomotor con **4.8 V**.
    - **10** [**kg·cm**] cuando se alimenta el servomotor con **7.5 V**.
- **La velocidad de rotación del motor** se controla por medio de un tiempo de espera entre los pulsos de la señal PWM mandados al servomotor, este tiempo de espera tiene una frecuencia máxima de 2.8 Hz, la mínima no está definida, puede ser la que sea. *Usar frecuencias mayores a 2.8 Hz por mucho tiempo calienta el motor y/o causa que no llegue a tiempo a la posición deseada*.

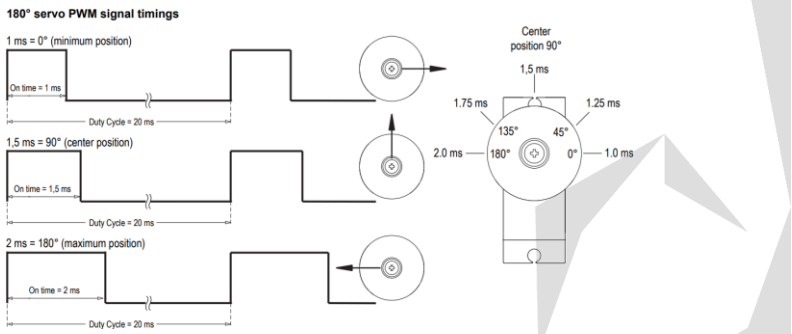
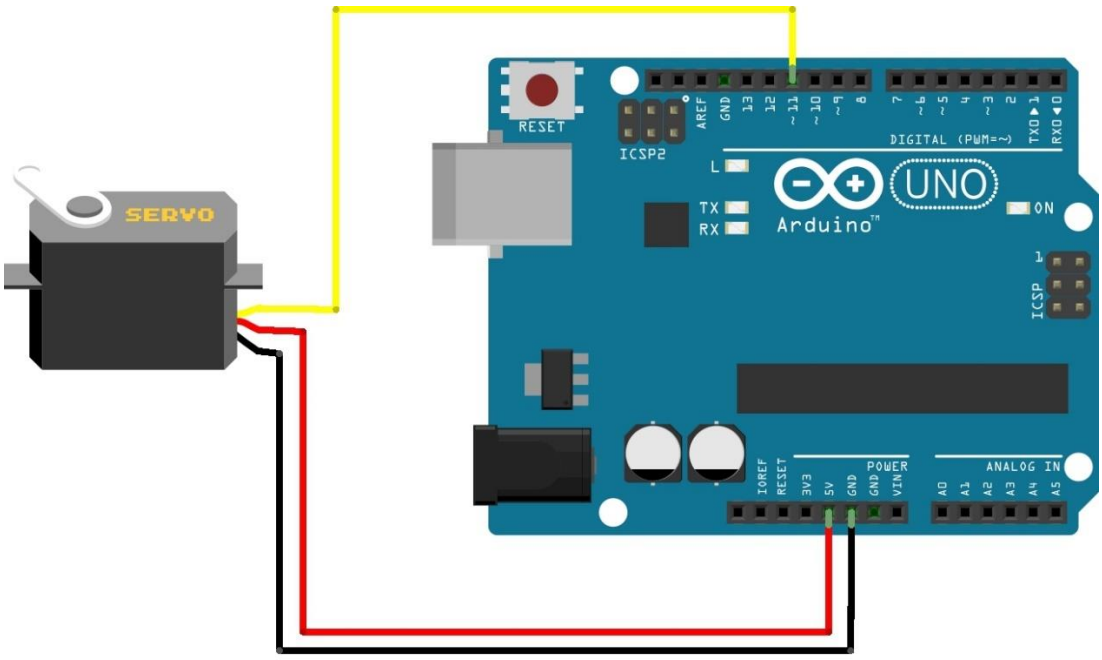


Diagrama de conexión del controlador (Arduino o FPGA), el controlador de motor y el motor a pasos:



Código Arduino - Control de Movimiento con Código:

```
/*29.1.-Control de movimiento de un servomotor con solamente código.
Servomotor SG90 que se alimenta con 4.8 a 6 V, cuenta con un pequeño torque de 1.2 [kg.cm]
cuando se alimenta con 4.8 V y 1.8 [kg.cm] cuando se alimenta con 6 V, su posición es controlada
por una señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).
-------------------------------------------------------------------------------------------
Servomotor MG995 que se alimenta con 4.8 a 7.5 V, cuenta con un torque medio de 8.5 [kg.cm] cuando
se alimenta con 4.8 V y 10 [kg.cm] cuando se alimenta con 7.5 V, su posición es controlada por una
señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).*/
//IMPORTACIÓN DE LIBRERÍAS
#include <Servo.h> //Librería de control de servomotores.

//DECLARACIÓN DE VARIABLES Y/O CONSTANTES
/*Objeto de la clase Servo, este no recibe ningún parámetro, solamente sirve para poder usar los
métodos de la librería.*/
Servo servomotor;
int PINPWMSERVO = 2; //Pin 2 = Cable de señal PWM conectada al PIN2 digital del Arduino.
/*El motor utiliza corriente cuando se mueve, aumenta mucho la corriente que consume cuando se
mueve rápido y consume muy poca cuando mantiene su posición, hay que tomar mucho en cuenta esto
cuando se prueba el motor, ya que, si se usa un valor bajo en el delay, la corriente que demanda el
servomotor puede dañar el puerto serial de la computadora.
-------------------------------------------------------------------------------------------
Servomotor SG90 (El chiquito azul): Si el delay de la velocidad baja de ser 350 milisegundos, el
servomotor no es que demande más corriente y el puerto serial comienza a fallar, sino que no le da
tiempo de llegar al otro punto de la secuencia, más que nada si queremos que vaya de 0 a 180°, si
es un gir que vaya a una posición menor, tal vez si se podría, pero al transcurrir mucho tiempo
este se comenzará a calentar.
-------------------------------------------------------------------------------------------
Servomotor MG995 (El grande negro): Si el delay de la velocidad baja de ser 860 milisegundos, el
servomotor demanda mucha corriente si se conecta directamente y el puerto serial comienza a fallar.*/
int velocidadMotor = 900;

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*servomotor.attach(): Método que sirve para inicializar el objeto del servomotor:
    - El primer parámetro indica el Pin del Arduino al que se conectó el cable de la señal PWM.*/
  servomotor.attach(PINPWMSERVO);
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*servomotor.write(0): Método usado para controlar la posición del eje perteneciente a un servomotor.
  Recibe como argumento un valor numérico que representa el ángulo de giro deseado.*/
  servomotor.write(0);
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos,
  pero cuando se usa después del método Servo.write(), lo que hace es dictar el tiempo que se debe
  tardar de llegar de un punto a otro en una secuencia.*/
  delay(velocidadMotor);
  servomotor.write(180);
  delay(velocidadMotor);
}
```

Código Arduino - Control de Movimiento con Código y Duty Cycle Personalizado:

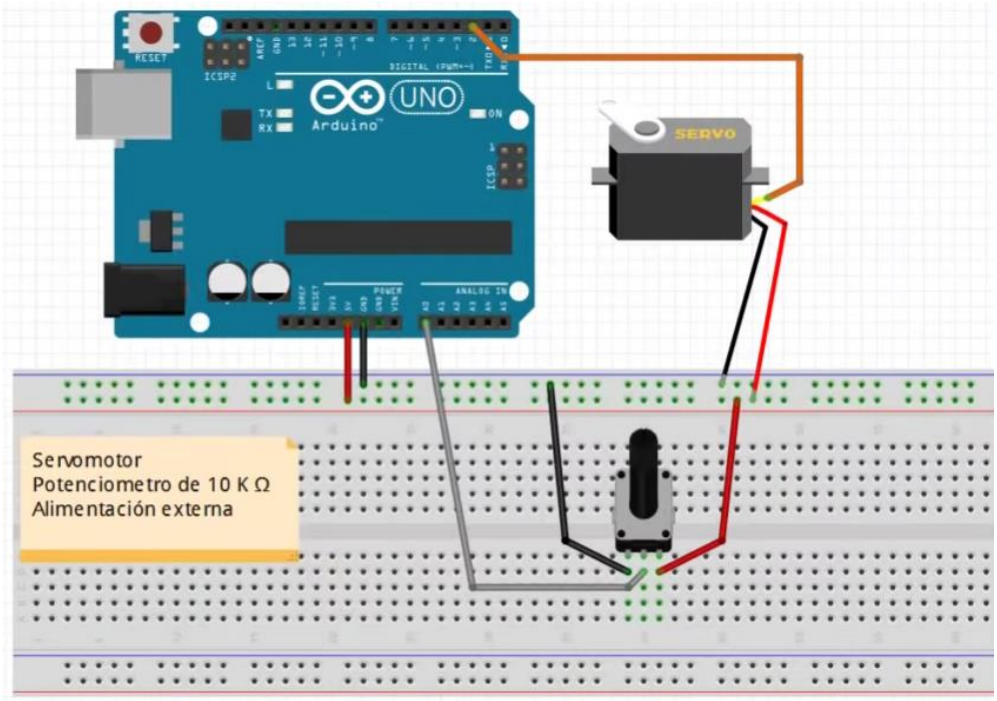
Utilizando el código de Arduino mostrado a continuación se podrá editar el porcentaje del ciclo en alto de los servomotores utilizados, en este caso al experimentar se encontró lo siguiente:

- **Micro Servomotor SG90:** Su ciclo alto para que el motor se sitúe en la posición 0° es de 0.6 ms, no de 1 ms, como lo dice en el datasheet y su ciclo alto debe durar 2.5 ms para que se sitúe en la posición de 180°.
- **Servomotor MG995:** Su ciclo alto para que el motor se sitúe en la posición 0° es de 0.66 ms, no de 1 ms, como lo dice en el datasheet y su ciclo alto debe durar 2.5 ms para que se sitúe en la posición de 160°.

Editando esto, se puede lograr que el movimiento llegue a una posición un poco mayor o menor y es necesario porque con los valores de 1000 µs y 2000 µs ambos motores solo llegaban a girar de 0 a 90°. Si en el motor suena un clic o vibra al llegar a una posición, esto significa que se ha alcanzado su tope.

```
/*29.2.-Control de movimiento de un servomotor con solamente código, editando el duty cycle mínimo
y máximo que se alcanza en los motores.
Servomotor SG90 que se alimenta con 4.8 a 6 V, cuenta con un pequeño torque de 1.2 [kg.cm]
cuando se alimenta con 4.8 V y 1.8 [kg.cm] cuando se alimenta con 6 V, su posición es controlada
por una señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).
-----------------------------------------------------------------------------------------------
Servomotor MG995 que se alimenta con 4.8 a 7.5 V, cuenta con un torque medio de 8.5 [kg.cm] cuando
se alimenta con 4.8 V y 10 [kg.cm] cuando se alimenta con 7.5 V, su posición es controlada por una
señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).*/
//IMPORTACIÓN DE LIBRERÍAS
#include <Servo.h> //Librería de control de servomotores.

//DECLARACIÓN DE VARIABLES Y/O CONSTANTES
/*Objeto de la clase Servo, este no recibe ningún parámetro, solamente sirve para poder usar los
métodos de la librería.*/
Servo servomotor;
int PINPWMSERVO = 11;      //Pin 11 = Cable de señal PWM conectada al PIN11~ digital del Arduino.
/*Cuando el valor alto del duty cycle en la señal PWM dura 1000 µs = 1 ms, la posición del motor
está en 0°. Esta es la teoría, pero si el motor no da el giro completo o suena un clic cuando lo
hace, se puede mover este valor para encontrar el óptimo, además al editar esto se puede cambiar
aunque sea un poco el alcance de giro de los motores.*/
int DUTYCYCLEMIN = 600;  //High Duty Cycle Mínimo ≈ 1000 µs = 1 ms
/*Cuando el valor alto del duty cycle en la señal PWM dura 2000 µs = 2 ms, la posición del motor
está en 180°. Esta es la teoría, pero si el motor no da el giro completo o suena un clic cuando lo
hace, se puede mover este valor para encontrar el óptimo, además al editar esto se puede cambiar
aunque sea un poco el alcance de giro de los motores.*/
int DUTYCYCLEMAX = 2500;  //High Duty Cycle Máximo ≈ 2000 µs = 2 ms
//VALORES ÓPTIMOS DE DUTY CYCLE:
//Servomotor SG90 (El chiquito azul); DUTYCYCLEMIN = 600; DUTYCYCLEMAX = 2500; Giro de 0 a 180°.
//Servomotor MG995 (El grande negro); DUTYCYCLEMIN = 650; DUTYCYCLEMAX = 2500; Giro de 0 a 160°.
/*El motor utiliza corriente cuando se mueve, aumenta mucho la corriente que consume cuando se
mueve rápido y consume muy poca cuando mantiene su posición, hay que tomar mucho en cuenta esto
cuando se prueba el motor, ya que, si se usa un valor bajo en el delay, la corriente que demanda el
servomotor puede dañar el puerto serial de la computadora.
-----------------------------------------------------------------------------------------------
Servomotor SG90 (El chiquito azul): Si el delay de la velocidad baja de ser 350 milisegundos, el
servomotor no es que demande más corriente y el puerto serial comienza a fallar, sino que no le da
tiempo de llegar al otro punto de la secuencia, más que nada si queremos que vaya de 0 a 180°, si
es un gir que vaya a una posición menor, tal vez si se podría, pero al transcurrir mucho tiempo
este se comenzará a calentar.
-----------------------------------------------------------------------------------------------
Servomotor MG995 (El grande negro): Si el delay de la velocidad baja de ser 860 milisegundos, el
servomotor demanda mucha corriente si se conecta directamente y el puerto serial comienza a fallar.*/
int velocidadMotor = 900;

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
void setup() {
  /*servomotor.attach(): Método que sirve para inicializar el objeto del servomotor:
    - El primer parámetro indica el Pin del Arduino al que se conectó el cable de la señal PWM.*/
  servomotor.attach(PINPWMSERVO, DUTYCYCLEMIN, DUTYCYCLEMAX);
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*servomotor.write(0): Método usado para controlar la posición del eje perteneciente a un servomotor.
  Recibe como argumento un valor numérico que representa el ángulo de giro deseado.*/
  servomotor.write(0);
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos,
  pero cuando se usa después del método Servo.write(), lo que hace es dictar el tiempo que se debe
  tardar de llegar de un punto a otro en una secuencia.*/
  delay(velocidadMotor);
  servomotor.write(180);
  delay(velocidadMotor);
}
```

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



Código Arduino - Control de Movimiento con Potenciómetro:

```
/*29.3.-Control de movimiento de un servomotor por medio de un potenciómetro, editando además el
duty cycle mínimo y máximo que se alcanza en los motores.
Servomotor SG90 que se alimenta con 4.8 a 6 V, cuenta con un pequeño torque de 1.2 [kg.cm]
cuando se alimenta con 4.8 V y 1.8 [kg.cm] cuando se alimenta con 6 V, su posición es controlada
por una señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).
-------------------------------------------------------------------------------------------------
Servomotor MG995 que se alimenta con 4.8 a 7.5 V, cuenta con un torque medio de 8.5 [kg.cm] cuando
se alimenta con 4.8 V y 10 [kg.cm] cuando se alimenta con 7.5 V, su posición es controlada por una
señal PWM (Pulse Width Module) con una frecuencia de 50 Hz (20 [ms]).*/
//IMPORTACIÓN DE LIBRERÍAS
#include <Servo.h> //Librería de control de servomotores.

//DECLARACIÓN DE VARIABLES Y/O CONSTANTES
/*Objeto de la clase Servo, este no recibe ningún parámetro, solamente sirve para poder usar los
métodos de la librería.*/
Servo servomotor;
int PINPWMSERVO = 2;     //Pin 2 = Cable de señal PWM conectada al PIN2 digital del Arduino.
/*Cuando el valor alto del duty cycle en la señal PWM dura 1000 µs = 1 ms, la posición del motor
está en 0°. Esta es la teoría, pero si el motor no da el giro completo o suena un clic cuando lo
hace, se puede mover este valor para encontrar el óptimo, además al editar esto se puede cambiar
aunque sea un poco el alcance de giro de los motores.*/
int DUTYCYCLEMIN = 600;  //High Duty Cycle Mínimo ≈ 1000 µs = 1 ms
/*Cuando el valor alto del duty cycle en la señal PWM dura 2000 µs = 2 ms, la posición del motor
está en 180°. Esta es la teoría, pero si el motor no da el giro completo o suena un clic cuando lo
hace, se puede mover este valor para encontrar el óptimo, además al editar esto se puede cambiar
aunque sea un poco el alcance de giro de los motores.*/
int DUTYCYCLEMAX = 2500; //High Duty Cycle Máximo ≈ 2000 µs = 2 ms
//VALORES ÓPTIMOS DE DUTY CYCLE:
//Servomotor SG90 (El chiquito azul); DUTYCYCLEMIN = 600; DUTYCYCLEMAX = 2500; Giro de 0 a 180°.
//Servomotor MG995 (El grande negro); DUTYCYCLEMIN = 650; DUTYCYCLEMAX = 2500; Giro de 0 a 160°.
/*El motor utiliza corriente cuando se mueve, aumenta mucho la corriente que consume cuando se
mueve rápido y consume muy poca cuando mantiene su posición, hay que tomar mucho en cuenta esto
cuando se prueba el motor, ya que, si se usa un valor bajo en el delay, la corriente que demanda el
servomotor puede dañar el puerto serial de la computadora.
-------------------------------------------------------------------------------------------------
Servomotor SG90 (El chiquito azul): Si el delay de la velocidad baja de ser 350 milisegundos, el
servomotor no es que demande más corriente y el puerto serial comienza a fallar, sino que no le da
tiempo de llegar al otro punto de la secuencia, más que nada si queremos que vaya de 0 a 180°, si
es un gir que vaya a una posición menor, tal vez si se podría, pero al transcurrir mucho tiempo
este se comenzará a calentar.
-------------------------------------------------------------------------------------------------
Servomotor MG995 (El grande negro): Si el delay de la velocidad baja de ser 860 milisegundos, el
servomotor demanda mucha corriente si se conecta directamente y el puerto serial comienza a fallar.*/
int velocidadMotor = 900;
int PINPOT = A0;         //Pin A0 = Cable de la terminal central del pot (patita de en medio).

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL
```

```
void setup() {
  /*servomotor.attach(): Método que sirve para inicializar el objeto del servomotor:
    - El primer parámetro indica el Pin del Arduino al que se conectó el cable de la señal PWM.*/
  servomotor.attach(PINPWMSERVO, DUTYCYCLEMIN, DUTYCYCLEMAX);
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*analogRead(): Método que lee un valor analógico por medio del ADC del Arduino que consta de 10
  bits, por lo que convertirá los valores de tensión de 0 a 5V que reciba a un equivalente de 0 a
  2^10 - 1 = 1023.*/
  int LECTURAPOT = analogRead(PINPOT);
  /*map(): Método que realiza una regla de 3, convirtiendo un número de un rango a otro que sea
  equivalente al primero, en sus parámetros recibe lo siguiente:
    - Primer parámetro: Recibe la variable de donde proviene el número a convertir.
    - Segundo parámetro: Recibe el valor mínimo del primer rango de valores.
    - Tercer parámetro: Recibe el valor máximo del primer rango de valores.
    - Cuarto parámetro: Recibe el valor mínimo del segundo rango de valores, al que queremos llegar.
    - Quinto parámetro: Recibe el valor máximo del segundo rango de valores, al que queremos llegar.
  De esta manera se convierten los valores de de 0 a 1023 entregados por el método analogRead() a
  valores de ángulos de rotación de 0 a 180° que comprenda el servomotor.*/
  int ANGULO = map(LECTURAPOT, 0, 1023, 0, 180);
  /*Servo.write(0): Método usado para controlar la posición del eje perteneciente a un servomotor.
  Recibe como argumento un valor numérico que representa el ángulo de giro deseado.*/
  servomotor.write(ANGULO);
}
```

Control de Servomotor en Verilog y VHDL

Se realizarán 2 programas individuales de Verilog y VHDL donde se controlará el circuito primero con solamente código y después se usarán los switches de la tarjeta de desarrollo Nexys 2 para indicarle al servomotor hasta qué ángulo debe llegar. Es importante mencionar que con estos programas solamente se puede controlar el micro servo **SG90**, ya que lo que entrega de corriente y tensión la Nexys 2 no es suficiente para alimentar el servomotor **MG995**:

- 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 **pines** 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 **pines** es de **20mA**.

El cable de la **señal PWM** será conectado directamente al puerto **JA1** de la Nexys 2 y la alimentación del servomotor será conectada a los puertos **+JA6 o +JA12** y **-JA5 o -JA11**.

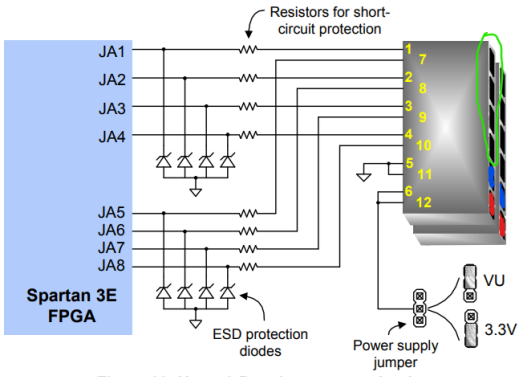


Figure 23: Nexys2 Pmod connector circuits

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[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:; P18	JD10: J14[4]

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3

Código Verilog – Control de Movimiento con Código y Duty Cycle Personalizado:

Divisor de Reloj y Creación de Señal PWM:

```
//SENAL PWM CON FRECUENCIA DE 50HZ PARA CONTROL DE SERVOMOTOR:
//Este proceso sirve para primero crear un divisor de reloj, donde se puede dictar al reloj en que frecuencia
//quiero que opere el servomotor, indicando así su velocidad de rotación, en este mismo proceso del divisor de reloj
//se crea un contador que creara la frecuencia de 50 Hz para la señal PWM, después lo que se hace es usar ese reloj
//modificado para crear otro contador que declare los pasos del estado en alto de la señal PWM, que durara de 1 a 2 ms
//para así mover el servomotor hacia varias posiciones donde debe parar durante una secuencia, que abarca posiciones de
//0 a 180 grados.

module divisorDeRelojPWM(
    input relojNexys2,        //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input rst,                //Botón de reset.
    //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg PWM            //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 [25:0]divisorDeReloj;                        //Vector con el que se obtiene el divisor de reloj.
        //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 tomar del vector, para asignársela al contador que dicta la velocidad del
        //servomotor cuando va de una posición a otra.
        integer conteoFrecuenciaPWM = 1;        //Integer para el contador de uno a 1,000,000.
        //Variable que obtiene un conteo para obtener la señal de 50 Hz que manda pulsos al servomotor con el fin de que
        //llegue a posiciones de 0 a 180 grados, esta tiene un periodo de T = 1/50 = 0.02s = 2ms, aunque después de mandar
        //estos pulsos, puede tener un tiempo de espera antes de mandar el otro y eso es lo que dicta la velocidad de rotación
        //que lleva al llegar de una posición a otra.
        //El valor del conteo se obtiene al dividir el periodo de 20ms sobre el periodo de la señal de 50MHz proveniente del
        //reloj de la Nexys 2: T50Mz = 1/50,000,000 = 20e-9s = 20ns. Por lo tanto, al realizar la operación se obtiene que:
        //T50Hz = 0.02s = 2ms; T50Mz = 20e-9s = 20ns; ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000.
        integer selectorDutyCycle = 1;          //Integer para el selector de posiciones de la señal PWM.
        //La signal selectorDutyCycle cuenta de 0 a 2 porque crea una secuencia de 3 posiciones para el servomotor, que en este
        //caso lo posiciona en 0, 90 y 180 grados consecutivamente, haciendo avanzar el estado en alto de la señal PWM de 1ms
        //a 1.5ms y finalmente a 2ms.
        integer DutyCycle_1a2ms;                //Integer para crear el pulso en alto para la secuencia de 1 a 2ms.
        //Guarda el valor del contador selectorDutyCycle que creara una secuencia de posiciones y lo multiplica por cierto
        //número de pasos, que en teoría son 50,000 para alcanzar un máximo de 1ms en el estado alto del duty cycle. El numero
        //de pasos necesario para la multiplicación se obtiene al dividir el tiempo de duración del pulso en alto entre el
        //periodo de la señal de 50MHz obtenida del reloj de la Nexys 2, la operación es la siguiente:
        //T1ms = 1ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleEnAlto = T1ms /T50Mz = 1e-3/2e-9 = 50,000.

        //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@(), ademas 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(rst))
        begin : clkdiv //El nombre del always@() es clkdiv.
                if(rst) begin
                        //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 = 26'b00000000000000000000000000;
                end else if(conteoFrecuenciaPWM > 1000000) begin
                        conteoFrecuenciaPWM = 1;
                end else begin
                        divisorDeReloj = divisorDeReloj + 1;            //Esto crea al divisor de reloj.
                        conteoFrecuenciaPWM = conteoFrecuenciaPWM + 1; //Esto crea la frecuencia de 50 Hz para la señal PWM.
                end
        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 y asignara cierta velocidad de rotación al servomotor cuando se dirige de una posición a otra.
        //El servomotor SG90 (chiquito azul) puede recibir una frecuencia mínima de 0.4 Hz y máxima de 2.8Hz, esto nos da
        //la posibilidad de elegir de la coordenada 24 (1.49 Hz) a la coordenada 26 (0.3725 Hz) que ya no está incluida en
        //la tabla del divisor de reloj. La frecuencia mínima no esta tan definida, puede ser la que sea, aquí está limitada
        //por la tabla del divisor.
        always@(posedge(divisorDeReloj[24]))
        begin : highDutyCyclePWM      //El nombre del always@() es highDutyCyclePWM.
                //La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de subida
                //en la señal del divisorDeReloj en su coordenada 24, que proporciona una frecuencia de 1.49Hz.
                if(selectorDutyCycle == 2) begin
                        //Cuando el conteo del duty cycle en estado activo (1 lógico), que se da con una frecuencia de
                        //1.49Hz, valga 2, se reiniciará el conteo, regresando el selector y servomotor a su posición
                        //inicial, ya que en este caso se eligen los tiempos de 0, 0.5 y 1 ms, correspondientes a las
                        //posiciones 0, 90 y 180 grados.
                        selectorDutyCycle = 0;
                end else begin      //Esto se ejecutará cuando no se cumpla la condición anterior.
                        //Crea el conteo del estado en alto de la señal PWM con frecuencia de 50 Hz para que alcance las
                        //duraciones de 1 a 2ms, paulatinamente, creando una secuencia de posiciones que va de 0 a 90
                        //grados, luego de 90 a 180 y regresa después a la de 0 grados.
                        selectorDutyCycle = selectorDutyCycle + 1;
```

```
                end
                //Con la variable DutyCycle_1a2ms se realiza el avance del pulso en la señal PWM de 1 a 2ms de forma
                //paulatina, creando la secuencia de posiciones, esto se hace multiplicando el número del selector por un
                //número de pasos, para ello se toma en cuenta que el número de pasos necesario para la multiplicación se
                //obtiene al dividir el tiempo de duración del pulso en alto entre el periodo de la señal de 50MHz obtenida
                //del reloj de la Nexys 2, la operación es la siguiente:
                //T1ms = 1ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleEnAlto = T1ms /T50Mz = 1e-3/2e-9 = 50,000.
                //Esta es la teoría, pero los pasos del servomotor no siempre siguen esto al pie de la letra, por lo que se
                //debe calibrar el servomotor, viendo por cada número de pasos, la posición que alcanza.
                //AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
                //El servomotor SG90 (chiquito azul) realiza un giro de 0 a 180 grados con un paso de 92,109.
                //El servomotor SG90 (chiquito azul) realiza un giro de 0, 90 y 180 grados con un paso de 41,868.
                //Al realizar la operación de 92109/2 = 46054, podemos ver que las posiciones no son completamente lineales, o
                //al menos no tienen una pendiente de m = 1.
                //Este es un código de ejemplo, pero si en verdad se quisiera hacer una secuencia con un control super preciso
                //Se tendría que ver el paso exacto que se debe dar para cada posición y luego crear un bucle case que lleve a
                //cada posición dicha secuencia.
                DutyCycle_1a2ms = 44775 * selectorDutyCycle ; //25,000*0 = 0ms; 25,000*1 = 0.5ms; 25,000*2 = 50,000 = 1ms;
                //Se debe tomar en cuenta que, al realizar diferentes secuencias, se debe cambiar de igual manera el número de
                //pasos, dividiendo el número de pasos que llega de 0 a 180 grados, osea 89,550, entre el número de acciones
                //de la secuencia, considerando además que, al terminar la secuencia, el motor vuelve a su posición inicial de
                //0 grados.
        end

        //Para mantener el estado alto de la señal PWM de 1 a 2 ms con el fin de mover el servomotor de 0 a 180 grados, se
        //debe realizar un conteo como el que se realizó para crear la frecuencia de 50 Hz en la señal PWM, para ello se
        //realiza el siguiente calculo:
        //T1ms = 1ms;       T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1ms/20ns   = 1e-3/20e-9   = 50,000;  0 grados servo.
        //T1ms = 1.5ms;     T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1.5ms/20ns = 1.5e-3/20e-9 = 75,000;  90 grados servo.
        //T1ms = 2ms;       T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 2ms/20ns   = 2e-3/20e-9   = 100,000; 180 grados servo.
        //Con este cálculo podemos saber que, para que el pulso en alto abarque un rango inicial de 1ms se debe usar 50e3
        //pasos, a este ancho inicial de pulso se le debe sumar el valor de la signal DutyCycle_1a2ms para que el pulso
        //paulatinamente vaya aumentando de 1 a 2ms, creando así la secuencia que va a las posiciones 0, 90 y 180 grados.
        //Pero recordemos que esta es la teoría, para alcanzar dichas posiciones se debe realizar una calibración, donde
        //veremos el número de pasos reales para alcanzar cada posición.
        //AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
        //El servomotor SG90 alcanza un rango de 1ms con un paso inicial de 26e3.
        always@(conteoFrecuenciaPWM, DutyCycle_1a2ms)
        begin : pwmSignal //El nombre del always@() es pwmSignal.
                if(conteoFrecuenciaPWM <= (26000 + DutyCycle_1a2ms)) begin
                        PWM = 1'b1;
                end else begin
                        PWM = 1'b0;
                end
        end
endmodule
```

Código UCF:

```
//ENTRADAS DEL MODULO divisorDeReloj:
net "relojNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "rst" loc = "H13";//El botón de Reset está conectado al push button BTN3.

//CABLE DE SENAL PWM DEL SERVOMOTOR CONECTADO A LOS PUERTOS JA:
net "PWM" loc = "L15";//La señal PWM del servomotor está conectada al puerto JA1.
```

Código VHDL – Control de Movimiento con Código y Duty Cycle Personalizado:

Divisor de Reloj y Creación de Señal PWM:

```
--SENAL PWM CON FRECUENCIA DE 50HZ PARA CONTROL DE SERVOMOTOR:
--Este proceso sirve para primero crear un divisor de reloj, donde se puede dictar al reloj en que frecuencia
--quiero que opere el servomotor, indicando así su velocidad de rotación, en este mismo proceso del divisor de reloj
--se crea un contador que creara la frecuencia de 50 Hz para la señal PWM, después lo que se hace es usar ese reloj
--modificado para crear otro contador que declare los pasos del estado en alto de la señal PWM, que durara de 1 a 2 ms
--para así mover el servomotor hacia varias posiciones donde debe parar durante una secuencia, que abarca posiciones de
--0 a 180 grados.

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 divisorDeRelojPWM is
    Port ( relojNexys2 : in  STD_LOGIC;          --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
           rst : in  STD_LOGIC;                  --Botón de reset.
           PWM : out  STD_LOGIC);                --Señal PWM.
end divisorDeRelojPWM;

--Arquitectura: Aquí se hará el divisor de reloj y la señal PWM haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeRelojPWM 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 (25 downto 0);    --Vector con el que se obtiene el divisor de reloj.
--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 al contador que dicta la velocidad del
--servomotor cuando va de una posicion a otra.
signal conteoFrecuenciaPWM: integer range 1 to 1e6 := 1; --Signal para el contador de uno a 1,000,000.
--Variable que obtiene un conteo para obtener la señal de 50 Hz que manda pulsos al servomotor con el fin de que
--llegue a posiciones de 0 a 180 grados, esta tiene un periodo de T = 1/50 = 0.02s = 2ms, aunque después de mandar
--estos pulsos, puede tener un tiempo de espera antes de mandar el otro y eso es lo que dicta la velocidad de rotación
--que lleva al llegar de una posición a otra.
--El valor del conteo se obtiene al dividir el periodo de 20ms sobre el periodo de la señal de 50MHz proveniente del
--reloj de la Nexys 2: T50Mz = 1/50,000,000 = 20e-9s = 20ns. Por lo tanto, al realizar la operación se obtiene que:
--T50Hz = 0.02s = 2ms; T50Mz = 20e-9s = 20ns; ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000.
signal selectorDutyCycle: integer range 0 to 2 := 0; --Signal para el selector de posiciones de la señal PWM.
--La signal selectorDutyCycle cuenta de 0 a 2 porque crea una secuencia de 3 posiciones para el servomotor, que en este
--caso lo posiciona en 0, 90 y 180 grados consecutivamente, haciendo avanzar el estado en alto de la senal PWM de 1ms
--a 1.5ms y finalmente a 2ms.
signal DutyCycle_1a2ms: integer;                    --Crea el pulso en alto para la secuencia de 1 a 2ms.
--Guarda el valor del contador selectorDutyCycle que creara una secuencia de posiciones y lo multiplica por cierto
--numero de pasos, que en teoría son 50,000 para alcanzar un máximo de 1ms en el estado alto del duty cycle. El numero
--de pasos necesario para la multiplicación se obtiene al dividir el tiempo de duración del pulso en alto entre el
--periodo de la señal de 50MHz obtenida del reloj de la Nexys 2, la operación es la siguiente:
--T1ms = 1ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleEnAlto = T1ms /T50Mz = 1e-3/2e-9 = 50,000.

begin
        --A los process se les puede dar un nombre diferente para diferenciarlos entre sí, se hace esto porque hay
        --programas de VHDL donde se usan varios process y de esta forma es más fácil no confundir uno con otro, la sintaxis
        --es la siguiente:
        --nombreProcess : process(Entradas o signals que se usan en el bucle, condicional u operación matemática) begin
        --                 Contenido del process.
        --end process;
        clkdiv: process(relojNexys2, rst) begin
                if(rst = '1') then --Cuando el botón Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciara.
                        divisorDeReloj <= "00000000000000000000000000";
                elsif(conteoFrecuenciaPWM > 1e6) then
                        --Se reinicia el conteo para obtener la frecuencia de 50 Hz de la señal PWM cuando se llegue al tope
                        --calculado anteriormente, donde:
                        --ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000 = 1e6.
                        conteoFrecuenciaPWM <= 1;
                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.
                        conteoFrecuenciaPWM <= conteoFrecuenciaPWM + 1;--Esto crea la frecuencia de 50 Hz para la señal PWM.
                end if;
        end process clkdiv;

        --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 y asignara cierta velocidad de rotación al servomotor cuando se dirige de una posición a otra.
        --El servomotor SG90 (chiquito azul) puede recibir una frecuencia mínima de 0.4 Hz y máxima de 2.8Hz, esto nos da
        --la posibilidad de elegir de la coordenada 24 (1.49 Hz) a la coordenada 26 (0.3725 Hz) que ya no está incluida en
        --la tabla del divisor de reloj. La frecuencia mínima no esta tan definida, puede ser la que sea, aquí está limitada
        --por la tabla del divisor.
        highDutyCyclePWM: process(divisorDeReloj(25)) begin
                if(rising_edge(divisorDeReloj(25))) then
                        --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco de
                        --subida en la señal del divisorDeReloj en su coordenada 24, que proporciona una frecuencia
                        --de 1.49Hz.
                        if(selectorDutyCycle = 2) then
                                --Cuando el conteo del duty cycle en estado activo (1 lógico), que se da con una
                                --frecuencia de 1.49Hz, valga 2, se reiniciará el conteo, regresando el selector y
                                --servomotor a su posición inicial, ya que en este caso se eligen los tiempos de 0, 0.5 y
                                --1 ms, correspondientes a las posiciones 0, 90 y 180 grados.
                                selectorDutyCycle <= 0;
                        else
                                --Crea el conteo del estado en alto de la señal PWM con frecuencia de 50 Hz para que
                                --alcance las duraciones de 1 a 2ms, paulatinamente, creando una secuencia de posiciones
                                --que va de 0 a 90 grados, luego de 90 a 180 y regresa después a la de 0 grados.
                                selectorDutyCycle <= selectorDutyCycle + 1;
                        end if;
                end if;
                --Con la variable DutyCycle_1a2ms se realiza el avance del pulso en la señal PWM de 1 a 2ms de forma
                --paulatina, creando la secuencia de posiciones, esto se hace multiplicando el número del selector por un
                --numero de pasos, para ello se toma en cuenta que el número de pasos necesario para la multiplicación se
                --obtiene al dividir el tiempo de duración del pulso en alto entre el periodo de la señal de 50MHz obtenida
                --del reloj de la Nexys 2, la operación es la siguiente:
                --T1ms = 1ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleEnAlto = T1ms /T50Mz = 1e-3/2e-9 = 50,000.
                --Esta es la teoría, pero los pasos del servomotor no siempre siguen esto al pie de la letra, por lo que se
                --debe calibrar el servomotor, viendo por cada número de pasos, la posición que alcanza.
                --AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
                --El servomotor SG90 (chiquito azul) realiza un giro de 0 a 180 grados con un paso de 92,109.
                --El servomotor SG90 (chiquito azul) realiza un giro de 0, 90 y 180 grados con un paso de 41,868.
                --Al realizar la operación de 92109/2 = 46054, podemos ver que las posiciones no son completamente lineales, o
                --al menos no tienen una pendiente de m = 1.
                --Este es un código de ejemplo, pero si en verdad se quisiera hacer una secuencia con un control super preciso
                --Se tendría que ver el paso exacto que se debe dar para cada posición y luego crear un bucle case que lleve a
                --cada posición dicha secuencia.
                DutyCycle_1a2ms <= 44775 * selectorDutyCycle; --25,000*0 = 0ms; 25,000*1 = 0.5ms; 25,000*2 = 50,000 = 1ms;
                --Se debe tomar en cuenta que, al realizar diferentes secuencias, se debe cambiar de igual manera el número de
                --pasos, dividiendo el número de pasos que llega de 0 a 180 grados, osea 89,550, entre el número de acciones
                --de la secuencia, considerando además que, al terminar la secuencia, el motor vuelve a su posición inicial de
                --0 grados.
        end process highDutyCyclePWM;

        --Para mantener el estado alto de la señal PWM de 1 a 2 ms con el fin de mover el servomotor de 0 a 180 grados, se
        --debe realizar un conteo como el que se realizó para crear la frecuencia de 50 Hz en la señal PWM, para ello se
        --realiza el siguiente calculo:
```

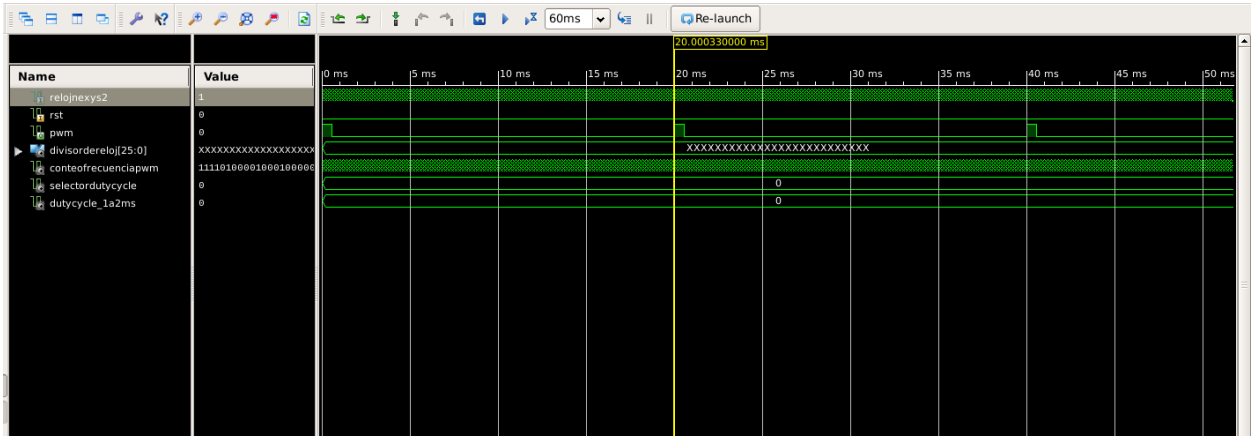
```
        --T1ms = 1ms;        T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1ms/20ns   = 1e-3/20e-9   = 50,000;  0 grados servo.
        --T1ms = 1.5ms;      T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1.5ms/20ns = 1.5e-3/20e-9 = 75,000;  90 grados servo.
        --T1ms = 2ms;        T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 2ms/20ns   = 2e-3/20e-9   = 100,000; 180 grados servo.
        --Con este cálculo podemos saber que para que el pulso en alto abarque un rango inicial de 1ms se debe usar 50e3
        --pasos, a este ancho inicial de pulso se le debe sumar el valor de la signal DutyCycle_1a2ms para que el pulso
        --paulatinamente vaya aumentando de 1 a 2ms, creando así la secuencia que va a las posiciones 0, 90 y 180 grados.
        --Pero recordemos que esta es la teoría, para alcanzar dichas posiciones se debe realizar una calibración, donde
        --veremos el número de pasos reales para alcanzar cada posición.
        --AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
        --El servomotor SG90 alcanza un rango de 1ms con un paso inicial de 26e3.
        pwmSignal: process(conteoFrecuenciaPWM, DutyCycle_1a2ms)begin
                --Este if crea el estado en alto y bajo de la señal PWM que crea la secuencia, yendo de 0 a 90 y 180 grados.
                if(conteoFrecuenciaPWM <= (26e3 + DutyCycle_1a2ms)) then
                        PWM <= '1';
                else
                        PWM <= '0';
                end if;
        end process pwmSignal;
end frecuenciaNueva;
```

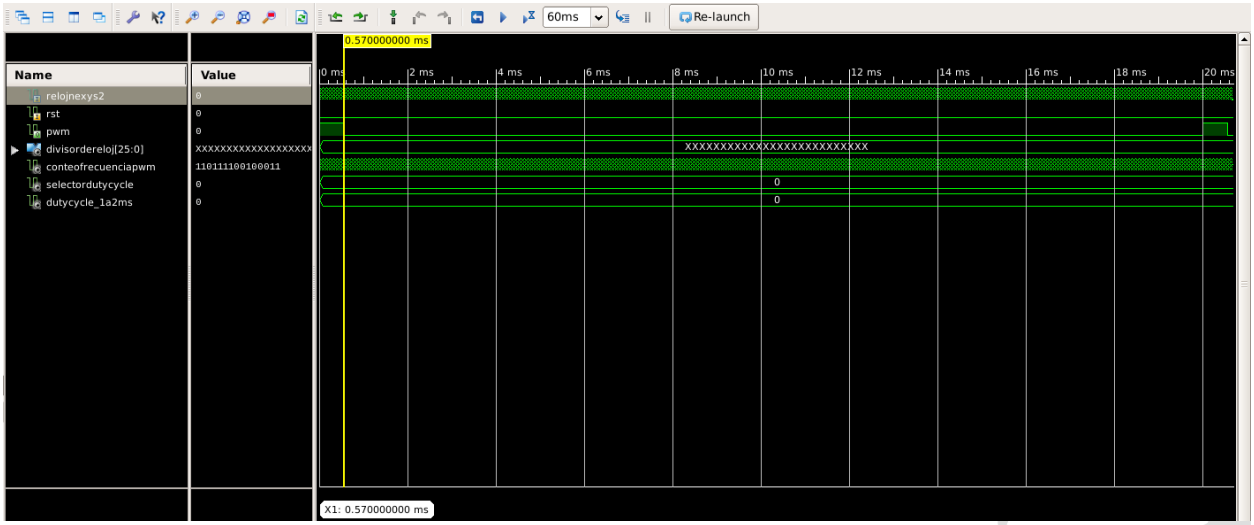
Código UCF:

```
//ENTRADAS DEL MODULO divisorDeReloj:
net "relojNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "rst" loc = "H13";//El botón de Reset está conectado al push button BTN3.

//CABLE DE SENAL PWM DEL SERVOMOTOR CONECTADO A LOS PUERTOS JA:
net "PWM" loc = "L15";//La señal PWM del servomotor está conectada al puerto JA1.
```

Simulación PWM:

En la simulación no es visible el cambio de 1ms a 2ms en la señal PWM, esto se debe a que la acción del divisor de reloj nunca es visible en la simulación del ISE de Xilinx, pero así se muestra que la frecuencia de 50 Hz si se está ejecutando y además que se ve el pulso de la señal PWM.





Código Verilog – Control de Movimiento con Switches:

Divisor de Reloj y Creación de Señal PWM:

```
//SEÑAL PWM CON FRECUENCIA DE 50HZ PARA CONTROL DE SERVOMOTOR:
//Este proceso sirve para primero crear un divisor de reloj, donde se puede dictar al reloj en que frecuencia
//quiero que opere el servomotor, indicando así su velocidad de rotación, en este mismo proceso del divisor de reloj
//se crea un contador que creara la frecuencia de 50 Hz para la señal PWM, después lo que se hace es usar ese reloj
//modificado para monitorear los switches del selector y así seleccionar una posición, a donde el servomotor se ira
//y mantendrá su posición hasta que cense que haya ocurrido un cambio, siempre y cuando el valor del selector sea
//distinto de cero.

module controlSwitchesPWM(
    input relojNexys2,                  //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input rst,                          //Botón de reset.
    input [5:0] selectPos,              //6 switches para elegir una posición de 30, 60, 90, 120, 150 o 180 grados.
    //Las salidas usadas dentro de un condicional o bucle se declaran como reg.
    output reg [5:0] ledAngulo,
    output reg PWM                      //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 [25:0]divisorDeReloj;         //Vector con el que se obtiene el divisor de reloj.
        //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 tomar del vector, para asignársela al contador que dicta la velocidad del
        //servomotor cuando va de una posición a otra.
        integer conteoFrecuenciaPWM = 1; //Integer para el contador de uno a 1,000,000.
        //Variable que obtiene un conteo para obtener la señal de 50 Hz que manda pulsos al servomotor con el fin de que
        //llegue a posiciones de 0 a 180 grados, esta tiene un periodo de T = 1/50 = 0.02s = 2ms, aunque después de mandar
        //estos pulsos, puede tener un tiempo de espera antes de mandar el otro y eso es lo que dicta la velocidad de
        //rotación que lleva al llegar de una posición a otra.
        //El valor del conteo se obtiene al dividir el periodo de 20ms sobre el periodo de la señal de 50MHz proveniente del
        //reloj de la Nexys 2: T50Mz = 1/50,000,000 = 20e-9s = 20ns. Por lo tanto, al realizar la operación se obtiene que:
        //T50Hz = 0.02s = 2ms; T50Mz = 20e-9s = 20ns; ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000.
        //AL realizar la calibración, el valor real de los pasos para alcanzar la posición 0 grados es de = 27000.
        integer posicion0Grados = 26000; //Integer que dicta los pasos para situar el eje del motor en 0 grados.
        //Variable asignada al integer conteoFrecuenciaPWM para situar el estado alto de la señal PWM en 1ms, posicionando al
        //motor en una posicion de 0 grados inicialmente.
        integer movServo = 0;             //Integer que dicta los pasos que debe dar el motor para llegar a una posición.
        //EL valor del integer movServo será modificado dependiendo del valor ingresado por los switches del vector
        //conteoFrecuenciaPWM, para así situar el eje del motor en una posición especifica y dejarlo ahí, para ello la
        //variable selectorDutyCycle solo puede abarcar 1 valor.


        //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 sí 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(rst))
        begin : clkdiv //El nombre del always@() es clkdiv.
                if(rst) begin
                        //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 = 26'b00000000000000000000000000;
                end else if(conteoFrecuenciaPWM > 1000000) begin
                        conteoFrecuenciaPWM = 1;
                end else begin
                        divisorDeReloj = divisorDeReloj + 1;          //Esto crea al divisor de reloj.
                        conteoFrecuenciaPWM = conteoFrecuenciaPWM + 1; //Esto crea la frecuencia de 50 Hz para la señal PWM.
                end
        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 y asignara cierta velocidad de respuesta al control de movimiento por medio de switches.
        always@(posedge(divisorDeReloj[10]))
        begin : highDutyCyclePWM //El nombre del always@() es highDutyCyclePWM.
                //Cuando el valor del selector sea distinto de cero, vera cual es la posición que le corresponde y la asignara
                //a la variable movServo, que moverá el servomotor a la posición de 30, 60, 90, 120, 150 o 180 grados.
                if(selectPos != 6'b000000) begin
                        case(selectPos)//CASE evalúa los diferentes valores de la variable que tenga en su paréntesis
                                //Los switches del selector se levantan para que el integer movimientoServo adopte
                                //diferentes números que muevan el servomotor a todas las posibles posiciones, que son las
                                //siguientes:
                                6'b000001 : begin
                                        movServo = 12329; //Bit menos significativo del selector = 12,329 = 30 grados.
                                        ledAngulo = 6'b000001;
                                end
                                6'b000010 : begin
```

```
                                        movServo = 26419; //Bit 2 del selector = movServo = 26,419 = 60 grados.
                                        ledAngulo = 6'b000010;
                                end
                                6'b000100 : begin
                                        movServo = 41868; //Bit 3 del selector = movServo = 41,868 = 90 grados.
                                        ledAngulo = 6'b000100;
                                end
                                6'b001000 : begin
                                        movServo = 57774; //Bit 4 del selector = movServo = 57,774 = 120 grados.
                                        ledAngulo = 6'b001000;
                                end
                                6'b010000 : begin
                                        movServo = 75633; //Bit 5 del selector = movServo = 75,633 = 150 grados.
                                        ledAngulo = 6'b010000;
                                end
                                6'b100000 : begin
                                        movServo = 92109; //Bit más significativo del selector = 92,109 = 180 grados.
                                        ledAngulo = 6'b100000;
                                end
                                //Si no se ha seleccionado nada o se mete con los switches cualquier otra cosa, el
                                //servomotor se mueve a su posición inicial.

                                default: begin
                                        movServo = 0;
                                        ledAngulo = 6'b000000;
                                end
                        endcase
                end
        end

        //Para mantener el estado alto de la señal PWM de 1 a 2 ms con el fin de mover el servomotor de 0 a 180 grados, se
        //debe realizar un conteo como el que se realizó para crear la frecuencia de 50 Hz en la señal PWM, para ello se
        //realiza el siguiente calculo:
        //T1ms = 1ms;   T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1ms/20ns   = 1e-3/20e-9   = 50,000;  0 grados servo.
        //T1ms = 1.5ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1.5ms/20ns = 1.5e-3/20e-9 = 75,000;  90 grados servo.
        //T1ms = 2ms;   T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 2ms/20ns   = 2e-3/20e-9   = 100,000; 180 grados servo.
        //Con este cálculo podemos saber que, para que el pulso en alto abarque un rango inicial de 1ms se debe usar 50e3
        //pasos, a este ancho inicial de pulso se le debe sumar el valor de la signal DutyCycle_la2ms para que el pulso
        //paulatinamente vaya aumentando de 1 a 2ms, creando así la secuencia que va a las posiciones 0, 90 y 180 grados.
        //Pero recordemos que esta es la teoría, para alcanzar dichas posiciones se debe realizar una calibración, donde
        //veremos el número de pasos reales para alcanzar cada posición.
        //AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
        //El servomotor SG90 alcanza un rango de 1ms con un paso inicial de 26e3.
        always@(conteoFrecuenciaPWM, posicion0Grados, movServo)
        begin : pwmSignal //El nombre del always@() es pwmSignal.
                if(conteoFrecuenciaPWM <= (posicion0Grados + movServo)) begin
                        PWM = 1'b1;
                end else begin
                        PWM = 1'b0;
                end
        end
endmodule
```

Código UCF:

```
//ENTRADAS DEL MODULO divisorDeReloj:
net "relojNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "rst" loc = "H13";//El boton de Reset esta conectado al push button BTN3.
//Bit mas significativo del selectPos conectado al switch SW5 en el puerto L13.
net "selectPos[5]" loc = "L13";
//Bit 5 del selectPos conectado al switch SW4 en el puerto L14.
net "selectPos[4]" loc = "L14";
//Bit 4 del selectPos conectado al switch SW3 en el puerto K17.
net "selectPos[3]" loc = "K17";
//Bit 3 del selectPos conectado al switch SW2 en el puerto K18.
net "selectPos[2]" loc = "K18";
//Bit 2 del selectPos conectado al switch SW1 en el puerto H18.
net "selectPos[1]" loc = "H18";
//Bit menos significativo del selectPos conectado al switch SW0 en el puerto G18.
net "selectPos[0]" loc = "G18";

//CABLE DE SENAL PWM DEL SERVOMOTOR CONECTADO A LOS PUERTOS JA:
net "PWM" loc = "L15";//La senal PWM del servomotor esta conectada al puerto JA1.
//Bit mas significativo del ledAngulo conectado al led LD5 en el puerto P15.
net "ledAngulo[5]" loc = "P15";
//Bit 5 del ledAngulo conectado al led LD4 en el puerto E17.
net "ledAngulo[4]" loc = "E17";
//Bit 4 del ledAngulo conectado al led LD3 en el puerto K14.
net "ledAngulo[3]" loc = "K14";
//Bit 3 del ledAngulo conectado al led LD2 en el puerto K15.
net "ledAngulo[2]" loc = "K15";
//Bit 2 del ledAngulo conectado al led LD1 en el puerto J15.
net "ledAngulo[1]" loc = "J15";
//Bit menos significativo del ledAngulo conectado al led LD0 en el puerto J14.
net "ledAngulo[0]" loc = "J14";
```

Código VHDL – Control de Movimiento con Switches:

Divisor de Reloj y Creación de Señal PWM:

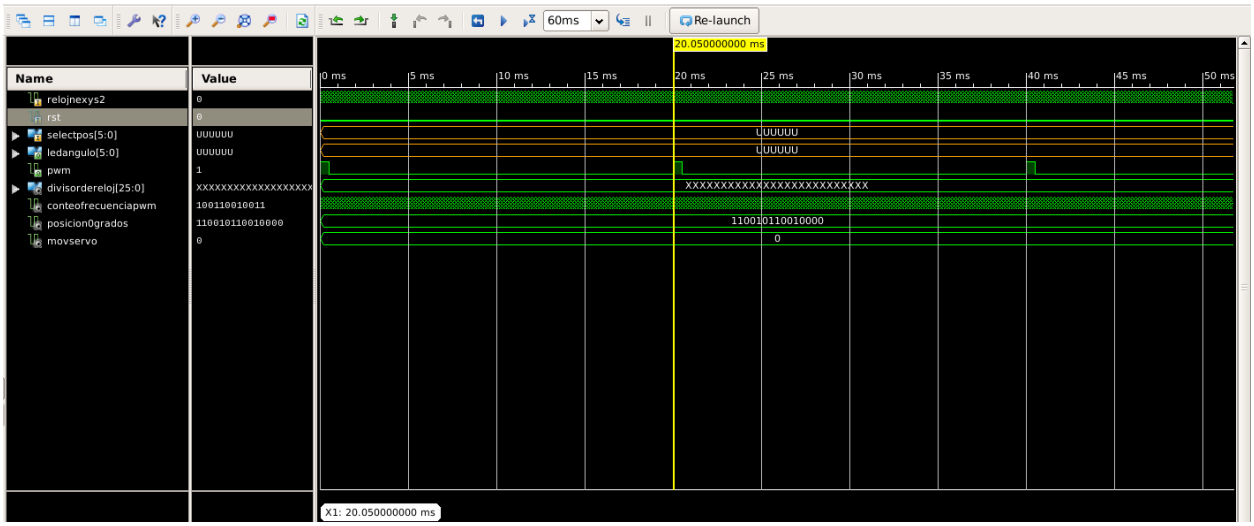
```
--SEÑAL PWM CON FRECUENCIA DE 50HZ PARA CONTROL DE SERVOMOTOR:
--Este proceso sirve para primero crear un divisor de reloj, donde se puede dictar al reloj en que frecuencia
--quiero que opere el servomotor, indicando así su velocidad de rotación, en este mismo proceso del divisor de reloj
--se crea un contador que creara la frecuencia de 50 Hz para la señal PWM, después lo que se hace es usar ese reloj
--modificado para monitorear los switches del selector y así seleccionar una posición, a donde el servomotor se ira
--y mantendrá su posición hasta que cense que haya ocurrido un cambio, siempre y cuando el valor del selector sea
--distinto de cero.

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 controlSwitchesPWM is
    Port ( relojNexys2 : in  STD_LOGIC;--Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
           rst : in  STD_LOGIC;                              --Botón de reset.
           selectPos : in  STD_LOGIC_VECTOR (5 downto 0);  --Selector de posición.
           ledAngulo : out  STD_LOGIC_VECTOR (5 downto 0); --Leds del selector de posición.
           PWM : out  STD_LOGIC);                            --Señal PWM.
end controlSwitchesPWM;

--Arquitectura: Aquí se hará el divisor de reloj y la señal PWM haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of controlSwitchesPWM 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 (25 downto 0);     --Vector con el que se obtiene el divisor de reloj.
--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 al contador que dicta la velocidad del
--servomotor cuando va de una posición a otra.
signal conteoFrecuenciaPWM: integer range 1 to 1e6 := 1; --Signal para el contador de uno a 1,000,000.
--Variable que obtiene un conteo para obtener la señal de 50 Hz que manda pulsos al servomotor con el fin de que
--llegue a posiciones de 0 a 180 grados, esta tiene un periodo de T = 1/50 = 0.02s = 2ms, aunque después de mandar
--estos pulsos, puede tener un tiempo de espera antes de mandar el otro y eso es lo que dicta la velocidad de rotación
--que lleva al llegar de una posición a otra.
--El valor del conteo se obtiene al dividir el periodo de 20ms sobre el periodo de la señal de 50MHz proveniente del
--reloj de la Nexys 2: T50Mz = 1/50,000,000 = 20e-9s = 20ns. Por lo tanto, al realizar la operación se obtiene que:
--T50Hz = 0.02s = 2ms; T50Mz = 20e-9s = 20ns; ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000.
signal posicion0Grados: integer := 26e3;         --Integer que dicta los pasos para situar el eje del motor en 0 grados.
--Variable asignada al integer conteoFrecuenciaPWM para situar el estado alto de la señal PWM en 1ms, posicionando al
--motor en una posición de 0 grados inicialmente.
signal movServo: integer range 0 to 92109 := 0; --Dicta los pasos que debe dar el motor para llegar a una posición.
--EL valor del integer movServo será modificado dependiendo del valor ingresado por los switches del vector
--conteoFrecuenciaPWM, para así situar el eje del motor en una posición especifica y dejarlo ahí, para ello la
--variable selectorDutyCycle solo puede abarcar 1 valor.

begin
        --A los process se les puede dar un nombre diferente para diferenciarlos entre sí, se hace esto porque hay
        --programas de VHDL donde se usan varios process y de esta forma es más fácil no confundir uno con otro, la sintaxis
        --es la siguiente:
        --nombreProcess : process(Entradas o signals que se usan en el bucle, condicional u operación matemática) begin
        --                 Contenido del process.
        --end process;
        clkdiv: process(relojNexys2, rst) begin
                if(rst = '1') then --Cuando el botón Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciara.
                        divisorDeReloj <= "00000000000000000000000000";
                elsif(conteoFrecuenciaPWM > 1e6) then
                        --Se reinicia el conteo para obtener la frecuencia de 50 Hz de la señal PWM cuando se llegue al tope
                        --calculado anteriormente,
                        --donde: ConteoFrecuenciaPWM = T50Hz/T50Mz = 2ms/20ns = 2e-3/2e-9 = 1,000,000 = 1e6.
                        conteoFrecuenciaPWM <= 1;
                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.
                        conteoFrecuenciaPWM <= conteoFrecuenciaPWM + 1; --Crea la frecuencia de 50 Hz para la señal PWM.
                end if;
        end process clkdiv;

        --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 y asignara cierta velocidad de respuesta al control de movimiento por medio de switches.
        highDutyCyclePWM: process(divisorDeReloj(12)) begin
                if(rising_edge(divisorDeReloj(12))) then
                        --Cuando el valor del selector sea distinto de cero, vera cual es la posición que le corresponde y
                        --la asignará a la variable movServo, que moverá el servomotor a la posición de 30, 60, 90, 120, 150
                        --o 180 grados.
                        if(selectPos /= "000000") then
                        --CASE se usa para evaluar los diferentes valores de la variable que tenga en su parentesis
                                case(selectPos) is
                                        --Los switches del selector se levantan para que el integer movimientoServo
                                        --adopte diferentes números que muevan el servomotor a todas las posibles
                                        --posiciones, que son las siguientes:
                                        when "000001" =>
                                                movServo <= 12329;
                                                --Bit menos significativo del selector = 12,329 = 30 grados.
                                                ledAngulo <= "000001";
```

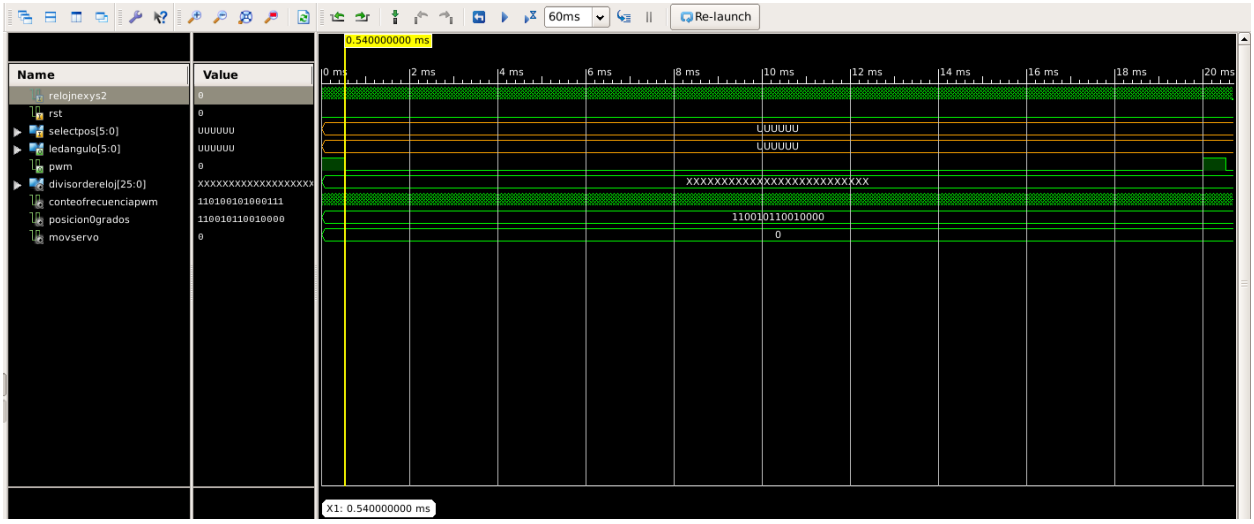
```
                                        when "000010" =>
                                                movServo <= 26419;
                                                --Bit 2 del selector = movServo = 26,419 = 60 grados.
                                                ledAngulo <= "000010";
                                        when "000100" =>
                                                movServo <= 41868;
                                                --Bit 3 del selector = movServo = 41,868 = 90 grados.
                                                ledAngulo <= "000100";
                                        when "001000" =>
                                                movServo <= 57774;
                                                --Bit 4 del selector = movServo = 57,774 = 120 grados.
                                                ledAngulo <= "001000";
                                        when "010000" =>
                                                movServo <= 75633;
                                                --Bit 5 del selector = movServo = 75,633 = 150 grados.
                                                ledAngulo <= "010000";
                                        when "100000" =>
                                                movServo <= 92109;
                                                --Bit mas significativo del selector = 92,109 = 180 grados.
                                                ledAngulo <= "100000";
                                        --Si no se ha seleccionado nada o se mete con los switches cualquier otra cosa,
                                        --el servomotor se mueve a su posición inicial.
                                        when others =>
                                                movServo <= 0;
                                                ledAngulo <= "000000";
                                end case;
                        end if;
                end if;
        end process highDutyCyclePWM;

        --Para mantener el estado alto de la señal PWM de 1 a 2 ms con el fin de mover el servomotor de 0 a 180 grados, se
        --debe realizar un conteo como el que se realizó para crear la frecuencia de 50 Hz en la señal PWM, para ello se
        --realiza el siguiente calculo:
        --T1ms = 1ms;   T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1ms/20ns   = 1e-3/20e-9   = 50,000;  0 grados servo.
        --T1ms = 1.5ms; T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 1.5ms/20ns = 1.5e-3/20e-9 = 75,000;  90 grados servo.
        --T1ms = 2ms;   T50Mz = 20e-9s = 20ns; ConteoDutyCycleAlto = 2ms/20ns   = 2e-3/20e-9   = 100,000; 180 grados servo.
        --Con este cálculo podemos saber que para que el pulso en alto abarque un rango inicial de 1ms se debe usar 50e3
        --pasos, a este ancho inicial de pulso se le debe sumar el valor de la signal DutyCycle_la2ms para que el pulso
        --paulatinamente vaya aumentando de 1 a 2ms, creando así la secuencia que va a las posiciones 0, 90 y 180 grados.
        --Pero recordemos que esta es la teoría, para alcanzar dichas posiciones se debe realizar una calibración, donde
        --veremos el número de pasos reales para alcanzar cada posición.
        --AL REALIZAR LA CALIBRACION DEL SERVOMOTOR SE OBTIENE LO SIGUIENTE:
        --El servomotor SG90 alcanza un rango de 1ms con un paso inicial de 26e3.
        pwmSignal: process(conteoFrecuenciaPWM, posicion0Grados, movServo)begin
                --Este if crea el estado en alto y bajo de la señal PWM que crea la secuencia, yendo de 0 a 90 y 180 grados.
                if(conteoFrecuenciaPWM <= (posicion0Grados + movServo)) then
                        PWM <= '1';
                else
                        PWM <= '0';
                end if;
        end process pwmSignal;
end frecuenciaNueva;
```

Código UCF:

```
//ENTRADAS DEL MODULO divisorDeReloj:
net "relojNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "rst" loc = "H13";//El boton de Reset esta conectado al push button BTN3.
//Bit mas significativo del selectPos conectado al switch SW5 en el puerto L13.
net "selectPos[5]" loc = "L13";
//Bit 5 del selectPos conectado al switch SW4 en el puerto L14.
net "selectPos[4]" loc = "L14";
//Bit 4 del selectPos conectado al switch SW3 en el puerto K17.
net "selectPos[3]" loc = "K17";
//Bit 3 del selectPos conectado al switch SW2 en el puerto K18.
net "selectPos[2]" loc = "K18";
//Bit 2 del selectPos conectado al switch SW1 en el puerto H18.
net "selectPos[1]" loc = "H18";
//Bit menos significativo del selectPos conectado al switch SW0 en el puerto G18.
net "selectPos[0]" loc = "G18";

//CABLE DE SENAL PWM DEL SERVOMOTOR CONECTADO A LOS PUERTOS JA:
net "PWM" loc = "L15";//La senal PWM del servomotor esta conectada al puerto JA1.
//Bit mas significativo del ledAngulo conectado al led LD5 en el puerto P15.
net "ledAngulo[5]" loc = "P15";
//Bit 5 del ledAngulo conectado al led LD4 en el puerto E17.
net "ledAngulo[4]" loc = "E17";
//Bit 4 del ledAngulo conectado al led LD3 en el puerto K14.
net "ledAngulo[3]" loc = "K14";
//Bit 3 del ledAngulo conectado al led LD2 en el puerto K15.
net "ledAngulo[2]" loc = "K15";
//Bit 2 del ledAngulo conectado al led LD1 en el puerto J15.
net "ledAngulo[1]" loc = "J15";
//Bit menos significativo del ledAngulo conectado al led LD0 en el puerto J14.
net "ledAngulo[0]" loc = "J14";
```

Simulación PWM:





Al realizar la comparación de los programas de control de un servomotor realizados con Arduino y VHDL o Verilog, podremos ver que su mayor diferencia es que los pasos necesarios para mover el motor a una posición en específico varían mucho, pero no se notó alguna gran diferencia entre el control que proporcionan ambos, ya que el duty dycle en ambos casos es personalizable para cada servomotor.

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, “Servomotor Explicado”, 2022 [Online], Available: https://www.youtube.com/watch?v=qbKmI6_V_V8

Bitwise Ar, “Arduino desde cero en Español - Capítulo 6 - Servomotor (conexión, modelos, ajustes para uso óptimo)”, 2017 [Online], Available: https://www.youtube.com/watch?v=6bPVZg17vKc

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