INDICE

1.- OBJETIVO Y RESUMEN DEL PROYECTO 4

2.- MOTORES PASO A PASO 5

2.1.- INTRODUCCIÓN 5
2.2.-DESCRIPCIÓN DE LOS MOTORES PASO A PASO 6
2.2.1.- HOLDING TORQUE 8
2.2.2.- DETENT TORQUE 8
2.3.- CLASIFICACIÓN DE LOS MOTORES PASO A PASO 8
2.3.1.- MOTORES DE RELUCTANCIA VARIABLE 9
2.3.2.- MOTORES DE IMÁN PERMANENTE 12
2.3.3.- MOTORES HÍBRIDOS 13
2.3.4.- MOTORES DE IMANES PERMANENTES "CLAW-POLES" 16
2.4.- CARACTERÍSTICAS DE LOS MOTORES PASO A PASO 17
2.4.1.- CARACTERÍSTICAS ESTÁTICAS 17
2.4.1.1.- Características T/θ 17
2.4.1.2.- Características T/I 19
2.4.2.- CARACTERÍSTICAS DINÁMICAS 20
2.4.2.1.- Curvas características par/frecuencia 20
2.5.- MODOS DE EXCITACIÓN 22
2.5.1.- MODO PASO ENTERO 23
2.5.1.1.- Fases excitadas alternativamente 23
2.5.1.2.- Fases siempre excitadas 24
2.5.2.- MODO MEDIO PASO 25

3.- DRIVER MOTOR PASO A PASO 26

3.1.- TARJETAS CONMUTACIÓN MOSFETS 26
3.2.- TARJETA ENCODER 27

4.- COMUNICACIÓN DE DATOS USB 2.0 29

4.1.- INTRODUCCIÓN 29
4.2.- TOPOLOGÍA 30
4.3.- FUNCIONAMIENTO 30
4.4.- TIPOS DE TRANSFERENCIA 31
4.5.- SEÑALIZACIÓN Y CONECTORES 32
4.6.- POTENCIA 33
4.7.- FUTURO DEL USB 34

5.- MICROCONTROLADOR PIC18F4550 35

5.1.- INTRODUCCIÓN AL PIC18F4550 35
5.2.- FUNCIONES DEL MICROPROCESADOR 37
5.3.- COMUNICACIÓN USB CON EL ORDENADOR 38
5.4.- SEÑALES DE SALIDA PARA EL CONTROL DE LOS DRIVERS 39
5.4.1.- SEÑALES ANALÓGICAS 40
5.4.1.1.- Filtro paso bajo y seguidor de tensión 42
5.4.2.- SEÑALES DIGITALES 45
5.4.3.- ACELERACIÓN Y DECELERACIÓN 46
5.4.3.1.- Ecuaciones del movimiento del motor paso a paso 46
5.4.3.2.- Rampa de velocidad lineal 48
5.4.3.3.- Cálculo exacto del tiempo entre pasos 49
5.5.- ENTORNO DE TRABAJO CON EL MICROPROCESADOR 50
5.5.1.- MPLAB IDE 50
5.5.2.- MPLAB C18 50
5.5.2.1.- Creación de un nuevo proyecto 51
5.6.- CÓDIGO C EN EL MICROPROCESADOR 55
5.6.1.- MAIN.C 56
5.6.2.- USER.C 62
5.6.3.- ENCADAIN.C 69
5.6.4.- USER.H 77

6.- INTERFAZ CONTROL MOTOR PASO A PASO EN C# 80

6.1.- INTRODUCCIÓN AL LENGUAJE C# Y .NET FRAMEWORK 80
6.1.1.- LENGUAJE C# 80
6.1.2.- ARQUITECTURA DE LA PLATAFORMA .NET FRAMEWORK 82
6.1.3.- FORMULARIO WINDOWS FORMS 84
6.2.- FUNCIONAMIENTO DEL INTERFAZ CONTROL MOTOR PASO A PASO 85
6.2.1.- CONTROL DEL MOTOR 86
6.2.1.1.- Tipo de movimiento 86
6.2.1.2.- Sentido de giro 87
6.2.1.3.- Variables control motor paso a paso 87
6.2.1.4.- Impulsos del encoder 88
6.2.1.5.- Applicate & Stop 89
6.2.2.-GRÁFICA VELOCIDAD (RPM)-TIEMPO (S) 89
6.3.- ESTRUCTURA INTERNA DEL INTERFAZ CONTROL MOTOR PASO A PASO 90
6.3.1.- TEMPORIZADOR INTERFAZ CONTROL MOTOR PASO A PASO 91
6.3.2.- APPLICATE. PUESTA EN MARCHA DEL MOTOR PASO A PASO 92
6.3.3.- STOP. PARADA DEL MOTOR PASO A PASO 93
6.3.4.- ESTRUCTURA CONTROL DE MOVIMIENTO 94
6.4.- ENTORNO DE TRABAJO EN C# 95
6.4.1.- INTRODUCCIÓN AL ENTORNO IDE (VISUAL C#) 95

6.4.2.- HERRAMIENTAS DE VISUAL C# 95
6.4.3.- CÓMO EXPONE LAS HERRAMIENTAS EL IDE 96
6.4.3.1.- Ventanas del Editor y del Diseñador de Windows Forms 96
6.4.3.2.- Explorador de soluciones y Diseñador de proyectos 97
6.4.3.3.- Ventanas Compilador, Depurador y Lista de errores 97
6.4.4.- PERSONALIZAR EL IDE 98
6.5.- CÓDIGO C# EN EL ORDENADOR 99
6.5.1.- USBDEMO.CS 100
6.5.2.- CONTROLMOVIMIENTO.CS 111
6.5.3.- USB_INTERFACE_INIGO.CS 114

7.- FUNCIONAMIENTO GLOBAL DE SISTEMA DE CONTROL DE UN MOTOR PASO A PASO 118

7.1.- FLUJO DE DATOS EN EL ENCENDIDO 119
7.2.- FLUJO DE DATOS EN EL APAGADO 120
7.3.- FLUJO DE DATOS DESDE EL ENCODER 120

8.- CONCLUSIONES 122

9.- LÍNEAS FUTURAS 124

10.- BIBLIOGRAFÍA 125

1.- Objetivo y resumen del proyecto

El objetivo de este proyecto es crear un equipo con el que comprender y controlar, desde el ordenador, el funcionamiento de un motor paso a paso bifásico de imanes permanentes, a través de una comunicación USB, la ayuda de un microprocesador y el imprescindible driver del motor paso a paso.

Uno de los elementos utilizados en el control de un motor paso a paso es el microprocesador, encargado tanto de enviar las consignas al controlador (driver), como de captar las señales provenientes del encoder. Para cumplir esta función se ha decidido la utilización del microprocesador 18F4550 de MicroChip, principalmente por su capacidad de comunicación mediante el protocolo USB 2.0 con el ordenador.

La programación del microprocesador se realiza en Ansi C mediante las herramientas específicas que MicroChip proporciona, para poder trabajar con ellas es obligatorio un aprendizaje previo tanto del las herramientas específicas que proporciona MicroChip como del lenguaje Ansi C.

La aplicación Windows para crear el interfaz en el ordenador ha sido desarrollada en la plataforma Microsoft .NET utilizando el lenguaje orientado a objetos C #, un lenguaje potente y sencillo.

El primer paso en la programación de nuestro interfaz, es introducirnos en el lenguaje C# y conocer sus diferentes posibilidades.

Desde el interfaz creado en C# podremos configurar diferentes aspectos del motor paso a paso, como la velocidad de giro, modo de funcionamiento, número de pasos de aceleración, la velocidad final,… pero también podremos observar en una gráfica las consecuencias de las diferentes configuraciones.

Debemos mencionar que en este proyecto se han utilizado herramientas actuales como son la plataforma Microsoft .NET, el microprocesador 18F4550, la comunicación USB 2.0, con el fin adquirir unos conocimientos extrapolables a otras áreas de trabajo por un lado y de tener una base sólida sobre la que realizar actualizaciones en un futuro fácilmente.

2.- Motores paso a paso

2.1.- Introducción

El motor paso a paso es el convertidor electromecánico que permite la conversión de una información en forma de energía eléctrica, en una energía mecánica y una información de posición. Está constituido por un estator cuyos devanados se llaman fases y un rotor de un elevado número de polos. Su funcionamiento es síncrono y la alimentación cíclica de sus fases debe originar en cambio de configuración un giro elemental del rotor, constante, llamado paso.

Existe una gran diversidad de modelos de estos motores dependiendo del número de fases de su estator, de si la alimentación de estas es unipolar o bipolar, del número de paso por vuelta y de si su rotor es de reluctancia variable, imanes permanentes o híbridos.

En cuanto al control, existen tres modos de realizarlos, paso entero, medio paso y micropaso.

En el paso entero, cada vez que se modifica la alimentación de las fases del estator se avanza un paso disponiendo de par nominal del rotor.

En el medio paso se avanza sólo medio paso con lo que se dispone de mejor resolución, pero el par en las posiciones situadas entre pasos regulares se reduce a la mitad.

Estos dos tipos de funcionamiento disponen en el mercado de gran variedad de integrados para su control.

El funcionamiento en micropaso consiste en alimentar al mismo tiempo varias fases a la vez con corrientes medias distintas, de modo que la posición media del flujo en el entrehierro se puede fijar en cualquier posición. Con este funcionamiento se consigue una resolución inmejorable y existen en el mercado distintas tarjetas de control basadas en microprocesador.

Merece la pena comentar que el motor paso a paso es la primera de las máquinas eléctricas que sin el uso de la electrónica no tiene razón de ser.

El control de posición de motores paso a paso se puede efectuar en lazo abierto siempre que se tomen las precauciones necesarias para no perder ningún paso. Indicaremos que utilizando técnicas de PWM para el control de la corriente, asegurando de esta manera un aprovechamiento máximo de par y con la programación adecuada de aceleración y deceleración, se puede trabajar perfectamente en lazo abierto siempre que las variaciones que el par de carga sean conocidas de antemano. Es en estas aplicaciones de carga conocida donde el motor paso a paso tiene sus posibilidades industriales. En el caso de tener que accionar cargas desconocidas, su funcionamiento en lazo cerrado sería del todo perfecto, pero el coste del transductor de realimentación generalmente no justifica esta aplicación.

2.2.-Descripción de los motores paso a paso

La siguiente figura representa la sección de un típico motor paso a paso. Estudiaremos de forma sencilla el funcionamiento de esta máquina.

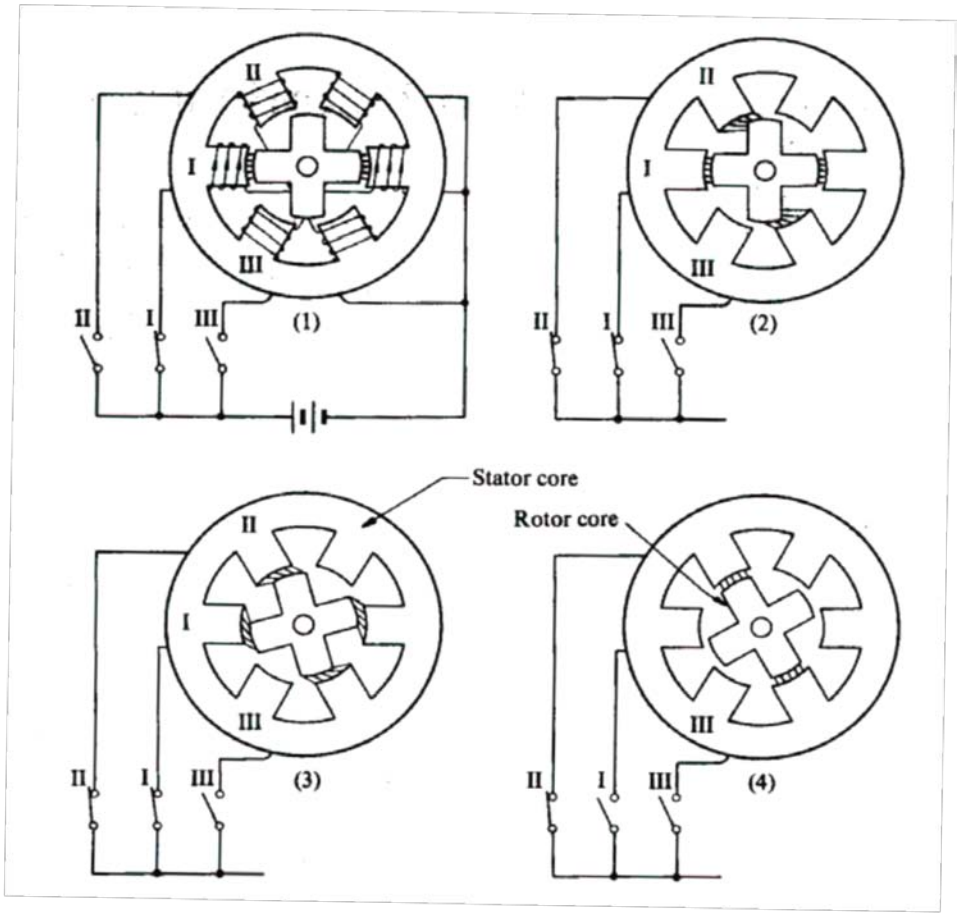


Figura 2.1-Esquema básico de funcionamiento de un motor paso a paso

El estator tiene seis polos salientes, mientras que el rotor solamente dispone de cuatro. Ambos se suelen construir generalmente de acero blando. Los polos del estator se han bobinado para formar tres fases, cada una de las cuales consta de dos bobinas conectadas en serie y situadas físicamente en polos opuestos. La corriente se aplica desde una fuente de potencia DC a través de los interruptores I, II y III.

Estando el motor en el estado (1) es la fase I que está excitada. El flujo magnético que cruza el entrehierro debido a esta excitación se indica con flechas. En este estado los dos polos del estator pertenecientes a la fase I están alineados con dos de los cuatro polos del rotor, permaneciendo este en posición de equilibrio.

Continuamos con el estado (2). Para ello cerramos el interruptor II. Primeramente se establece el flujo representado en (2), creándose un par en sentido anti-horario, debido a las tensiones Maxwell, que obliga al rotor a alcanzar la posición de equilibrio del estado (3), girando para ello 15^{o}.

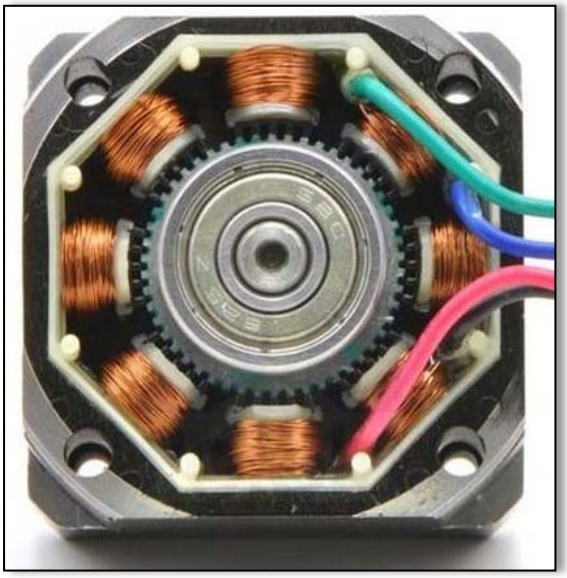


Figura 2.2

Cada vez que realizamos una apertura o cierre de un interruptor se produce un giro de 15^{o} de rotor. Este ángulo fijo se denomina ángulo de paso y es una característica básica dentro de este tipo de motores. Existen motores paso a paso con una extensa variedad de ángulos de paso dependiendo de la mayor o menor resolución que necesite. Así existen motores con ángulos que van desde las décimas de grado hasta los 90^{o}.

Si ahora abrimos el interruptor I, el rotor alcanza la posición de equilibrio representada en el estado (4).

Siguiendo una secuencia correcta de control de la apertura y cierre de los interruptores, podremos girar el motor en el sentido y a la velocidad que deseemos con la ventaja de no tener que utilizar ningún tipo de realimentación, Además el error de posición que puede tener este tipo de motores no es acumulativo y tiende a cero en cuatro pasos, es decir cada 360^{o} eléctricos. Cada cuatro pasos el rotor vuelve a la misma posición con respecto a la polaridad

magnética y a la trayectoria del flujo. La precisión en el posicionado es un factor que mide la calidad de estos motores. Se diseñan de modo que tras recibir una señal eléctrica pasen de una situación de equilibrio a otra posición de equilibrio diferente separada de la anterior un determinado ángulo. Esta precisión depende en gran manera del mecanizado del rotor y estator con lo que su fabricación es delicada. Cuando una carga se aplica sobre el eje, se produce un par elevado que trata de posicionar el rotor en su posición natural de equilibrio. La responsabilidad de que este par sea mayor o menor recae en el entrehierro. Cuanto más pequeño sea, y esto depende del al calidad de la fabricación, el par que presente el motor a la carga y su precisión serán mayores.

Existen dos conceptos que sirven para diferenciar el comportamiento del motor paso a paso en cuanto al par mencionado anteriormente.

2.2.1.- Holding torque

Definido como el máximo par estático que se le puede aplicar al eje de un motor excitado sin causarle rotación continua.

2.2.2.- Detent torque

Definido como el máximo par estático que se le puede aplicar al eje de un motor no excitado sin causarle rotación continua.

En general cuanto mayor sea el “holding torque” menor es el error de posición debido a la presencia de una carga externa sobre el eje. El “detent torque” aparece solamente en los motores paso a paso de imanes permanentes que se discutirán posteriormente.

2.3.- Clasificación de los motores paso a paso

Anteriormente se ha dedicado un breve apartado para explicar el principio general de funcionamiento de estos motores de una manera sencilla. Vamos a

profundizar un poco más ya que dependiendo de su estructura física se puede hacer una clasificación en función del principio de funcionamiento.

2.3.1.- Motores de reluctancia variable

Pertenecen a esta categoría la mayoría de los motores paso a paso que se encuentran en el mercado. La figura mostrada a continuación nos servirá para indicar su funcionamiento:

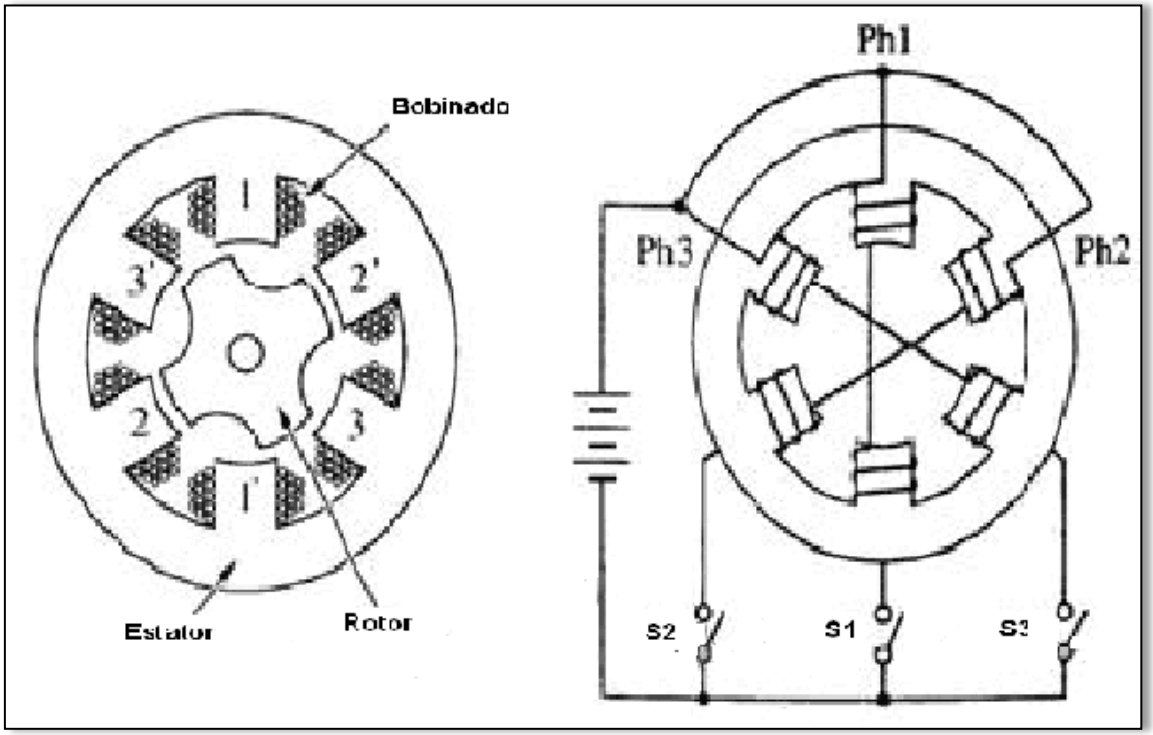


Figura 2.3-Sección de un motor paso a paso de reluctancia variable

En esta figura se representa un motor de tres fases con seis dientes salientes en el estator. Cada par de polos separados 180^{o} entre sí constituyen una fase. Las bobinas de cada fase arrolladas sobre los correspondientes polos se conectan en serie. El rotor consta de cuatro polos. Tanto el rotor como el estator deben de estar construidos con materiales de alta permeabilidad magnética y ser capaces de permitir el paso de un gran flujo magnético incluso cuando se aplique una pequeña fuerza magnetomotriz.

Aún cuando no siempre tiene por qué ser así, vamos a asumir que las polaridades de los polos pertenecientes a la misma fase son opuestas. Por

tanto, en la figura 2.3 constituirán el polo norte y los polos I', II' y III' el polo sur cuando circule corriente por sus devanados.

La corriente de cada fase se controla mediante la apertura y cierre de los diferentes interruptores. Si una corriente se aplica a las bobinas de la Fase 1º, dicho de otro modo, si excitamos las Fase 1, se establecerán unas líneas de flujo similares a las representadas en la figura 2.4

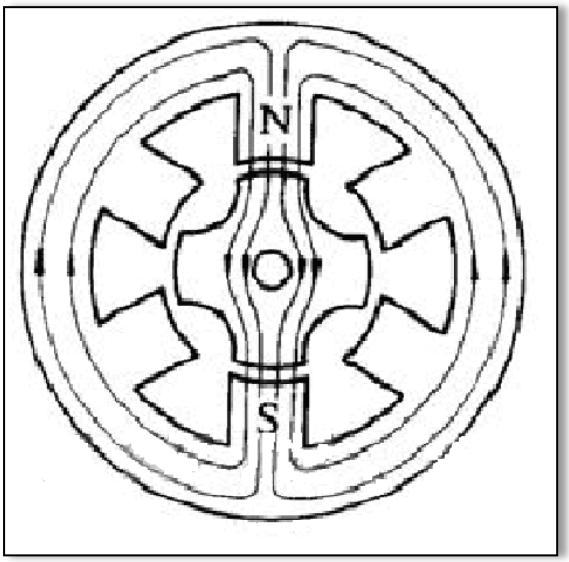


Figura 2.4-Líneas de flujo

El rotor se posicionará de modo que queden alineados dos polos opuestos suyos con los polos I y I' del estator. Cuando los polos del rotor y del estator quedan alineados se minimiza la reluctancia magnética del circuito magnético y el motor se encuentra entonces en una posición de equilibrio. Si el rotor tiende a moverse de su posición de equilibrio debido al par generado por la presencia de una carga externa, internamente se genera un par en sentido contrario que intenta conducir al rotor a su posición de equilibrio original. La figura 2.5 ilustra esta situación.

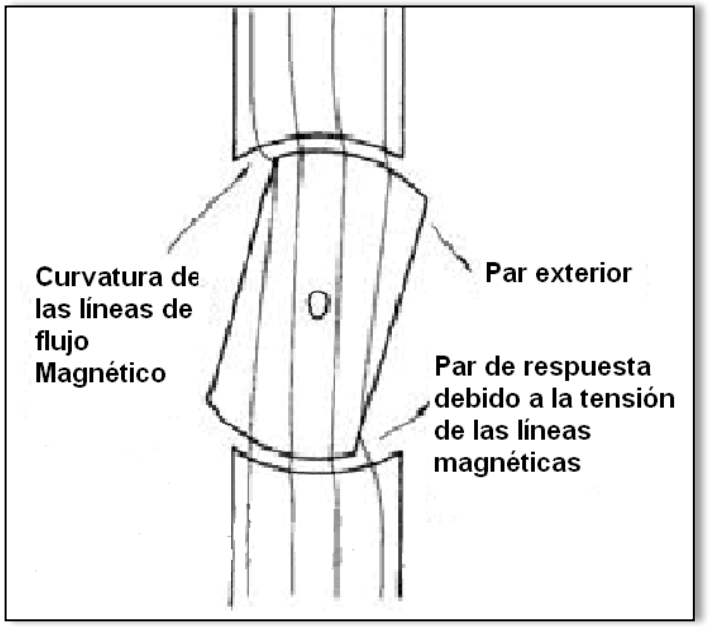


Figura 2.5- disposición de líneas de flujo al aplicarse un par externo

En esta figura el par externo se aplica en el sentido horario y el rotor se desplaza en esa misma dirección. Como resultado de este desplazamiento las líneas de flujo magnético que atraviesan el entrehierro y que inicialmente, antes de aplicar ningún par externo, eran rectilíneas se curvan en los bordes de los polos del estator y rotor. Estas

líneas magnéticas tienden a ser cortas y rectilíneas como sea posible, creando una tensión, conocida como *tensión de Maxwell*, que provoca un par de sentido contrario al par inicial que había distorsionado estas líneas de flujo.

Se puede ver en la misma figura como cuando los polos del rotor y del estator están desalineados la reluctancia magnética es mayor, de modo que el motor de reluctancia variable trabaja siempre en condiciones de reluctancia mínima. Veamos ahora qué ocurre cuando la Fase 1 se desconecta y se conecta las Fase 2. La reluctancia magnética del motor vista desde la fuente de potencia DC se incrementará súbitamente justo después de la conmutación de los interruptores. El resultado se puede ver en la figura2.6, el rotor girará 30^{o} en sentido anti-horario con el fin de restablecer las condiciones de reluctancia mínima.

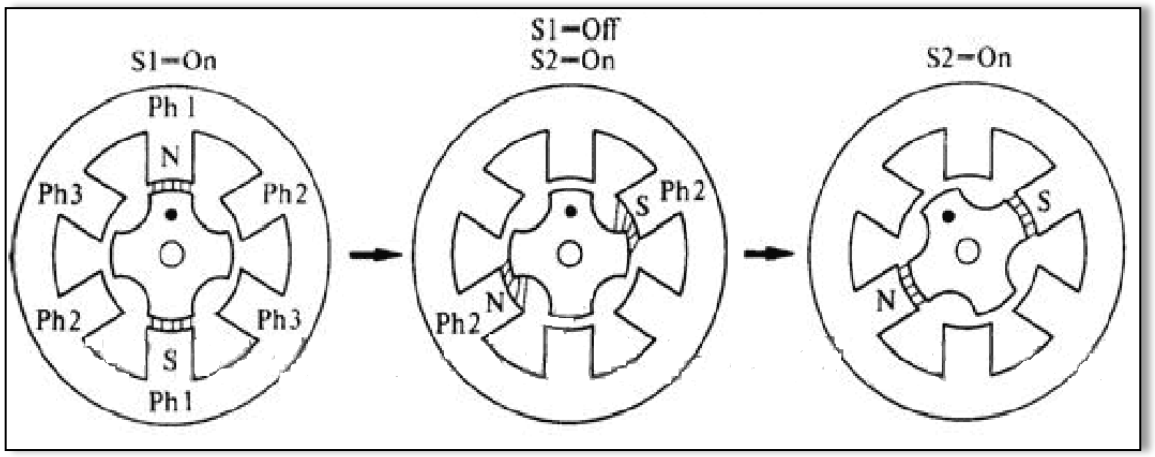


Figura 2.6- El rotor se desplaza un paso al cambiar la excitación de la Fase I a la Fase II

El entrehierro debe ser tan pequeño como sea posible para producir pares grandes a partir de pequeños volúmenes de rotor y poder alcanzar gran precisión en el posicionado. La figura 2.7 muestra dos entre-hierros diferentes. Para el mismo valor de fuerza magnetomotriz un entrehierro pequeño proporcionará mayor flujo magnético, lo que se traduce en un par mayor.

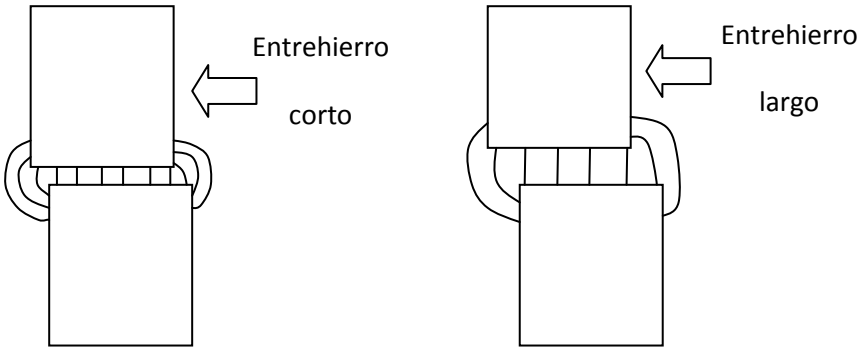


Figura 2.7- Comparación de las líneas de flujo para dos entrehierros diferentes

Está claro que el desplazamiento a partir de la posición de equilibrio cuando se aplica un par externo es más pequeño cuanto menor sea el entrehierro. Bajo otro punto de vista y olvidándonos de la figura 2.7 es posible también afirmar la necesidad de que el entrehierro sea el menor posible. El citado entrehierro es también el lugar donde se almacena prácticamente toda la energía en un circuito magnético. Pero en los motores paso a paso no deseamos que la energía suministrada por la fuente de potencia se almacene en ningún entrehierro sino que lo que se pretende es convertir la mayor parte de ella en trabajo mecánico de movimiento del rotor. Por tanto, toda la energía que se almacene en el entrehierro es energía de pérdidas que hay que minimizar construyendo entrehierros con el menor espesor posible. Actualmente los entrehierros van desde los 30 hasta las 100µm.

Para disminuir el ángulo de paso es necesario aumentar los polos del estator y el rotor. En el estator se suelen incluir una serie de dientes en cada polo, todos con la misma polaridad cuando se excita la fase correspondiente, para conseguir ángulos de paso menores.

2.3.2.- Motores de imán permanente

Se denominan así los motores paso a paso que poseen un imán como rotor. Para explicar el funcionamiento de este tipo de motores estudiaremos las figura 2.9 que representa un motor paso a paso de imán permanente de cuatro fases.

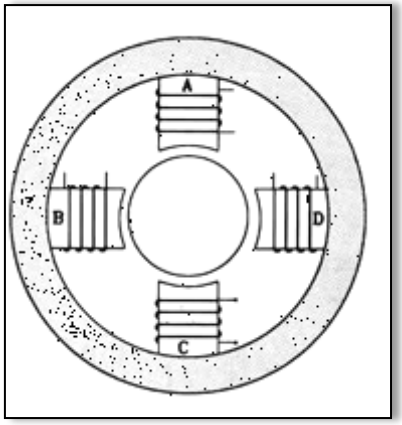


Figura 2.9-Motor paso a paso de imán permanente de cuatro fases

El imán cilíndrico se utiliza como rotor. El estator, por el contrario, está formado por cuatros polos bobinados constituyendo cada uno de ellos una fase diferente. Cuatro interruptores conectando cada fase con una fuente de potencia DC completan el esquema de control del motor. Si las fases se excitan con la secuencia Fase 1-> 2-> 3-> 4 el motor girará en sentido horario girando en cada paso 90^{o}. Para disminuir el ángulo de paso es necesario aumentar los polos del estator y los polos magnéticos del rotor.

Una característica destacable de este tipo de motores es que el rotor permanece en posiciones fijas aunque se desconecte la fuente de potencia. Estas posiciones coinciden con las posiciones que va alcanzando el motor si es excitado con una secuencia tal que en todos los casos es una sola fase la que está excitada.

Una desventaja importante de este tipo de motores es que la máxima densidad de flujo viene limitada por el magnetismo remanente del rotor.

2.3.3.- Motores híbridos

Este tipo de motores también tiene por rotor un imán permanente. Se le denomina híbrido porque su funcionamiento se basa en los dos tipos de motores explicados anteriormente.

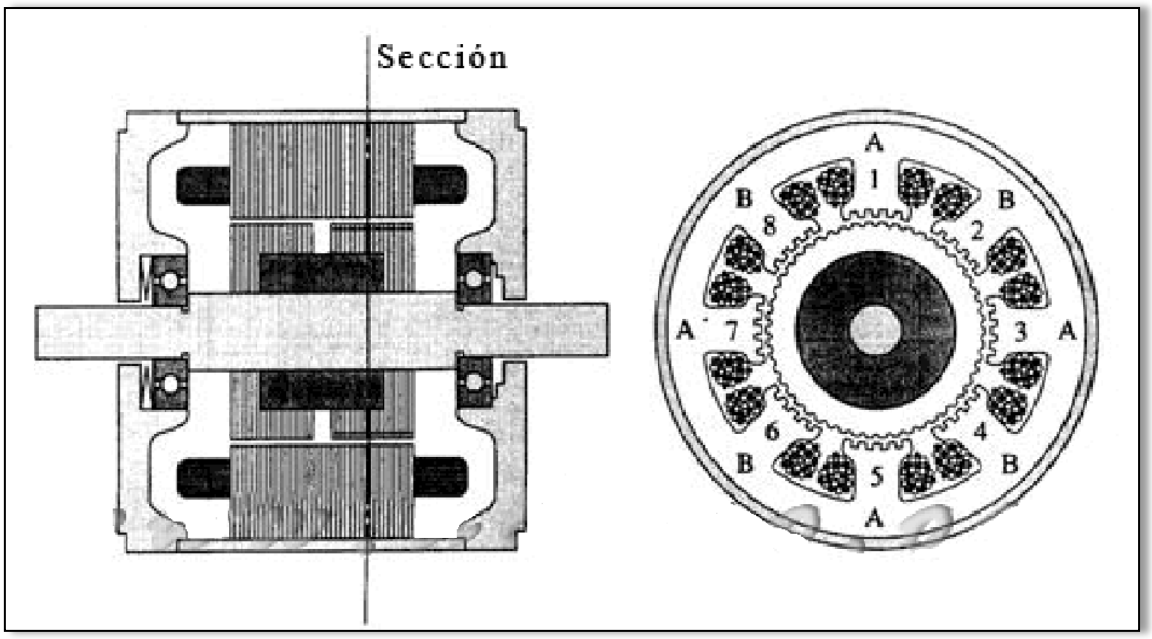


Figura 2.10-Sección de u motor paso a paso híbrido

La figura 2.10 ilustra un motor típico de estas características con cuatros fases. La estructura del estator coincide con la de un motor de reluctancia variable, no así los arrollamientos, ya que en este caso los dientes de los polos pueden corresponder a fases diferentes. En el caso de la figura, las bobinas de dos fases diferentes se arrollan en el mismo polo con lo cual según qué fase esté excitada en cada momento el polo pertenecerá a una fase o a otra.

Otra característica importante es la estructura del rotor

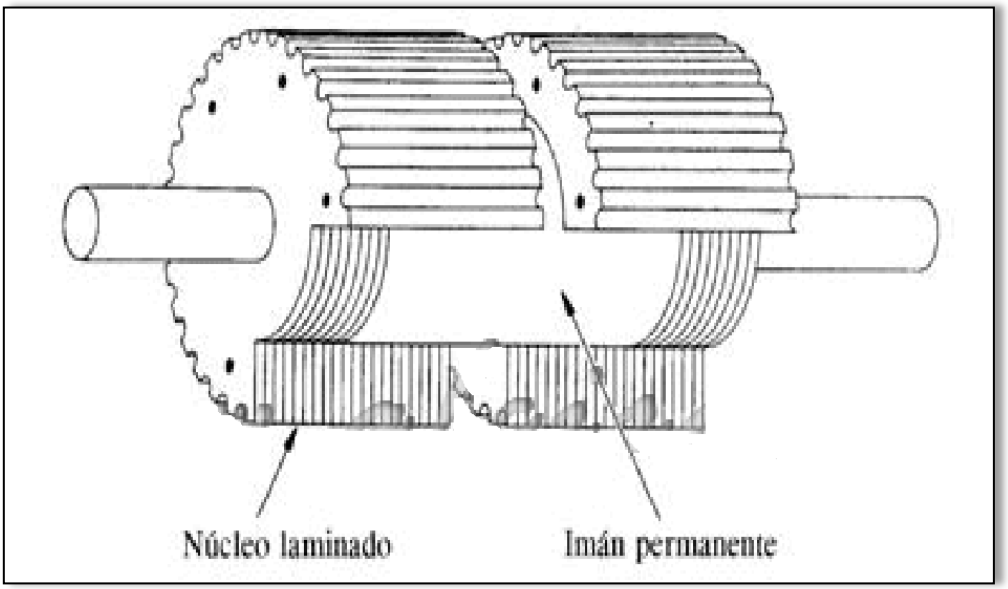


Figura 2.11-Estructura del rotor en un motor híbrido

La figura 2.11 ilustra como un imán permanente de forma cilíndrica se aloja en el núcleo del rotor. Está magnetizado longitudinalmente. Cada polo de este imán está recubierto de una estructura cilíndrica dentada construida generalmente de acero blando. Los dientes de las dos secciones están desalineados medio diente unos respecto otros.

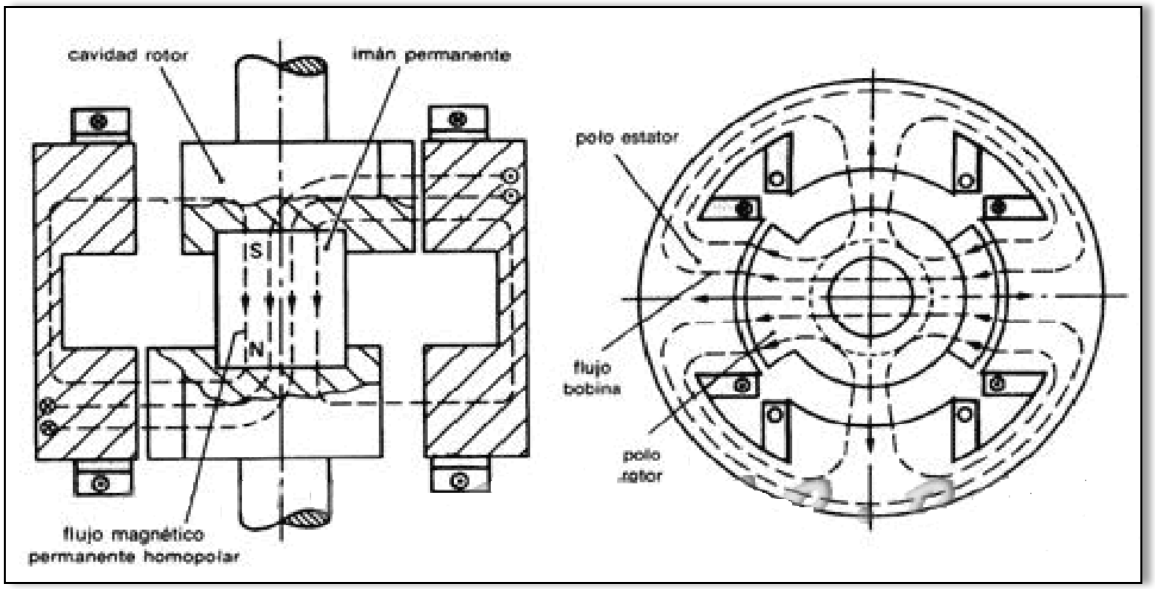


Figura 2.12.a Líneas de flujo producidas por el imán permanente del rotor

Figura 2.12.b Líneas de flujo producidas por el imán permanente del rotor

El campo magnético generado por las bobinas del estator se representa en la figura 2.12.b.

El funcionamiento para una secuencia de una fase activa lo muestra la figura 2.13. En el estado 1 los polos de la fase A están excitados, y los dientes del polo 1 atraen a los dientes del rotor del polo norte, mientras que los del polo 3 atraen de igual forma a los dientes del polo sur del rotor. Cuando la corriente (i) es conmutada a la fase B (estado 2), el rotor se desplaza un cuarto de espacio de un diente, quedando alineados el polo norte del rotor con el polo 2 del estator y el polo sur del rotor con el polo 4 del estator. De nuevo la corriente (-i) se conmuta a la fase A (estado 3) produciéndose un nuevo desplazamiento del rotor en un cuarto de espacio de diente, quedando alineado en sentido opuesto (polo 1 con polo sur y polo 3 con polo norte). Otra conmutación de la corriente (-i) en la fase B (estado 4) produce un nuevo desplazamiento y una nueva alineación inversa de los polos de esta fase con el rotor. Retornando al estado 1 (i), el rotor ha dado 4 pasos de un cuarto del espacio de un diente

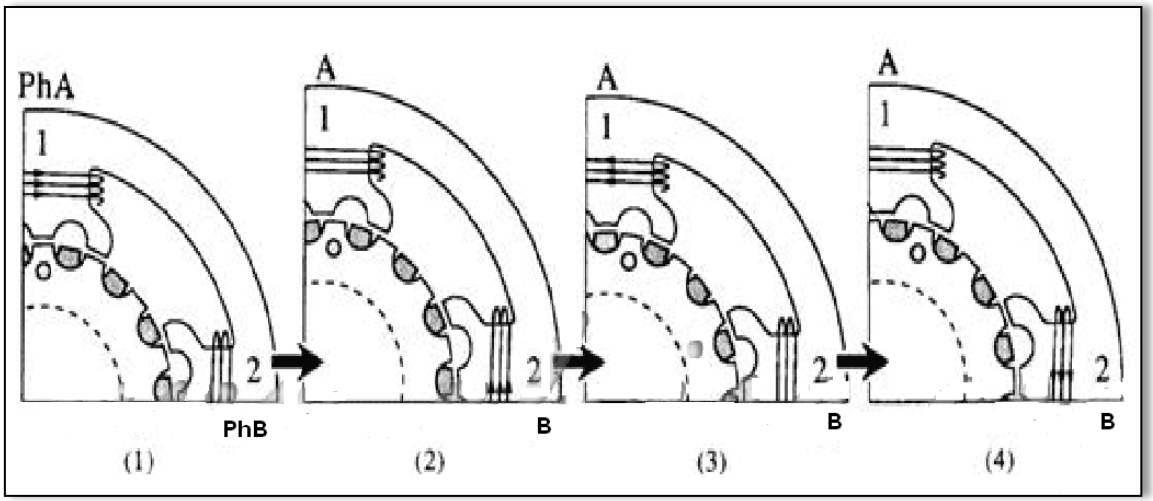


Figura 2.13- Operación de una fase activa de un motor híbrido de dos fases

2.3.4.- Motores de imanes permanentes "Claw-Poles"

Con la explicación de los tres tipos de motores anteriores se tiene ya un conocimiento básico del principio por el que operan la gran mayoría de los motores paso a paso. A la hora de su construcción, estos motores difieren ligeramente del modelo teórico al que deberían pertenecer y otro tanto ocurre con su modo de operación. Este es el caso del motor utilizado en este proyecto,

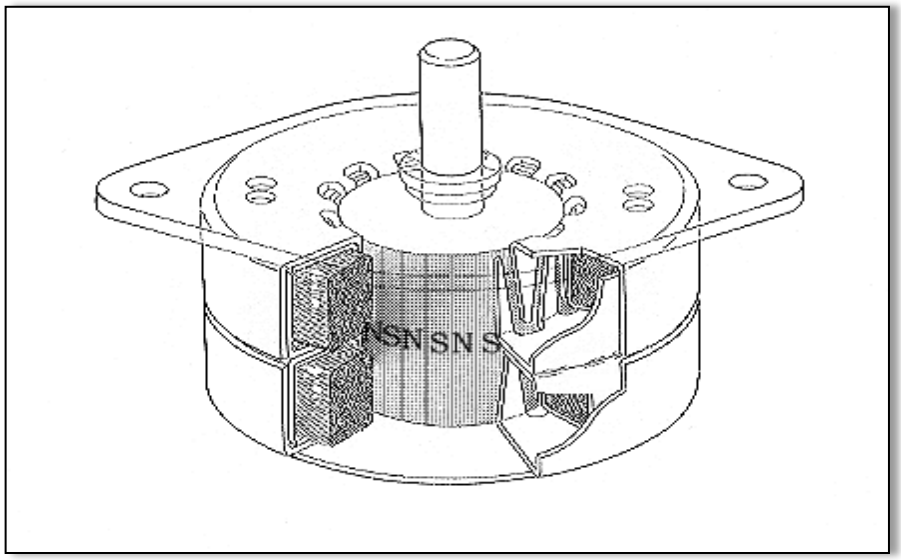


Figura 2.14- Sección de un motor de imán permanente "claw-pole"

que se puede encuadrar dentro de la categoría de motores de imanes permanentes, pero que su especial construcción lleva a dedicarle un apartado

exclusivo con el fin de comprender mejor su funcionamiento. La figura 2.14 muestra la sección de uno de estos motores.

Se observa como el estator está formado por dos partes. Cada una de estas partes está formada a su vez por dos estructuras provistas de dientes afilados que se entrelazan. Por el interior de estas estructuras dentadas se sitúan las bobinas necesarias para crear el campo magnético en el estator; en nuestro motor el número de bobinas es dos, una en cada parte del estator. Los dientes entre cada parte del estator están desalineados, una distancia correspondiente a medio diente. En este tipo especial de motores paso a paso, el rotor los constituye un imán permanente magnetizado con polaridad norte y sur tantas veces como pares de dientes entrelazados tiene el estator.

El movimiento se produce por la tensión de Maxwell originada en cada cambio de excitación de las fases debido a las polaridades magnéticas en rotor y estator. En cada paso el motor se desplaza medio diente hacia un sentido u otro, dependiendo del sentido de la corriente por las bobinas. Un motor bifásico con doce pares de dientes entrelazados en cada parte del estator dará cuarenta y ocho pasos por revolución lo que supone un ángulo de paso de 7.5^{o}.

2.4.- Características de los motores paso a paso

Vamos a estudiar a continuación los conceptos fundamentales que caracterizan a todo motor paso a paso.

2.4.1.- Características estáticas

Características con el motor en reposo

2.4.1.1.- Características T/θ

El motor paso a paso permanece en una posición de equilibrio alcanzada por medio de la excitación de las correspondientes fases. Si en estas condiciones aplicamos al eje del motor un par externo, el rotor se desplazará un

determinado ángulo respecto de la posición de equilibrio. La figura 2.15 muestra la relación entre el valor del par externo aplicado y este desplazamiento angular del rotor.

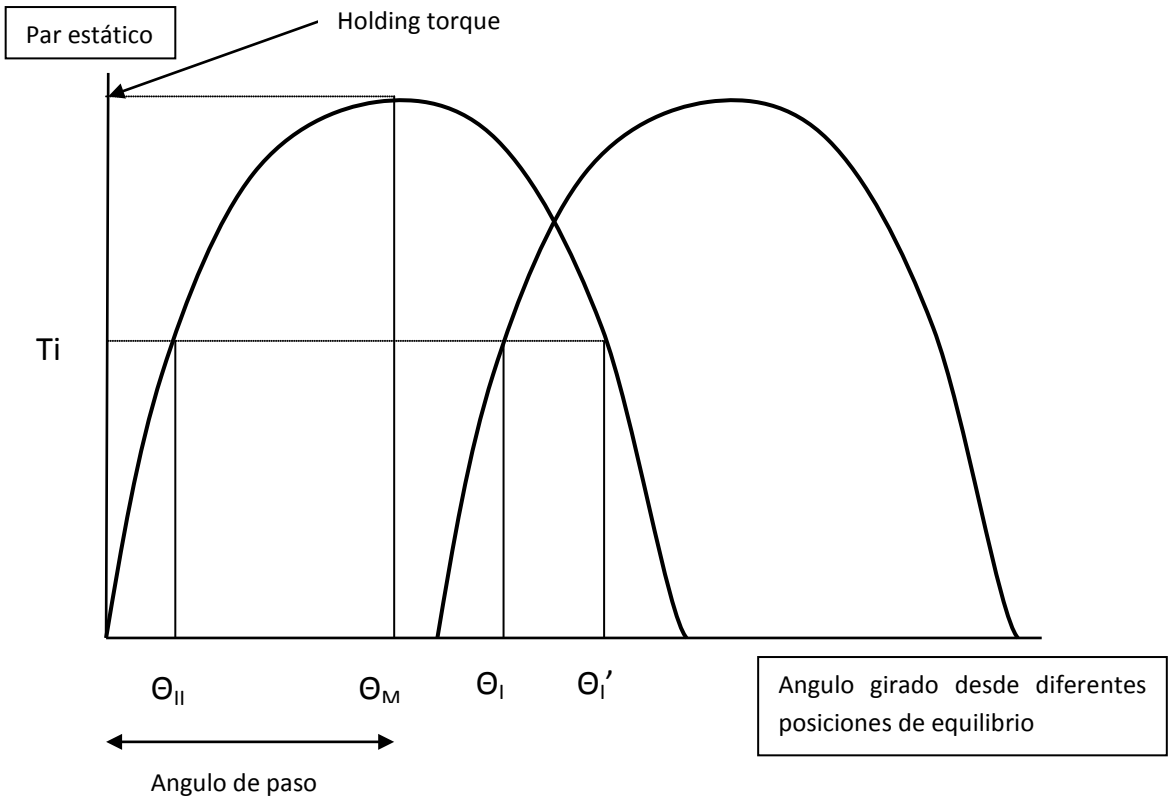


Figura 2.15- Curva característica de par de mantenimiento para varias intensidades.

A la curva resultante se le denomina genéricamente "Curva característica T/θ". El máximo de esta curva se denomina "holding torque" y ocurre en $\theta=\theta_M$. Para desplazamientos mayores a los que marca el pico, el par aplicado externamente no tiende a llevar el motor a su posición de equilibrio inicial sino a la siguiente. El holding torque se define rigurosamente como el "máximo par estático que puede ser aplicado al eje de un motor paso a paso excitado sin causarle una rotación continua".

La construcción de esta curva se realiza en dos partes. Primeramente se excita el motor y se van aplicando pares externos gradualmente mayores hasta llegar al holding torque. Si aplicamos un par mayor, el motor no será capaz de oponer la resistencia necesaria para contrarrestar ese par y comenzará a girar de forma continua en la dirección del par externo mientras este no cese. Para realizar la segunda parte de la curva es necesario estacionar el motor en la posición de

equilibrio siguiente a la que se encontraba inicialmente. Para ello habrá que excitar correctamente el motor y naturalmente de forma diferente a la inicial. Si aplicamos ahora un par externo T_i, el motor, respecto a esta nueva posición de equilibrio girará un ángulo θ_I. Si en esta posición modificamos las fases que están excitadas y volvemos a la misma excitación inicial, el motor se desplazará hasta θ_I con lo que ya tenemos construido un punto de la segunda parte de la curva. Significa que inicialmente, el motor, para un para externo T_i , gira u ángulo θ_{II}.

2.4.1.2.- Características T/I

El holding torque aumenta con la corriente de excitación de las fases. La figura 2.16 muestra la relación entre estos dos parámetros para dos tipos de motores paso a paso diferentes, uno de reluctancia variable y otro híbrido. Se puede observar como el par no se anula para corriente de excitación nula en el caso de motor híbrido debido a la presencia del imán permanente en el rotor.

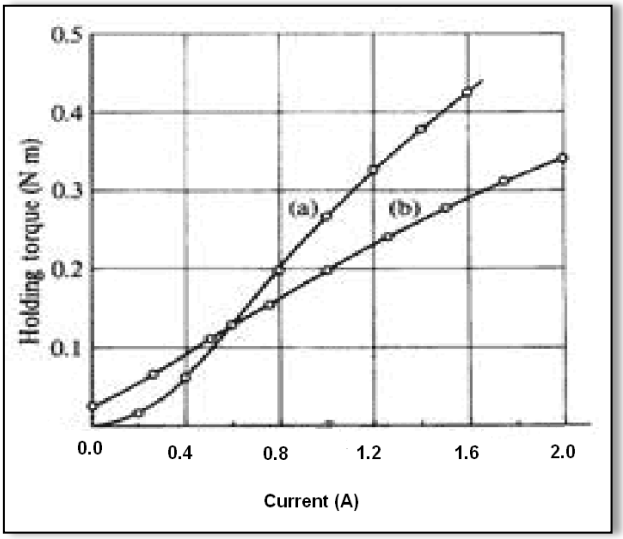


Figura 2.16-Ejemplo de característica par/intensidad. (a) Motor de reluctancia variable de cuatro fases y 1,8º, y (b) motor híbrido

2.4.2.- Características dinámicas

Las característica de comportamiento dinámico del motor que nos relacionan la velocidad y el par, pudiéndose derivar de éstas el arranque, el paro, y la aceleración.

2.4.2.1.- Curvas características par/frecuencia

Los motores paso a paso son usados para el posicionamiento en sistemas mecánicos que requieren un control preciso del paso, el par que generan tiene que ser suficiente para arrastrar las cargas a las que están sometidos, en secuencias de aceleración, desaceleración o trabajando a velocidad constante. Las condiciones de trabajo, las necesidades de velocidad y aceleración condicionan la elección del motor que debe cumplir con los requisitos de par/velocidad necesarios. Para llevar a cabo la elección, nos tenemos que basar

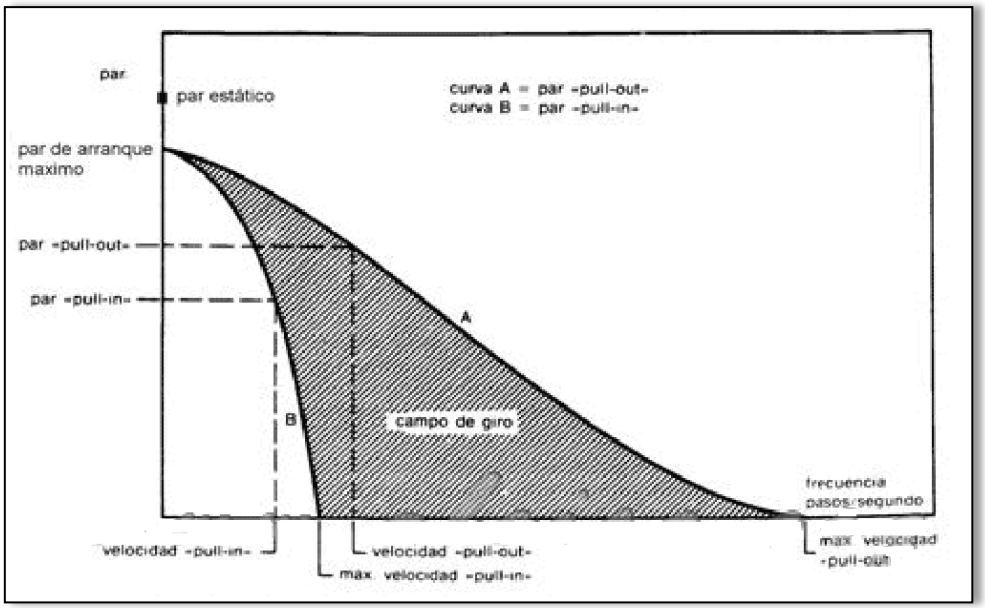


Figura 2.17-Curva de características dinámicas

en las curvas de par/velocidad que proporciona el fabricante. Éstas están formadas por dos curvas características; la primera denominada 'pull in' nos indica el par de arranque o paro sin pérdida de pasos en función de la velocidad de arranque o paro, con el motor en estado de reposo, la segunda 'pull out' nos da el par máximo de trabajo cuando el motor se encuentra en funcionamiento.

Entre las curva de 'pull in' y la de 'pull out' tenemos el área de aceleración desaceleración que se conoce como campo de giro o zona de arrastre.

La figura 2.17 nos muestra las curvas típicas de un motor paso a paso, la curva 'pull in' queda delimitando la zona de arranque/paro, indicándonos en sus extremos la máxima frecuencia de arranque y el par máximo de arranque. Para que el motor pueda arrancar, se tiene que confrontar la curva de arranque 'pull in' con la del par resistente del sistema y encontrar la frecuencia máxima de arranque, por encima de éste el par que entrega el motor es inferior al de la carga, quedando bloqueado. La curva 'pull out' establece el par máximo de trabajo y la máxima frecuencia de trabajo. Si la relación par/frecuencia cae fuera de los límites de la curva 'pull out' el rotor pierde el sincronismo del campo magnético generado por la excitación, provocando la pérdida de pasos o el paro completo, dejando el motor en un estado de oscilación sin movimiento.

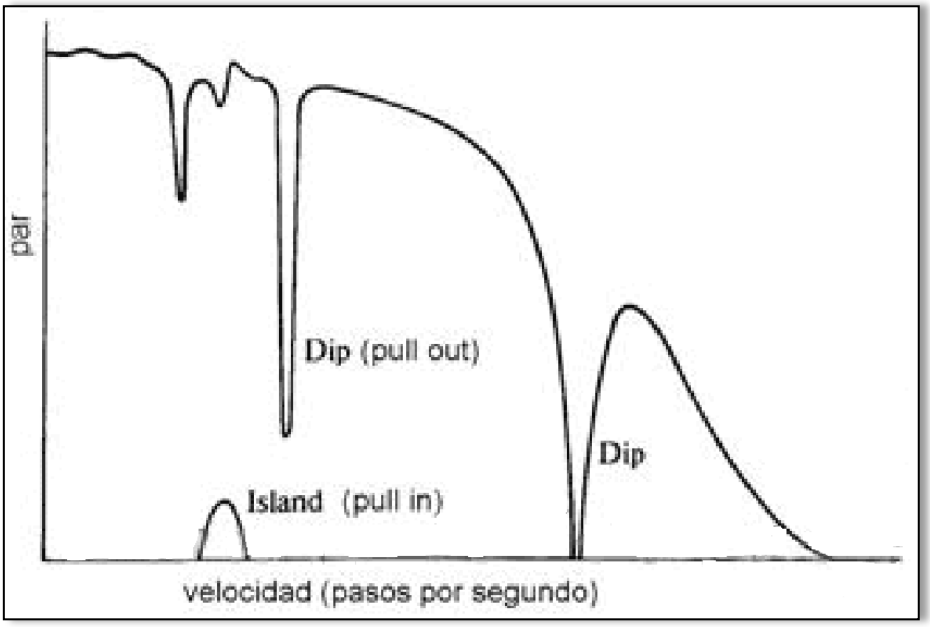


Figura 2.18-Ejemplo de curvas de característica con dips y islands

Las curvas de par/frecuencia presentan una serie de inestabilidades en la zona de bajas frecuencia, de 10Hz a 100Hz, variando según el tipo de motor. L figura 2.18 muestra los denominados 'dips' valles o inclinaciones hacia abajo de la curva característica 'pull out'. Éstos ocurren por la resonancia mecánica que experimenta el motor a estas frecuencias, pudiendo ser variados por la acción de la carga del sistema mecánico que esté acoplado al eje del motor. Estos 'dips' producen una disminución drástica del par generado por el motor incluso la anulación total, provocando el paro de éste con cargas mínimas o trabajando en vacío.

Otro tipo de inestabilidades son las denominadas ‘islands' islas que forman parte de la curva de ‘pull in'. En estas zonas el motor no es capaz de arrancar y se pone a oscilar mientras tenga aplicado al eje un mínimo de par de fricción.

Por encima de una frecuencia de 100Hz y hasta el límite de la velocidad del motor, las curvas de características ‘pull in' y ‘pull out' son más o menos uniformes y no suelen presentar este tipo de irregularidades, por lo que el motor se suele arrancar y parar a una frecuencia mínima de 100Hz, manteniendo el régimen de giro siempre por encima de ésta. No obstante estos efectos se pueden minimizar mediante volantes de inercia ‘dampers' acoplados al eje del motor.

Otra forma de solventar este problema es trabajar en medios pasos ‘half stepping' o mejor en micropasos, ya que en este tipo de operaciones el movimiento del rotor no es incremental paso a paso con saltos angulares bruscos, sino que es prácticamente lineal, eliminándose las resonancias mecánicas que causan los problemas de inestabilidad.

2.5.- Modos de excitación

Hasta ahora y con el único objetivo de simplificar las explicaciones, la excitación de los motores paso a paso siempre ha sido la misma. En cada paso del motor solamente una fase estaba excitada. Obviamente esto no tiene por qué ser así siempre. Según el número de fases que tenga el motor, la secuencia de éstas, necesaria para hacerlo girar, varía. Nos centraremos en un motor bifásico bipolar, ya que este tipo de motor es con el que se ha llevado a cabo la realización práctica de este proyecto. El término bipolar hace referencia al hecho de que la corriente por las bobinas de cada fase puede ser bidireccional dependiendo que pareja de interruptores estén abiertos o cerrados. La figura 2.19 servirá para comentar los diferentes modos de excitación de este tipo de motor.

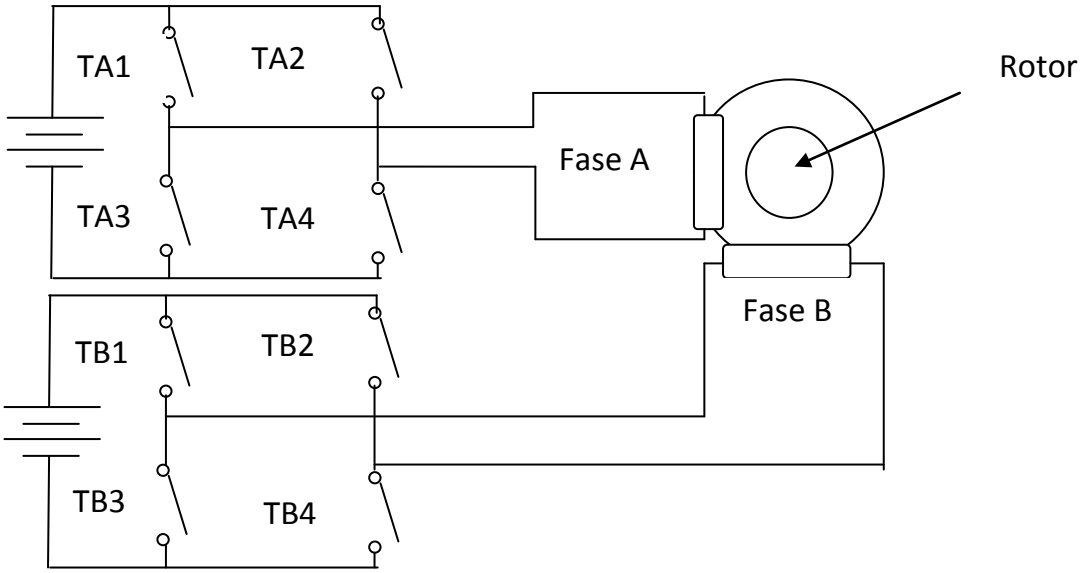


Figura 2.19-esquema base para comentar los modos de excitación

2.5.1.- Modo paso entero

2.5.1.1.- Fases excitadas alternativamente

En este modo de excitación, en cada secuencia de comunicación solamente una fase está excitada. Para realizar una secuencia completa es necesario realizar cuatro conmutaciones, en cada una de las cuales el motor se desplazará un ángulo de paso. Este modo de excitación suele recibir por esto el nombre de "Secuencia de 4 pasos". La figura 2.20 muestra esta secuencia.

El término "+" indica que la corriente por la fase circula en un determinado sentido y el término "-" indica que lo hace en sentido contrario. El término "off" indica que no circula corriente alguna por la fase.

Paso	Fase A	Fase B
1	-	Off
2	Off	-
3	+	Off
4	Off	+

Figura 2.20-Secuencia modo paso entero. Fases excitadas alternativamente

En este caso los dientes del estator y rotor están alineados para cada paso o posición.

2.5.1.2.- Fases siempre excitadas

En este modo de excitación después de cada conmutación siempre resultan estar excitadas las dos fases. La figura 2.21 muestra este modo de excitación. Como en el caso anterior, la secuencia completa se compone de cuatro conmutaciones en cada una de las cuales el motor gira un ángulo de paso.

Paso	Fase A	Fase B
1	-	-
2	+	-
3	+	+
4	-	+

Figura 2.21- Secuencia modo paso entero. Fases excitadas simultáneamente

En este caso los dientes de estator y rotor están desalineados medio paso en cada posición de equilibrio con respecto a cada posición de equilibrio alcanzada con el modo de excitación anterior. Esta diferencia es la base fundamental para realizar el modo de excitación que se expondrá a continuación, el modo medio paso.

Existen otras dos diferencias importantes entre estos dos modos de excitación. Una se refiere al par denominado anteriormente holding torque que puede proporcionar el motor. En este caso al estar siempre las dos fases excitadas el par resultante es mayor que en el caso anterior.

La otra diferencia estriba en las oscilaciones que se producen antes de alcanzar cada posición de equilibrio. Sin entrar con mayor profundidad en este tema diremos solamente que las oscilaciones son mucho menores en este caso que en el caso de que excitemos las fases alternativamente como resultado de los diferentes circuitos magnéticos que se producen en cada modo de excitación.

2.5.2.- Modo medio paso

Como su propio nombre indica, en este modo de excitación el motor se desplaza en cada conmutación la mitad del ángulo de paso. La secuencia de conmutación se basa en combinar las secuencias de los modos de excitación anteriores. La figura 2.22 muestra la secuencia para este modo de funcionamiento que necesita de ocho conmutaciones para completar una secuencia completa.

Paso	Fase A	Fase B
1	+	-
2	Off	-
3	-	-
4	-	Off
5	-	+
6	Off	+
7	+	+
8	+	Off

Figura 2.22-Secuencia modo medio paso

En las aplicaciones que utilicen este tipo de movimiento hay que tener en cuenta que el holding torque variará para cada paso ya que sólo se excitará una fase para una posición de paso, pero en el próximo paso se excitan las dos fases. Esto da el efecto de un paso fuerte y otro débil.

3.- Driver motor paso a paso

Los drivers que se han usado son los que se diseñaron para un proyecto anterior, realizado en 1997 por Felipe Carlos Zelaia, en el que se controlaba el mismo motor paso a paso. Están formados por tres tarjetas, dos de las cuales son las responsables de la conmutación de los MOSFETs del puente en H que controla la circulación de corriente por cada fase, mientras que la tercera tarjeta es la encargada de acondicionar las señales provenientes del encoder.

3.1.- Tarjetas conmutación MOSFETs

Se compone de dos tarjetas, en la figura 3.1 bajo el círculo rojo, que incorporan circuitos de control y potencia. Cada una de las tarjetas tiene como función el gobierno de una de las dos fases del motor paso a paso. Exteriormente difieren en que una de ellas lleva incorporada la conexión con el PC, pero se han diseñado de la misma manera y por lo tanto bastará con explicar el funcionamiento de una sola de ellas.

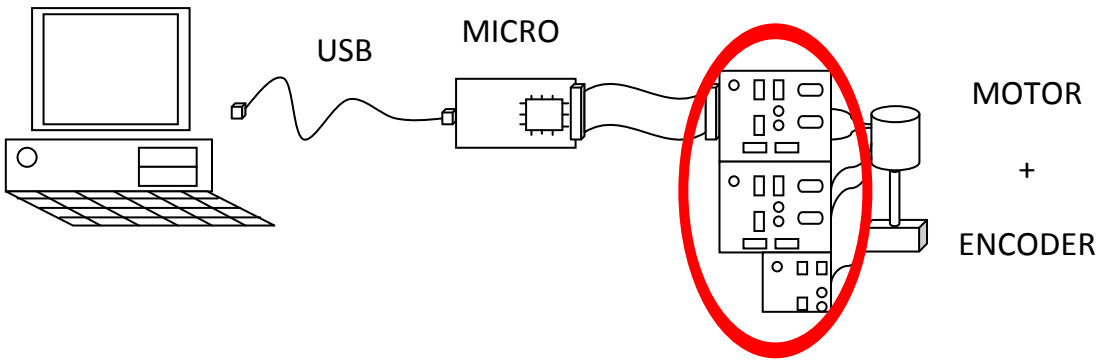


Figura 3.1 Sistema completo

Cada fase se encuentra conectada a la salida de un convertidor en puente de cuatro transistores MOSFET con encendidos y apagados controlados al objeto de excitar correctamente el motor. El control de corriente por lo devanados del motor se realiza por medio de la modulación PWM que realizan los transistores de las ramas inferiores a través de un circuito lógico.

Cada vez que se genera interrupción debido a que un contador interno del microprocesador ha finalizado su cuenta, se envían una serie de señales analógicas y digitales al driver del motor paso a paso. El intervalo entre la generación de dos interrupciones sucesivas determina de forma directa la velocidad de giro del motor. El modo en que se generan las interrupciones se explica con mayor precisión en el apartado dedicado al software. De momento sólo nos interesa saber que las tarjetas reciben señales tanto analógicas como digitales desde el microprocesador.

3.2.- Tarjeta encoder

Es la tarjeta encargada de acondicionar la salida que proporciona el encoder. Esta tarjeta tiene tres funciones:

- Acondicionar la señal que proporciona el encoder.
- Multiplicar por cuatro la resolución del encoder.
- Detectar el sentido de giro del motor paso a paso.

El encoder se alimenta con la misma tensión que el motor paso a paso, es decir, a 15 V. Este elemento proporciona a la salida dos señales entre 0 y 15 V desfasadas 90^{o} eléctricos entre sí. Los dos comparadores LM311, con sus correspondientes lazos de histéresis, se encargan de acondicionar estas dos señales a niveles TTL, de modo que se pueda atacar con estas señales los circuitos lógicos siguientes.

La resolución de este encoder es de 345 impulsos por revolución, pero tras pasar por la tarjeta se convierten en 1380 impulsos por revolución. Para conseguir esta mayor resolución, primeramente se realiza la función lógica XOR de las dos señales de salida del encoder, acondicionadas ya a niveles TTL, Con este simple paso, la resolución ya esta multiplicada por dos. El siguiente paso es realizar otra vez la función XOR utilizando como entradas esta misma señal anterior y un retraso adicional de esta señal en los flancos de subida y bajada a través de una red RC y dos inversores con Triger-Schimitt, incorporados para eliminar rebotes. La señal resultante se envía a través de un opto acoplador al microprocesador.

La última función de este circuito es detectar el sentido de giro del motor paso a paso. Una sola báscula Flip-Flop tipo D, capaz de detectar los flancos de subida, es la encargada de realizar esta función.

4.- Comunicación de datos USB 2.0

4.1.- Introducción

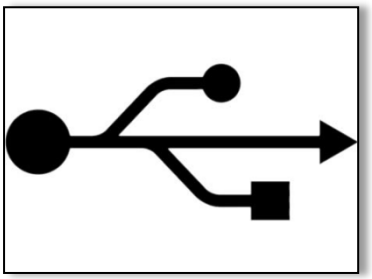


Figura4.1-USB 2.0.

El USB (Universal Serial Bus) es un puerto que sirve para conectar periféricos a un ordenador. Fue originalmente pensado para conectar dispositivos a los ordenadores, eliminando la necesidad de conectar tarjetas PCI (o similares), y también para conectar y desconectar dispositivos sin tener que reiniciar el ordenador.

El diseño del protocolo USB está a cargo del USB Implementers Forum (USB-IF), una organización compuesta por varias empresas de la rama de la computación y la electrónica, entre las que se encuentran Apple Computer, Hewlett-Packard, Microsoft e Intel.

Existen tres versiones del protocolo (1.0, 1.1 y 2.0). A diferencia de las anteriores, la última versión (2.0) soporta tasas de transferencia de altas velocidades, comparables (o incluso superiores) a la de un disco duro o almacenamiento magnético, lo cual ha permitido ampliar el uso del USB a aplicaciones de video y almacenamiento (discos duros externos). Una de las razones a la cual se atribuye su gran aceptación es que todas las versiones del protocolo son compatibles con las anteriores. Es decir, que cualquier dispositivo 2.0 puede ser conectado a un dispositivo 1.0, aunque funcionará a la velocidad del más lento.

Existen tres tipos de velocidades en la comunicación:

Tipo	Velocidad
Baja velocidad (low speed)	183 Kbytes/s (1.5Mbit/s)
Velocidad completa (full speed)	1.4 Mbytes/s (12Mbit/s)
Alta velocidad (high speed)	57 Mbytes/s (480 Mbit/s)

Figura 4.2-Velocidades de transferencia

4.2.- Topología

USB tiene un diseño asimétrico ya que consiste de un host controlador conectado a múltiples dispositivos conectados en daisy-chain (Esquema de cableado).

USB conecta varios dispositivos a un host controlador a través de cadenas de hubs. Los hubs (al igual que en redes) son dispositivos que permiten, a partir de un único punto de conexión, poder conectar varios dispositivos, es decir, disponer de varios puntos de conexión. De esta forma se crea una especie de estructura de árbol. El estándar admite hasta 5 niveles de ramificación por host controlador con un límite absoluto de 127 dispositivos conectados al mismo bus (incluyendo los hubs). Siempre existe un hub principal (conocido como el hub raíz) que está conectado directamente al host controlador.

Un mismo dispositivo USB puede cumplir varias funciones. Por ejemplo, un mouse puede ser también lector de tarjetas, y de esa forma sería como dos dispositivos conectados al bus USB. Por lo tanto es preferible hablar de funciones en lugar de dispositivos.

4.3.- Funcionamiento

Los dispositivos tienen asociados unos canales lógicos unidireccionales (llamados pipes) que conectan al host controlador con una entidad lógica en el dispositivo llamada endpoint. Los datos son enviados en paquetes de longitud variable. Típicamente estos paquetes son de 64, 128 o más bytes (64 bytes en el caso del software de este trabajo).

Estos endpoints (y sus respectivos pipes) son numerados del 0 al 15 en cada dirección, por lo que un dispositivo puede tener hasta 32 endpoints (16 de entrada y 16 de salida). La dirección se considera siempre desde el punto de vista del host controlador. Así un endpoint de salida será un canal que transmite datos desde el host controlador al dispositivo. Un

Figura4.4.-Conectores USB

endpoint solo puede tener una única dirección. El endpoint 0 (en ambas direcciones) está reservado para el control del bus.

Cuando un dispositivo es conectado al bus USB, el host controlador le asigna una dirección única de 7 bit (llamado proceso de enumeración) que es utilizada luego en la comunicación para identificar el dispositivo (o, en particular, la función). Luego, el host controlador consulta continuamente a los dispositivos para ver si tiene algo para mandar, de manera que ningún dispositivo puede enviar datos sin la solicitud previa explícita del host controlador.

Para acceder a un endpoint se utiliza una configuración jerárquica de la siguiente manera: un dispositivo/función conectado al bus tiene un único descriptor de dispositivo, quien a su vez tiene uno (o varios) descriptores de configuración. Estos últimos guardan generalmente el estado del dispositivo (ej: activo, suspendida, ahorro de energía, etc). Cada descriptor de configuración tiene uno (o más) descriptores de interfaz, y éstos a su vez tienen una configuración por defecto (aunque puede tener otras). Y éstos últimos finalmente son los que contienen los endpoint, que a su vez pueden ser reutilizados entre varias interfaces (y distintas configuraciones).

Como puede verse, la comunicación USB es bastante compleja y extremadamente más complicada que una simple comunicación serie.

4.4.- Tipos de transferencia

Los canales también se dividen en cuatro categorías según el tipo de transmisión:

- *Transferencias de control:* usado para comandos (y respuestas) cortos y simples. Es el tipo de transferencia usada por el pipe 0.
- *Transferencias isócronas:* proveen un ancho de banda asegurado pero con posibles pérdidas de datos. Usado típicamente para audio y video en tiempo real.
- *Transferencias interruptivas:* para dispositivos que necesitan una respuesta rápida (poca latencia), por ejemplo, mouse y otros dispositivos de interacción humana.

- *Transferencias masivas:* para transferencias grandes y esporádicas utilizando todo el ancho de banda disponible, pero sin garantías de velocidad o latencia. Por ejemplo, transferencias de archivos.

En realidad las transferencias interruptivas no son tales ya que los dispositivos no pueden enviar datos sin recibir autorización del host controlador. Por lo tanto, las transferencias interruptivas simplemente le dan más prioridad al sondeo del host controlador.

4.5.- Señalización y conectores

Las señales USB son transmitidas en un par trenzado (cuyos hilos son denominados D+ y D-) utilizando señalización diferencial half-duplex, minimizando el ruido electromagnético en tramos largos. El diseño eléctrico permite un largo máximo de 5 metros (sin precisar un repetidor intermedio).

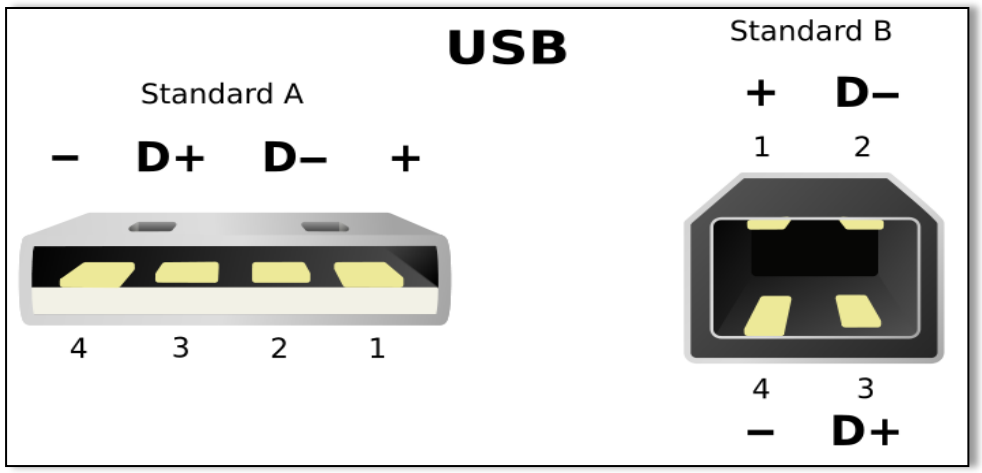


Figura 4.5- Conector estándar A y conector estándar B

Existen dos tipos de conectores: estándar y mini. Los estándares son los que típicamente encontramos en un ordenador y vienen en dos tipos: A y B. El tipo A es el que es plano y se encuentra del lado del host controlador, mientras que el tipo B es el cuadrado y se encuentra del lado del dispositivo. Todos los cables son machos, mientras que los enchufes (ya sea en el ordenador o los dispositivos) son hembra. No existen intercambiadores de género puesto que las conexiones cíclicas no están permitidas en un bus USB.

4.6.- Potencia

El bus USB suministra 5V de continua regulados por cada uno de sus puertos, entre los pines 1 y 4. Por lo tanto, dispositivos de bajo consumo de potencia (que de otra forma vendría con una fuente de alimentación) puede obtener de allí la corriente necesaria para el funcionamiento. El límite de corriente suministrada es de 500mA por cada puerto. Además, el estándar exige no más de 5.25V en ningún caso, ni menos de 4.375V en el peor caso. Típicamente el voltaje se mantiene en los 5V.

Algunos hubs se alimentan directamente del bus USB, en cuyo caso la corriente total de todos los dispositivos conectados a él no puede superar los

500mA. Sin embargo, la especificación permite solo un nivel de hub alimentados por el bus, de forma que no es posible conectado un hub sin alimentación a otro hub sin alimentación. Los hubs con alimentación propia no tienen esta restricción y generalmente son necesarios para conectar dispositivos de alto consumo como impresoras o discos duros.

Cuando un dispositivo es conectado le reporta al host controlador cuando potencia va a consumir. De esta manera el host controlador lleva un registro de los requisitos de cada puerto y generalmente cuando un dispositivo se excede generalmente se apaga, cortándole el suministro de corriente, de forma de no afectar al resto de los dispositivos. El estándar exige que los dispositivos se conecten en un modo de bajo consumo (100 mA máximo) y luego le comuniquen al host controlador cuanta corriente precisan, para luego cambiar a un modo de alto consumo (si el host se lo permite).

Los dispositivos que superen los límites de consumo deben utilizar su propia fuente de alimentación.

Los dispositivos que no cumplan con los requisitos de potencia y consuman más corriente del host de la prevista pueden dejar de funcionar sin previo aviso, en algunos casos, dejar todo el bus inoperativo.

4.7.- Futuro del USB

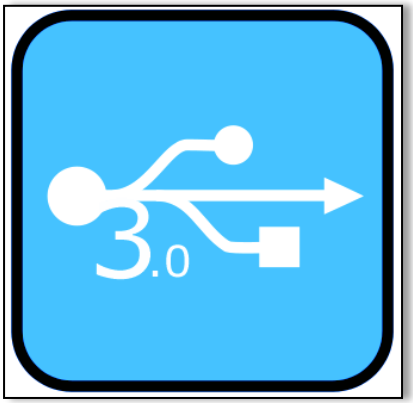


Figura4.3-USB 3.0

Las empresas de semiconductores están haciendo un gran esfuerzo en reducir los costos de los componentes mediante la integración de varias funciones de estos dispositivos en un solo chip, con la consiguiente reducción de la cantidad de partes y, sobre todo, del costo total.

Actualmente se estan tratando de desarrollar en dichos lugares los dispositivos flash a una velocidad mayor gracias al futuro puerto USB 3.0.

La principal novedad técnica del puerto USB 3.0. será que eleva a 4.8 gigabits/s la capacidad de transferencia que en la actualidad es de 480 Mb/s. Se mantendrá el cableado interno de cobre para asegurarse la compatibilidad con las tecnologías USB 1.0 y 2.0.

Si en USB 2.0 el cable dispone de cuatro líneas, un par para datos, una de corriente y una de toma de tierra, en USB 3.0 se añade cinco líneas. Dos de ellas se usarán para el envío de información y otras dos para la recepción, de forma que se permite el tráfico bidireccional, en ambos sentidos al mismo tiempo. El aumento del número de líneas permite incrementar la velocidad de transmisión desde los 480 Mb/s hasta los 4,8 Gb/s. De aquí se deriva el nombre que también recibe esta especificación: USB Superspeed.

5.- Microcontrolador PIC18F4550

5.1.- Introducción al PIC18F4550

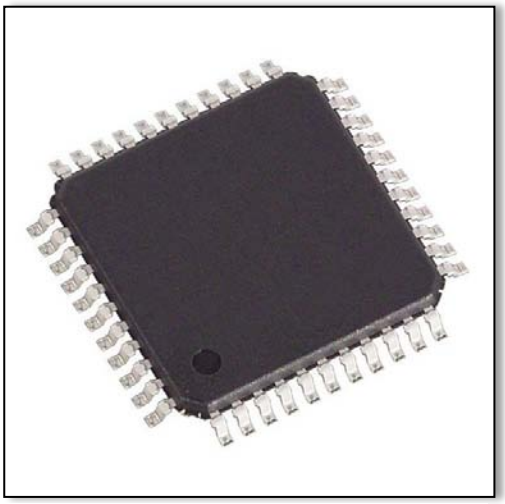


Figura 5.1 PIC18F4550 44-PINS

El PIC18F4550 es un microprocesador enfocado a aplicaciones de baja potencia (nanoWatt) y elevada conectividad. Dispone de 3 puertos serie: FS-USB(12Mbit/s), I²C™ y SPI™ (hasta 10 Mbit/s) y un puerto serie asíncrono (EUSART). También dispone de una elevada memoria RAM para almacenamiento en búfer y de mejorada memoria flash, lo que lo hace perfecto para aplicaciones de control y vigilancia que requieren de una conexión periódica a un ordenador personal a través del puerto USB, desde dónde se podrá realizar una carga y descarga de datos.

Pertenece a la familia de procesadores PICmicro de la empresa norteamericana Microchip cuya sede se ubica en Chandler, Arizona (USA).

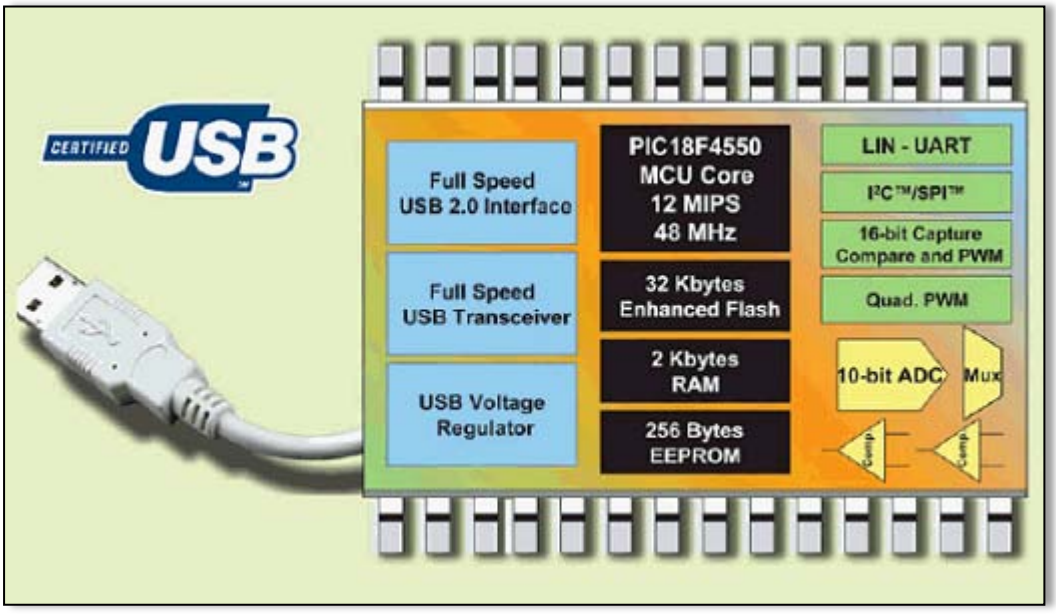


Figura 5.2

Lo particular del procesador PIC18F4550 es que es uno de los PICs que viene con soporte nativo para USB, lo cual quiere decir que incluyen un controlador USB

interno que dispone de patas de salida para conectar directamente al ordenador, sin la necesidad de pull-ups o ninguna circuitería externa.

Soporta cristales y osciladores de varias frecuencias como entrada y tiene post-scaler de manera que el procesador pueda trabajar a una frecuencia de 48 MHz, independiente del oscilador que se conecte. Para ello debe configurarse(a través de los configuration bits) el oscilador que se le ha conectado. Trabajar a 48 Mhz es un requisito para poder transferir a full-speed por el puerto USB. El controlador USB, por lo tanto, transfiere a full-speed (1.5 Mbytes/seg) por USB y es compatible con el estándar USB 2.0.

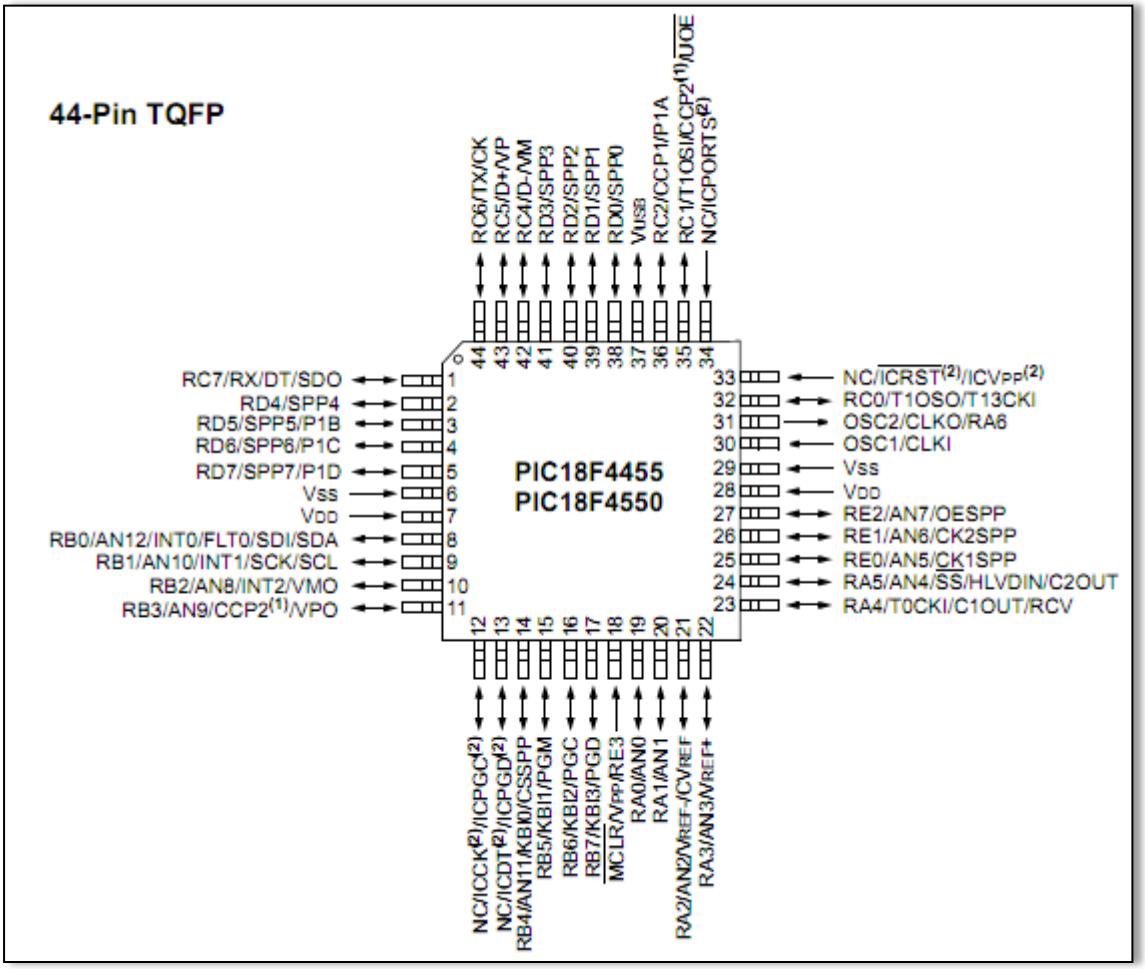


Figura 5.3

También cuenta con 35 patas de entrada/salida digitales de propósito general y viene disponible en varios empaquetados. Los puertos de entrada/salida son todos compatibles con la tecnología TTL. Cuando se los utiliza como salida, se comporta como un CMOS, siendo compatible con TTL, pudiendo manejar cualquier tipo de tecnología. Sin embargo cuando son configurados los puertos

como entrada, hay dos comportamientos posibles: puede ser exclusivamente TTL, o puede ser configurado para TTL o CMOS.

En cuanto a memoria, posee 32Kb de flash para almacenamiento de programas, 2Kb de SRAM para memoria volátil, y 256 bytes de EEPROM (memoria no-volátil) para almacenamiento permanente de datos como configuraciones y demás.

Otras características interesantes que posee son timers, interrupciones (externas e internas por timers) con dos niveles de prioridad y disparadas tanto por nivel como por flanco, un comparador analógico con un generador de voltaje de referencias de 16 niveles.

5.2.- Funciones del microprocesador

El microprocesador 18F4550 realiza una serie de funciones que vamos a explicar a continuación de manera simple, más adelante las explicaremos detenidamente.

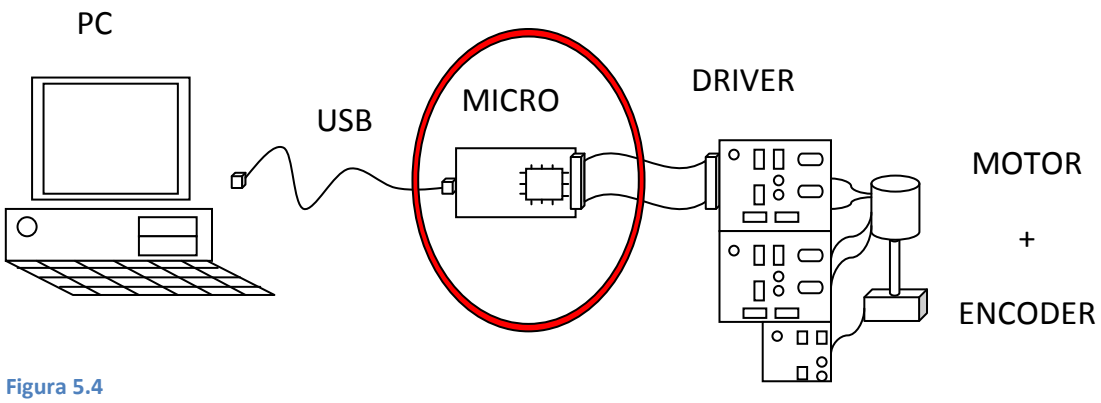


Figura 5.4

El microprocesador es el "puente" entre el ordenador y los drivers del motor paso a paso, el encargado de que las ordenes provenientes del ordenador lleguen eléctricamente a los drivers, al igual que de trasmitir la información del encoder al ordenador.

El ordenador se comunica con el procesador mediante el protocolo USB 2.0, mientras que con los drivers la comunicación se realiza con un bus de cables en paralelo.

Las funciones del microprocesador en el conjunto del sistema las enumeramos a continuación:

- Enviar y recibir información del ordenador, lo que supone que en todo momento debe de estar "escuchando". Para ello utiliza el puerto USB del que dispone.
- Enviar y recibir información de los drivers a través del bus paralelo que los comunica. Este conjunto de señales las componen señales digitales de entrada y salida, las cuales detallaremos en el próximo apartado.
- Trabajar con interrupciones y contadores internos para el correcto control de todas las señales con las que trabaja.
- Realizar operaciones algebraicas internas para la correcta aceleración y deceleración del motor paso a paso.

5.3.- Comunicación USB con el ordenador

El microprocesador 18F4550, como se ha mencionado anteriormente, está

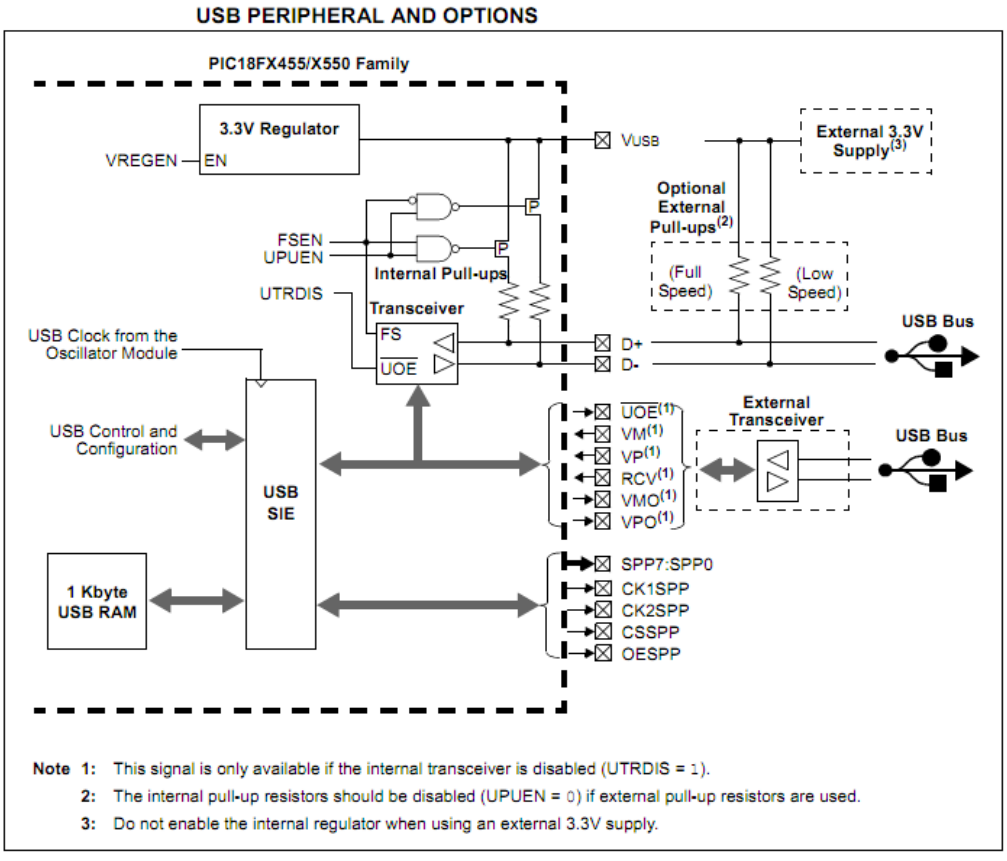


Figura 5.5

preparado para comunicarse mediante el protocolo USB 2.0. Puede trabajar a baja (1.5Mb/s) o alta (12Mb/s) velocidad, también soporta Interrupciones. Cada vez que se produce una transmisión o recepción de datos en el bus, se genera una interrupción en el PIC ante la cual la rutina de atención debe responder gestionando todos los aspectos de bajo nivel de la especificación USB. De esta manera para la aplicación principal que ejecuta el Microcontrolador el manejo del protocolo USB es transparente.

En nuestro caso disponemos de un cristal externo de 20 MHz, a partir del cual configuraremos el reloj del microprocesador a 48 MHz, con la finalidad de trabajar con el USB a alta velocidad.

Permite el uso de hasta 32 EndPoints (16 bidireccionales). Un endpoint es un buffer que almacena datos dentro del dispositivo (Típicamente es un registro). A su vez cada endpoint dispone de un identificador único que viene asignado de fábrica y una determinada orientación del flujo de datos (IN/OUT). Todos los dispositivos deben soportar el endpoint 0, que se usa para la configuración.

Para facilitar la configuración del módulo USB del microprocesador, MicroChip aporta las librerías necesarias para su utilización.

5.4.- Señales de salida para el control de los drivers

Las señales analógicas son las que indican a los drivers la consigna de corriente que deseamos que circule por cada fase, al ser dos fases, son dos consignas, ya que podremos mandar diferentes corrientes a cada fase.

Las señales digitales son las encargadas de controlar el puente en H (circuito electrónico que permite el control de la corriente en cada fase en ambos sentidos) de cada unas de las fases. Para ello se necesitarán las cuatros señales mencionadas arriba, dos por cada fase. En el apartado de señales digitales de salida detallamos su funcionamiento.

5.4.1.- Señales analógicas

El microprocesador 18F4550 sólo dispone de una salida analógica, mientras que el driver necesita dos para su correcto control..

La solución fue crear dos señales PWM (señal Modulada en Ancho de Pulso) de salida en el microprocesador e introducir entre este y los drivers un filtro RC paso bajo y un seguidor de tensión para cada una de estas señales PWM, con el fin de obtener las señales analógicas correctas para el driver.

En primer lugar para obtener las señales PWM, se ha utilizado una de las librerías que nos trae MicroChip (pwm.h), ello permite simplificar la programación de la función que controla el PWM.

PWM FUNCTIONS

Function	Description
`ClosePWMx`	Disable PWM channel x.
`OpenPWMx`	Configure PWM channel x.
`SetDCPWMx`	Write a new duty cycle value to PWM channel x.
`SetOutputPWMx`	Sets the PWM output configuration bits for ECCP x.
`CloseEPWMx`(1)	Disable enhanced PWM channel x.
`OpenEPWMx`(1)	Configure enhanced PWM channel x.
`SetDCEPWMx`(1)	Write a new duty cycle value to enhanced PWM channel x.
`SetOutputEPWMx`(1)	Sets the enhanced PWM output configuration bits for ECCP x.

Figura 5.6

Las funciones utilizadas en el programa del microprocesador son:

- *OpenPWM1 () & OpenPWM2 ():* Configura el canal PWM con el periodo deseado.
- *SetDCPWM1 () & SetDCPWM2 ()* : Determina el ancho de pulso positivo de nuestro PWM.
- *ClosePWM1 () & ClosePWM2()* : Deshabilita las señales PWM.

Las señales PWM utilizan el timer2 del Microcontrolador, por lo que antes de habilitar las señales PWM, el timer2 debe de estar preparado OpenTimer2 ();

El valor a introducir en *OpenPWM (period)* no es directamente el valor del periodo, es necesario resolver esta ecuación:

PWM period =[(**period**) + 1] x 4 x Tosc x TMR2 prescaler

- ✓ *PWM period*: Periodo de la señal que buscamos.
- ✓ *Period*: Valor a introducir en la función.
- ✓ *Tosc*: La inversa de nuestra frecuencia de oscilación (1/48Mhz).
- ✓ *TMR2 prescaler*: Pre-escalado asignado al abrir el timer2, 16 en nuestro caso

El valor máximo de *period* es 0xff (255), luego la frecuencia no podrá bajar de unos 3000 Hz, a penas que se modifique el prescaler del timer2.

Para conocer el valor a introducir en *SetDCPWM (dutycycle)* se debe despejar la siguiente ecuación:

Donde:

- ✓ *PWM(ton):* Tiempo en "on" deseado
- ✓ *Dutycycle:* Valor a introducir en la función.
- ✓ *Tosc:* La inversa de nuestra frecuencia de oscilación (1/48Mhz).
- ✓ *TMR2 prescaler*: Pre-escalado asignado al abrir el timer2, 16 en nuestro caso

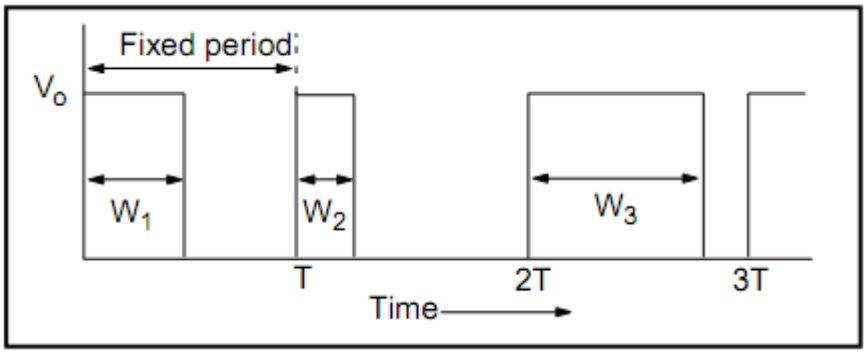


Figura 5.7

En este punto ya nos encontramos listos para obtener las dos señales moduladas en ancho de pulso, ahora es el momento de realizar el filtro para obtener las dos señales analógicas que buscamos, las cuales deben estar entre 0 y 5v

5.4.1.1.- Filtro paso bajo y seguidor de tensión

Para conseguir las dos señales analógicas que buscamos para el control de la corriente por las fases se han construido dos filtros paso bajo RC de primer orden y sendos seguidores de tensión, dónde la ganancia de tensión es 1.

La gráfica izquierda de la figura 5.8 muestra la respuesta en frecuencia ideal y real de un filtro paso bajo. Este tipo de filtro deja pasar todas las frecuencias desde cero hasta la frecuencia de corte y bloquea todas las frecuencias por encima de la misma.

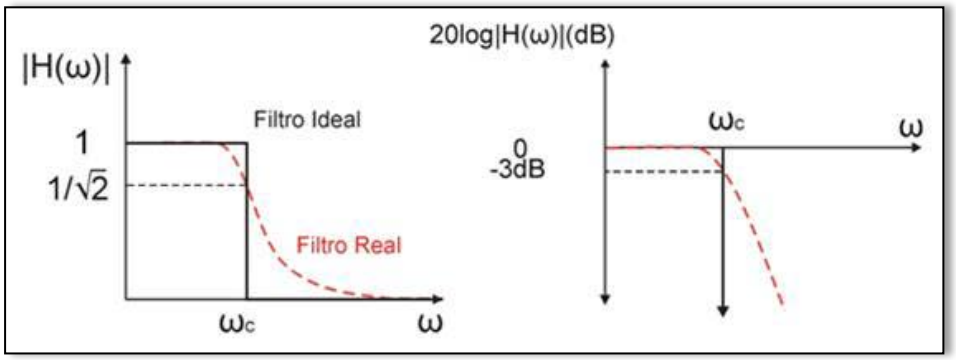


Figura 5.8

En los filtros paso bajo, las frecuencia entre cero y la frecuencia de corte se llaman *banda pasante*. Las frecuencias por encima de la frecuencia de corte son la *banda eliminada*. La zona entre la banda pasante y la banda eliminada se llama *región de transición*. Un filtro paso bajo ideal tiene atenuación cero (señal perdida) en la banda pasante, infinita en la banda eliminada y una transición vertical.

Una indicación más: el filtro paso bajo ideal no produce desfase en todas las frecuencias de la banda pasante. La ausencia de desfase es importante cuando la señal de entrada no es sinusoidal. Cuando un filtro tiene desfase cero, se mantiene la forma de una señal no sinusoidal cuando ésta lo atraviesa. Por ejemplo si la señal de entrada es una onda cuadrada, tiene una frecuencia fundamental y armónicos (como es nuestro caso). Si la frecuencia fundamental y los armónicos más significativos (aproximadamente los diez primeros) están dentro de la banda pasante, la onda cuadrada tendrá aproximadamente la misma forma de salida.

La frecuencia de corte, que es aquella frecuencia para el cual la amplitud de la señal de entrada se atenua 3 dB, viene dado por:

$$\frac{}{}$$

Cuando aumenta la frecuencia por encima de la frecuencia de corte, la reactancia capacitiva disminuye y reduce la tensión en la entrada no inversora. Como el circuito de retardo RC está fuera del lazo de realimentación, la tensión de salida decae. Cuando la frecuencia se aproxima a infinito, el condensador se

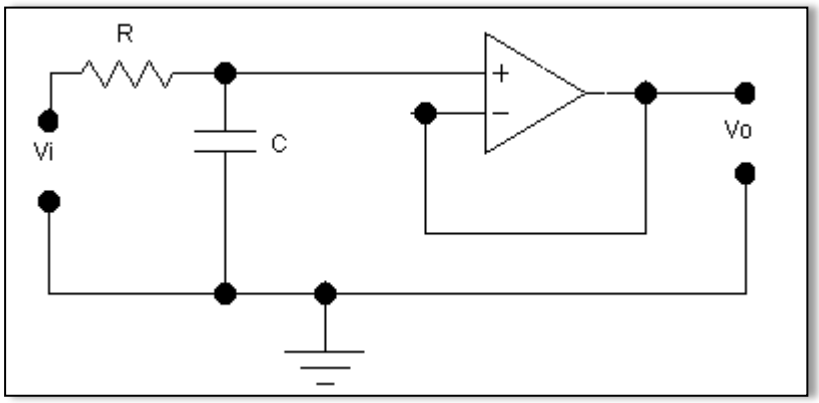
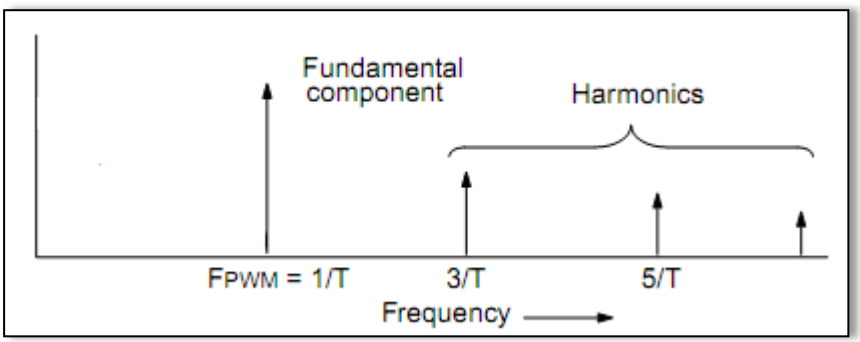


Figura 5.8.1

aproxima al corte, con lo que su tensión de entrada es cero.

En el caso particular que nos ocupa tenemos como entrada una señal PWM a una frecuencia constante y conocida, se busca eliminar todos los armónicos, excepto el de continua, que componen la señal.

Un estudio del análisis de Fourier para una señal PWM típica, nos muestra que existe un pico de frecuencia en Fpwm=1/T, mientras que el resto de amónicos aparecen en frecuencias=K/T (K valores impares), estos picos son ruidos no deseados y deben de ser eliminados.



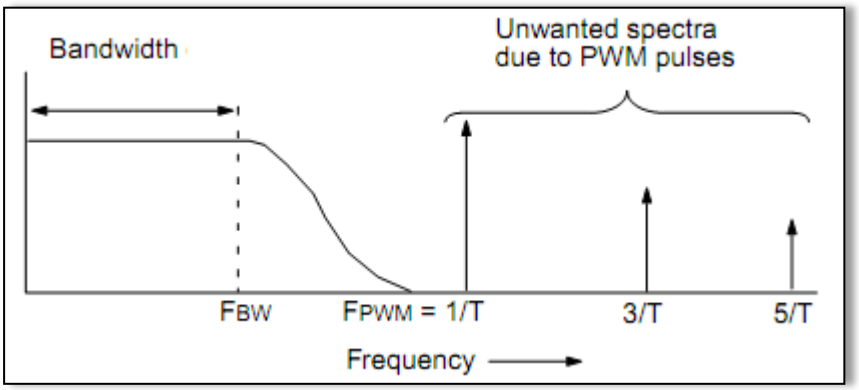
Armónicos de una señal PWM

A partir de esta información se ha escogido un R= 16,5k y un C=10nF, se ha buscado una elevada impedancia para que el consumo de corriente sea lo más pequeño posible.

Con ello obtenemos una atenuación en el armónico principal de -18dB, suficiente para nuestra aplicación.

= -18dB

De esta forma las dos señales analógicas se encuentran listas para el control de las corrientes por las dos fases del motor paso a paso



Figuro 5.8.2

5.4.2.- Señales digitales

Las salidas digitales del microprocesador 18F4550 son las encargadas de controlar los MOSFETs del puente en H de cada fase del motor.

La explicación se realizará solamente de una de las fases, la A en este caso, ya que la fase B es idéntica.

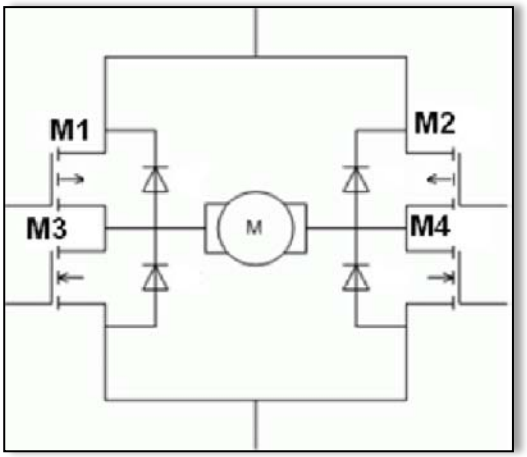


Figura 5.9

Cada tarjeta recibe un total de dos señales digitales denominadas PHASE_A y INHIBIT_A para la Fase A y PHASE_B y INHIBIT_B para la Fase B.

INHIBIT_A	PHASE_A	M1	M2	M3	M4
0	0	-	-	-	-
0	1	-	-	-	-
1	0	Off	On	PWM	Off
1	1	On	Of	Off	PWM

Figura 5.10-Tabla de la verdad fase A 1

M1, M2 son transistores PMOS mientras que M3 y M4 son NMOS. El hecho de que estén encendidos M1 y M4 supone que la corriente por la fase del motor circula de A a B. Este sentido es el contrario si los transistores MOSFET encendidos son M2 y M3. Por razones de seguridad nunca van a estar encendidos simultáneamente M1 y M3, ni M2 y M4.

Cuando los estados de las señales INHIBIT_A y PHASE_A permiten el encendido de cualquiera de los dos transistores MOSFET de la rama inferior, M3 o M4, éstos no lo están permanentemente, sino que conmutan a frecuencias de kilohercios con el fin de que por la fase del motor circule la consigna de corriente adecuada.

5.4.3.- Aceleración y deceleración

Con el objeto de no perder el sincronismo se alcanza la velocidad final tras un breve periodo de aceleración. En este apartado se explica cómo se ha realizado la aceleración y deceleración de motor.

La aceleración y deceleración es lineal, el algoritmo se basa en el que se expone en "Embedded Systems Programing" en Enero de del 2005 "Generate stepper-motor speed profiels in real time", un articulo del D. Austin. Este algoritmo permite una parametrización y cálculo en tiempo real, mediante la utilización de una operación aritmética de punto-fijo, sin la necesidad de tablas, adaptado específicamente al PIC.

5.4.3.1.- Ecuaciones del movimiento del motor paso a paso

Para crear el movimiento de rotación de un motor paso a paso, la corriente a través de las bobinas debe cambiar en el orden prefijado. Esto se logra utilizando un controlador que permite la correcta secuencia de salida cuando se somete a un pulso ("el pulso motor paso a paso") y una dirección de señal.

Para girar el motor paso a paso a una velocidad constante, los pulsos deben ser generados a un ritmo constante, que se muestra en la figura.

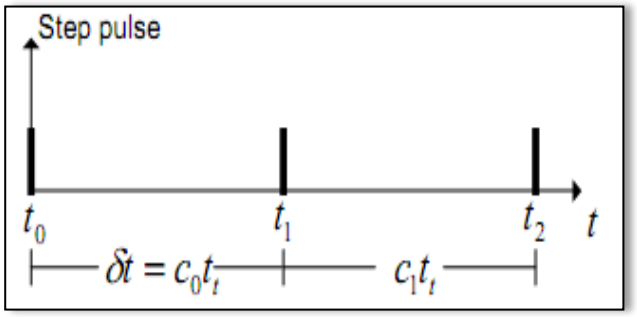


Figura 5.11-Pulsos motor paso a paso

Un contador genera estos pulsos a una frecuencia ft [Hz]. El retardo ðt definido por el contador *c es:*

$$\delta t = c t_t = \frac{c}{f_t} \text{ [s]}$$

El ángulo de paso del motor α, posición θ, y la velocidad ω son dados por:

$$\alpha = \frac{2\pi}{spr}\ [\text{rad}] \qquad \theta = n\alpha\ [\text{rad}] \qquad \omega = \frac{\alpha}{\delta t}\ [\text{rad/sec}]$$

Donde *spr* es número de pasos por vuelta, n el número de pasos, y 1 rad/seg = 9.55 rpm

5.4.3.2.- Rampa de velocidad lineal

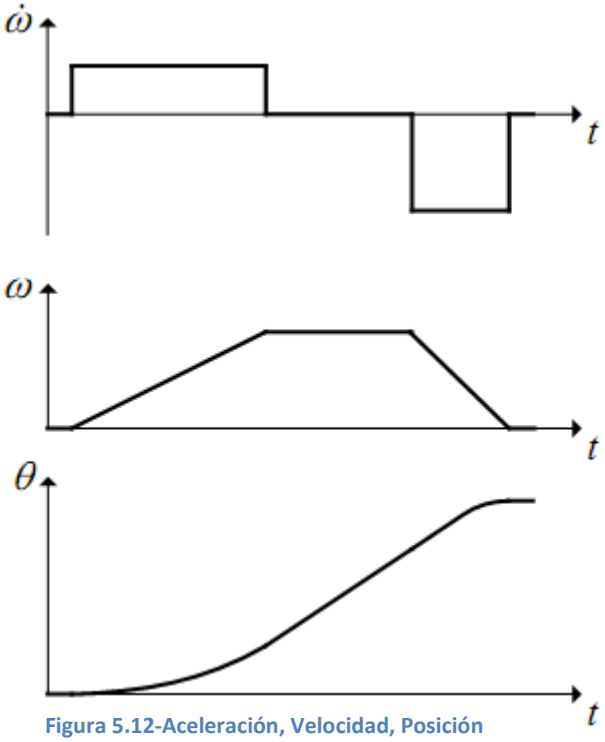


Figura 5.12-Aceleración, Velocidad, Posición

Para arrancar y detener el motor suavemente, es necesario un control de la aceleración y deceleración. Las siguientes figuras nos muestran la relación entre aceleración, velocidad y posición. Usando una aceleración/deceleración constante se obtiene un perfil de velocidad lineal.

El tiempo de retardo entre los impulsos del motor paso a paso controla la velocidad. Este tiempo de retardo debe de ser calculado con el fin de conseguir que la velocidad siga la rampa de velocidad lo más exactamente posible.

Los pasos discretos controlan el movimiento del motor paso a paso, y la

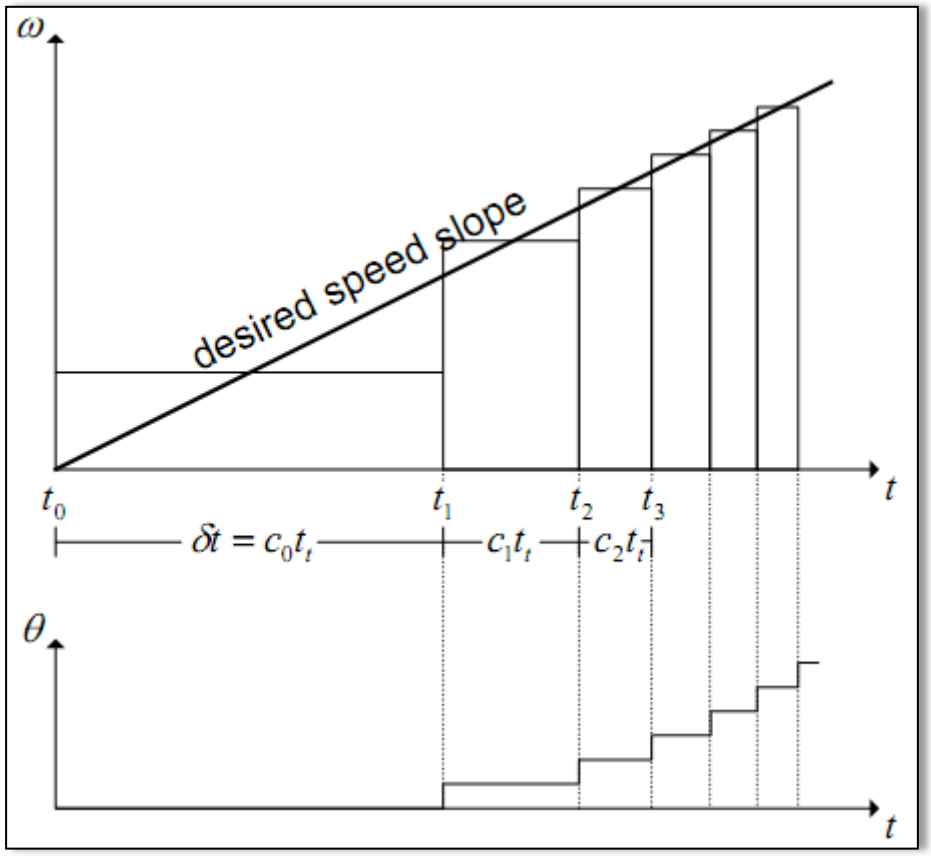


Figura 5.13-Perfil de velocidad- Pulsos/velocidad motor paso a paso

resolución de retardo de tiempo entre pasos está dado por la frecuencia del temporizador.

5.4.3.3.- Cálculo exacto del tiempo entre pasos

El primer retardo del contador c_0, así como lo sucesivos retardos c_n, están dados por las ecuaciones:

$$c_0 = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}} \qquad c_n = c_0\left(\sqrt{n+1} - \sqrt{n}\right)$$

La capacidad de cálculo de un microprocesador es limitada, y el cálculo de dos raíces cuadradas supone un consumo de recursos inncesario. Por lo tanto se busca una aproximación con menos gasto computacional.

El valor del contador en el tiempo n, usando aproximación por series de Taylor para el retardo de entre-paso es dado por:

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}$$

Este cálculo es mucho más rápido que la raíz cuadrada doble, pero introduce un error de 0,44 en n=1. La forma de corregir este error es multiplicar c_0 por 0.676.

5.5.- Entorno de trabajo con el microprocesador

Previamente a mostrar el código y una explicación de este, resulta imprescindible una pequeña introducción al entorno de trabajo del microprocesador, el cual está compuesto con el ensamblador MPLAD IDE y el compilador, MPLAB C18.

5.5.1.- MPLAB IDE

Ensamblador, enlazador, gestión de proyectos, depurador y simulador. La interfaz gráfica del usuario MPLAB IDE sirve como un único entorno para escribir, compilar y depurar código para aplicaciones embebidas. Permite manejar la mayoría de los detalles del compilador, ensamblador y enlazador, quedando la tarea de escribir y depurar la aplicación como foco principal del programador (usuario).

5.5.2.- MPLAB C18

MPLAB C18 es un compilador cruzado que se corre en un PC y produce código que puede ser ejecutado por la familia de microcontroladores de Microchip PIC18XXXX. Al igual que un ensamblador, el compilador traduce las declaraciones humanas en unos y ceros para ser ejecutados por el microcontrolador. La programación del microprocesador 18F4550 se ha realizado en el lenguaje de programación ANSI C.

Posee librerías para comunicaciones SPI, I2C, UART, USART, generación PWM, cadena de caracteres y funciones matemáticas de coma flotante.

Maneja números reales de 32 bits (float y double)

5.5.2.1.- Creación de un nuevo proyecto

Project - New

Nos aparecerá una pantalla donde le indicamos el nombre de nuestro proyecto y la carpeta donde será guardado.

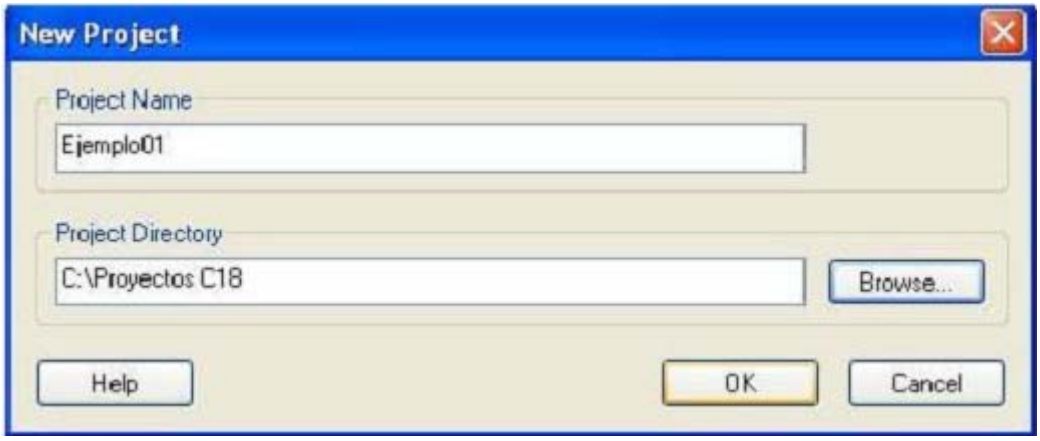
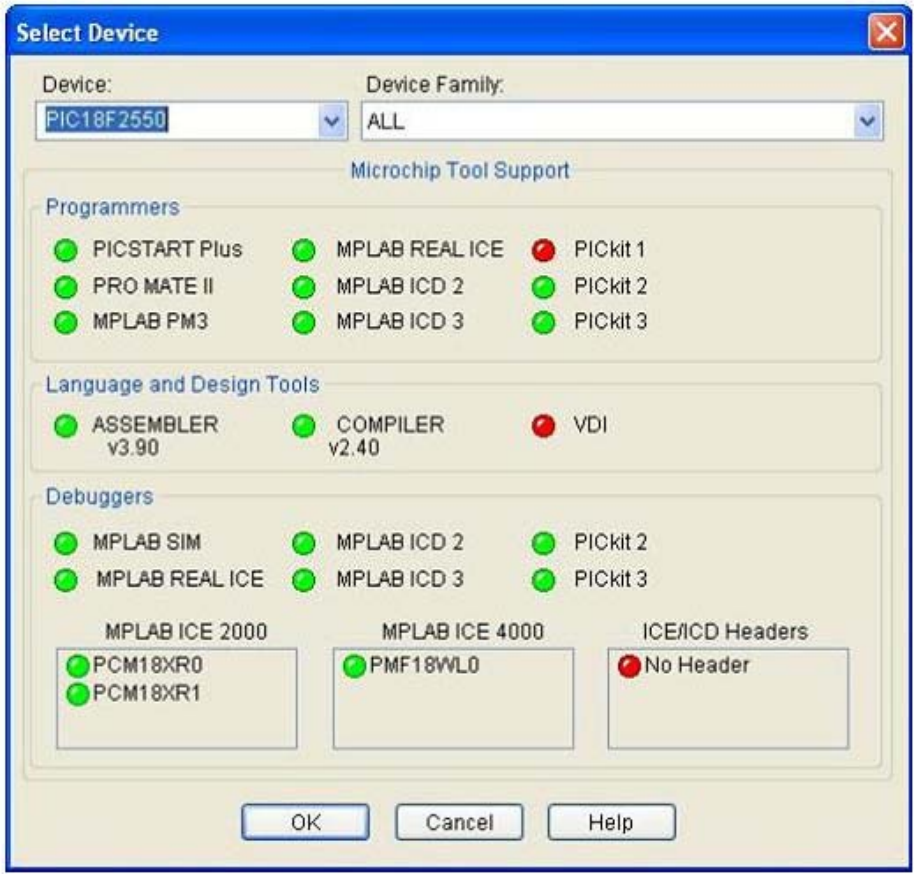


Figure 5.14

Pasamos a configurar el dispositivo con el cual trabajaremos:

Configure - Select Device



Seleccionamos el compilador:

Project - Select Lenguaje Toolsuite y nos aseguramos que todas las direcciones son correctas.

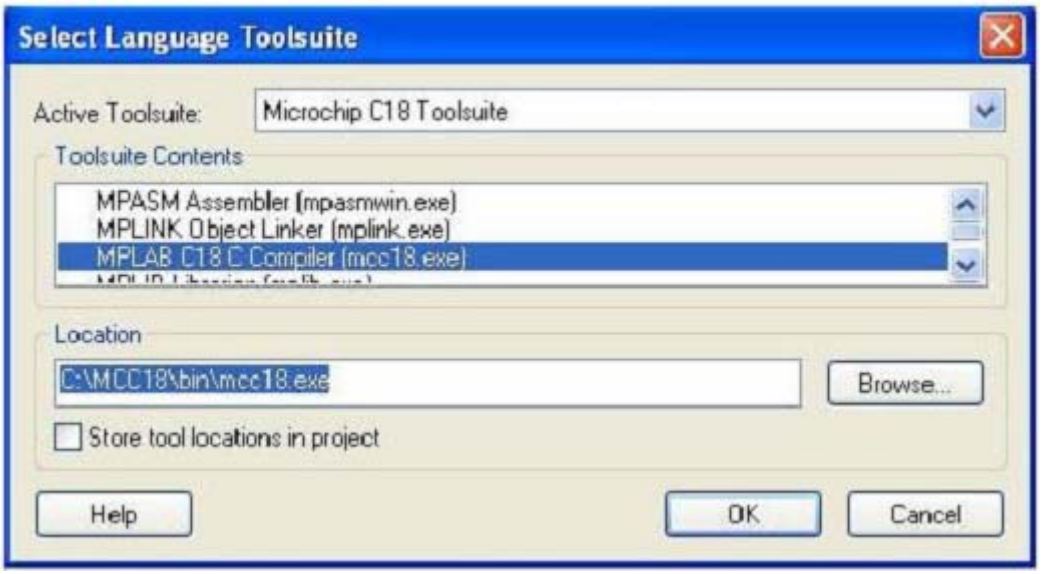


Figure 5.15

Configuramos los subdirectorios de trabajo:

Project - Build options - Project

Seleccionamos ubicación deficheros de declaraciones, bibliotecas y script de enlazado.

- Show directories for:
- Include Search Path
- Library Search Path
- Linker-Script Search Path

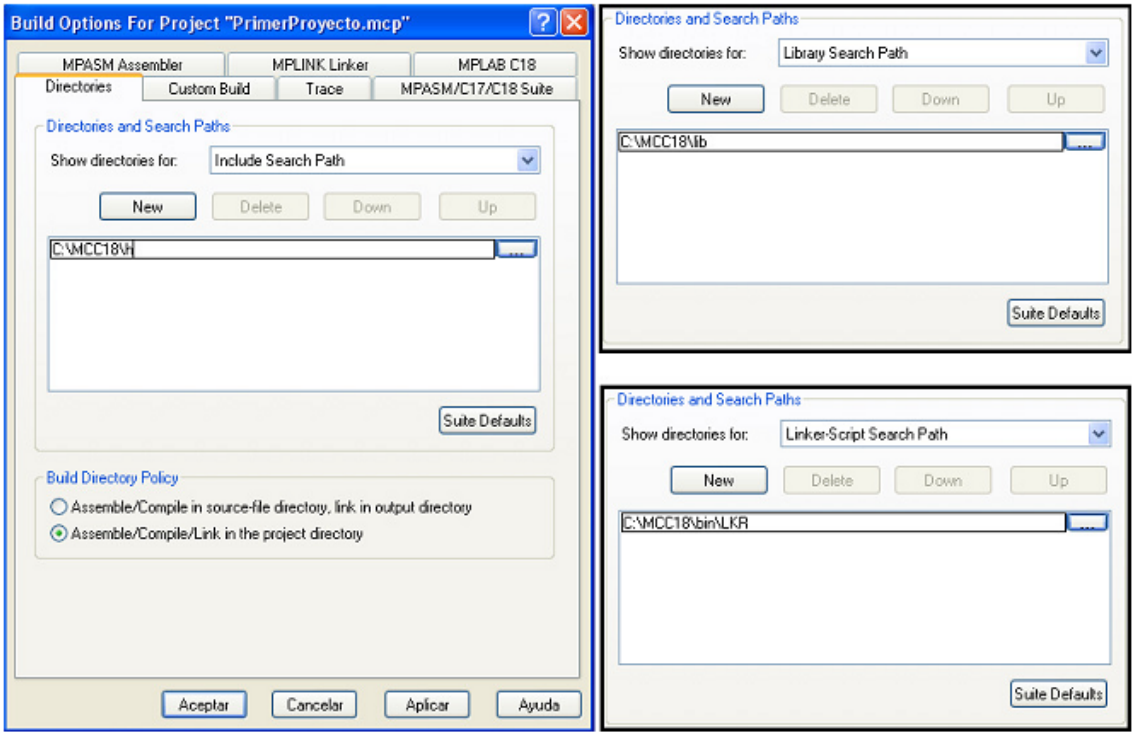


Figure 5.15

También podemos crear carpetas como Output y Objects para organizar en ellos los archivos de salida e intermedios generados en el proceso de compilación.

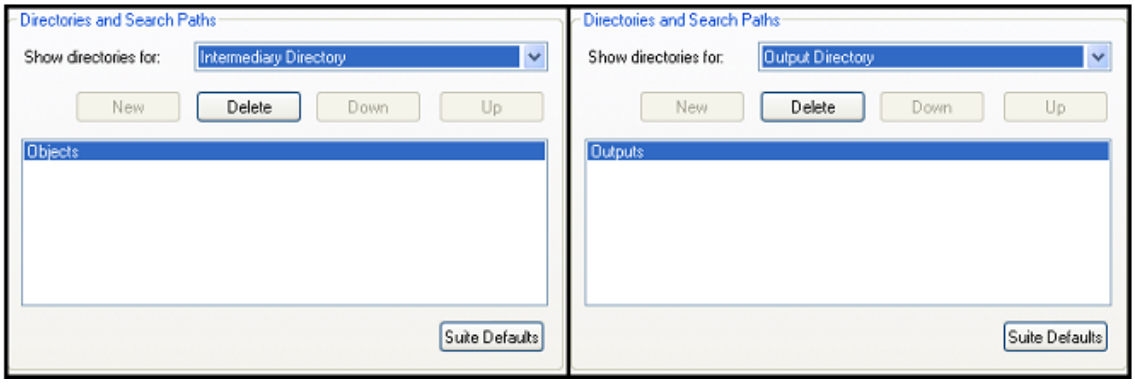


Figure 5.16

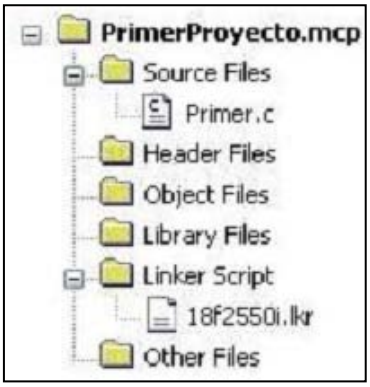


Figure 5.17

Luego vamos a New File y lo guardamos en nuestra carpeta eligiendo extensión .c agregándolo a nuestro proyecto.

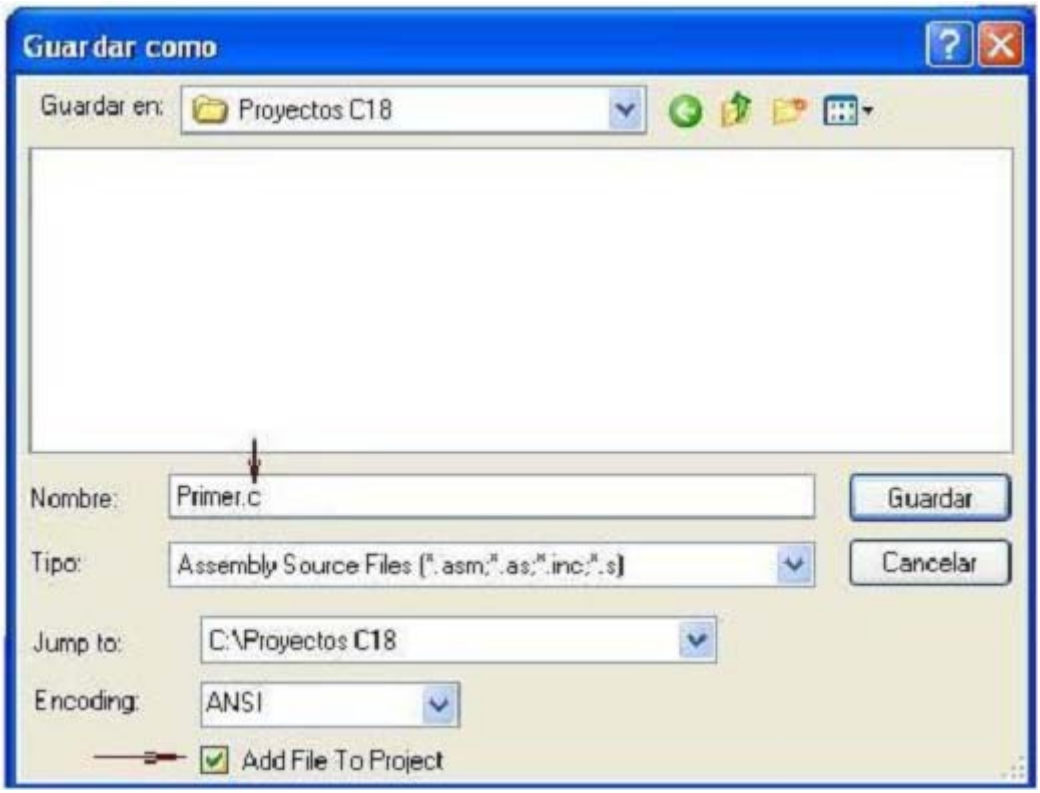


Figure 5.18

Con todo esto configurado ya podemos empezar a desarrollar nuestro código.

5.6.- Código C en el microprocesador

Ahora que ya se ha realizado una breve introducción al entorno de trabajo ha llegado el momento de explicar en detalle el código con el que se ha desarrollado el microprocesador de control del motor paso a paso, el que lo configura y hace trabajar de acuerdo a nuestras necesidades

El proyecto creado para nuestro sistema ha recibido el nombre de MCHPUSB.mcp y está dividido en diferentes carpetas.

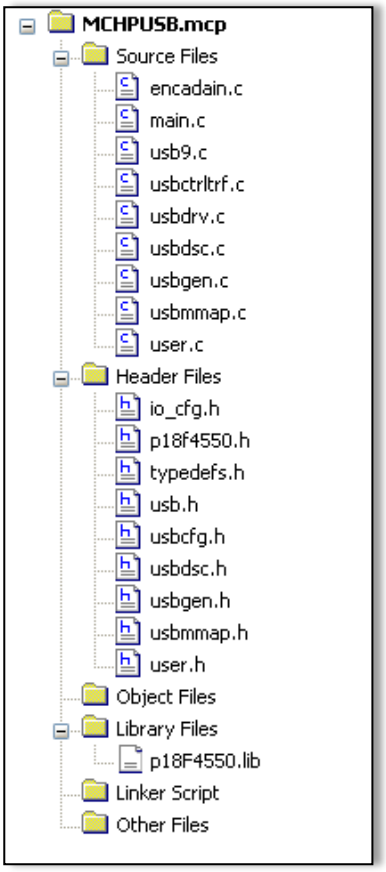


Figure 5.19 Archivos proyecto

La explicación se centrará en los archivos que han sido desarrollados íntegramente para este proyecto:

- Encadain.c
- Main.c
- User.c

Todos ellos pertenecientes a la carpeta “Source files”. Y por otro lado:

- User.h

Perteneciente a la carpeta “Header files.”

Es resto de archivos que se encuentran en el proyecto son librerías o archivos que no han sido desarrollados para este proyecto en particular y que cumplen funciones como la configuración del microprocesador para trabajar en la placa en la que viene montado, pero sobre todo para el funcionamiento de la comunicación USB.

5.6.1.- Main.c

```
/********************************************************************
 *
 *                Microchip USB C18 Firmware Version 1.0
 *
 ********************************************************************
 * FileName:        main.c
 * Dependencies:    See INCLUDES section below
 * Processor:       PIC18
 * Compiler:        C18 2.30.01+
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Rawin Rojvanit       11/19/04    Original.
 ********************************************************************/

/** I N C L U D E S **************************************************/
#include <p18f4550.h>
#include "system\typedefs.h"                        // Required
#include "system\usb\usb.h"                         // Required
#include "io_cfg.h"                                 // Required
#include "system\usb\usb_compile_time_validation.h" // Optional
#include "user\user.h"                              // Modifiable
#include <timers.h>
#include <math.h>
```

```
/** P R I V A T E  P R O T O T Y P E S ***************************************/
//extern unsigned char contadorAceleracion;
unsigned int contadorDeceleracion;
unsigned int contadorAceleracion;
unsigned char aceleracion;
unsigned short cont_pulsos;
static void InitializeSystem(void);
void USBTasks(void);
void contenido_interrupcion(void);
void pasoenterolfase(void);
void pasoentero2fase(void);
void mediopaso(void);
void OpenPWM2( char );
void OpenPWM1( char );
void SetDCPWM1( unsigned int );
void SetDCPWM2( unsigned int );

/** V E C T O R  R E M A P P I N G *******************************************/

extern void _startup (void);        // See c018i.c in your C18 compiler dir

#pragma idata


#pragma code _RESET_INTERRUPT_VECTOR = 0x000800

void _reset (void)
{
    _asm goto _startup _endasm
}
#pragma code

//Acciones cuando se produzcan interrupciones

//En el caso de interrupcion de alta prioridad
#pragma code _HIGH_INTERRUPT_VECTOR = 0x000008
void _high_ISR (void)
{
    _asm goto behitiko_etenaldi _endasm
}

//En el caso de interrupcion de baja prioridad
#pragma code _LOW_INTERRUPT_VECTOR = 0x000018
void _low_ISR (void)
{
   _asm goto behitiko_etenaldi _endasm
}

#pragma code
extern CONTROLMOVIMIENTO controlMovimiento;

//Se vendrá aquí cuando se produzca un interrupcion de baja o alta prioridad

#pragma interrupt behitiko_etenaldi
```

```
void behitiko_etenaldi(void)
{
//La "banderita" del contador del timer0 que nos avisa, por precaucion,
//en el caso de que hubiera mas interrupciones
if (INTCONbits.TMR0IF==1)
    {
    if (aceleracion ==1)
        {
        if (contadorAceleracion==0)
            {
                OpenPWM2( 0xff );
                OpenPWM1( 0xff );
                SetDCPWM1(controlMovimiento.corrientefaseA);
                SetDCPWM2( controlMovimiento.corrientefaseB );
                ContadorAjustableTemporizador0=controlMovimiento.contadorAcelTemp0;
                contadorAceleracion=contadorAceleracion+1;
            }
            else
            {
                if (contadorAceleracion<=controlMovimiento.pasosAceleracion)
                {
                    //Formula de modificacion de C en cada paso
                    ContadorAjustableTemporizador0=(ContadorAjustableTemporizador0-
                    (2*ContadorAjustableTemporizador0)/(4*contadorAceleracion+1));
                    contadorAceleracion=contadorAceleracion+1;
                }
            }
            contenido_interrupcion();
            //Ajuste del valor de C, para su introducción en el writetimer0
            //Se multiplica por 0.676 para compensar el error que se produce
            C_introducir0=(-(ContadorAjustableTemporizador0*3)+65536);
            WriteTimer0(C_introducir0);
            INTCONbits.TMR0IF = 0; //Bajamos la banderita

        }
    if (aceleracion==0)
        {

        if (contadorDeceleracion>0)
            {
                    //Formula de modificacion de C en cada paso
                    ContadorAjustableTemporizador0=(ContadorAjustableTemporizador0+
                    (2*ContadorAjustableTemporizador0)/(4*contadorDeceleracion+1));
                    contadorDeceleracion=contadorDeceleracion-1;
                    contenido_interrupcion();
                    //Ajuste del valor de C, para su introducción en el writetimer0
                    //Se multiplica por 0.676 para compensar el error que se produce
                    C_introducir0=(-(ContadorAjustableTemporizador0*3)+65536);
                    WriteTimer0(C_introducir0);
                    INTCONbits.TMR0IF = 0; //Bajamos la banderita
            }
        if (contadorDeceleracion==0)
            {
                mLED_3_Off();mLED_4_Off();
                PORTBbits.RB1=0;PORTCbits.RC6=0;PORTAbits.RA5=0;PORTBbits.RB4=0;
```

```
                ClosePWM1();ClosePWM2();
                CloseTimer0;
                INTCONbits.TMR0IF=0;
            }


        }

    }

//Interrupción por flanco de bajada en la entrada externa RB0

if (INTCONbits.INT0IF==1)
{

if(controlMovimiento.sentidoGiro==0) //Orden sentido horario por parte de C#
    {
    if(PORTAbits.RA3==1) //El encoder nos dice que es sentido horario
        {
        cont_pulsos=cont_pulsos+1;
        INTCONbits.INT0IF=0;
        }
    else
        {
        cont_pulsos=cont_pulsos-1;
        INTCONbits.INT0IF=0;
        }
    }
else
    {
    if (PORTAbits.RA3==0) //El encoder nos dice que el sentido es antihorario
        {
        cont_pulsos=cont_pulsos-1;
        INTCONbits.INT0IF=0;
        }
    else
        {
        cont_pulsos=cont_pulsos+1;
        INTCONbits.INT0IF=0;
        }

    }
}
}

/** D E C L A R A T I O N S **************************************************/
#pragma code
/******************************************************************************
 * Function:        void main(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
```

```
 * Side Effects:    None
 *
 * Overview:        Main program entry point.
 *
 * Note:            None
 *******************************************************************/
void main(void)
{
    InitializeSystem();
    while(1)
    {
        USBTasks();         // USB Tasks
        ProcessIO();        // See user\user.c & .h
    }//end while
}//end main

/********************************************************************
 * Function:        static void InitializeSystem(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        InitializeSystem is a centralize initialization routine.
 *                  All required USB initialization routines are called from
 *                  here.
 *
 *                  User application initialization routine should also be
 *                  called from here.
 *
 * Note:            None
 *******************************************************************/
static void InitializeSystem(void)
{
    ADCON1 |= 0x00;

    #if defined(USE_USB_BUS_SENSE_IO)
    tris_usb_bus_sense = INPUT_PIN; // See io_cfg.h
    #endif

    #if defined(USE_SELF_POWER_SENSE_IO)
    tris_self_power = INPUT_PIN;
    #endif

    mInitializeUSBDriver();         // See usbdrv.h

    UserInit();                     // See user.c & .h

}//end InitializeSystem

/********************************************************************
 * Function:        void USBTasks(void)
```

```
 * PreCondition:     InitializeSystem has been called.
 *
 * Input:            None
 *
 * Output:           None
 *
 * Side Effects:     None
 *
 * Overview:         Service loop for USB tasks.
 *
 * Note:             None
 *******************************************************************/
void USBTasks(void)
{
    /*
     * Servicing Hardware
     */
    USBCheckBusStatus();                    // Must use polling method
    if(UCFGbits.UTEYE!=1)
        USBDriverService();                 // Interrupt or polling method

}// end USBTasks

/*******************************************************************************
 * Function:         void contenido_interrupcion(void)
 *
 * PreCondition:
 *
 * Input:            None
 *
 * Output:           None
 *
 * Side Effects:     None
 *
 * Overview:         Da funcionalidad a las interrupciones del Timer0 que
 *                   controlan el tiempo de excitacion de los devanados
 *
 * Note:             None
 *******************************************************************************/


void contenido_interrupcion()
{
switch(controlMovimiento.tipoMovimiento)
{
case PASOENTERO1FASE:
pasoentero1fase();
break;
case PASOENTERO2FASE:
pasoentero2fase();
break;
case MEDIOPASO:
mediopaso();
break;
case MICROPASO:

break;
}
}

/** EOF main.c *************************************************************/
```

5.6.2.- user.c

```
/********************************************************************
*
*                Microchip USB C18 Firmware Version 1.0
*
********************************************************************
* FileName:         user.c
* Dependencies:     See INCLUDES section below
* Processor:        PIC18
* Compiler:         C18 2.30.01+
* Company:          Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology Incorporated
* (the "Company") for its PICmicro® Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* Author               Date        Comment
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Rawin Rojvanit       11/19/04    Original.
********************************************************************/

/** I N C L U D E S **************************************************/
#include <p18f4550.h>
#include <usart.h>
#include "system\typedefs.h"
#include <ADC.h>
#include "system\usb\usb.h"
#include "io_cfg.h"             // I/O pin mapping
#include "user\user.h"
#include <timers.h>
#include <portb.h>
#include <pwm.h>
/** V A R I A B L E S ************************************************/
#pragma udata
```

```
CONTROLMOVIMIENTO controlMovimiento;
byte counter;
byte trf_state;
byte velocL=0;
byte velocH=0;
DATA_PACKET dataPacket;
struct CONTROLMOVIMIENTO *pointerControlmovimiento;
//static unsigned int C_inicio;

/** P R I V A T E  P R O T O T Y P E S ********************************************/
extern unsigned int contadorDeceleracion;
extern unsigned int contadorAceleracion;
extern unsigned char aceleracion;
//extern unsigned char contadorAceleracion;
void ClosePWM1( void );
void ClosePWM2( void );
void OpenPWM1( char );
void SetDCPWM1( unsigned int);
void OpenPWM2( char  );
void SetOutputPWM1 (
        unsigned char ,
        unsigned char );
void SetOutputPWM2 (
        unsigned char ,
        unsigned char );
void SetDCPWM2( unsigned int dutycycle );
void BlinkUSBStatus(void);
void ServiceRequests(void);
void Leer_veloc(void);
void Denboragailu_0_on(struct CONTROLMOVIMIENTO *);
void Denboragailu_0_off(void);
/** D E C L A R A T I O N S **************************************************/
#pragma code
void UserInit(void)
{
 mInitAllLEDs();

//configuramos como entrada/salida los puertos utilizados
TRISB=0b00000001;
TRISCbits.TRISC6=0;
TRISCbits.TRISC7=0;
TRISAbits.TRISA5=0;
TRISAbits.TRISA3=1;
//Se programa Timer2 para con PWM obtener las referencias de corriente
OpenTimer2( TIMER_INT_OFF &T2_PS_1_16 &T2_POST_1_1 );
ADCON1 =0x0E;     //configures Vref to Vdd which is +5V in this case
INTCON = 0b11110000;
INTCON2= 0b00000000;
INTCON3= 0b00000000;
INTCON2bits.TMR0IP =1;          //Configuramos la interrupcio del timer0 con alta prioridad
RCONbits.IPEN = 1;
PORTBbits.RB2=1;
}//end UserInit
```

```
/*****************************************************************************
 * Function:        void Leer_veloc (void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Funcion para lectura del contador de impulsos
 *
 * Note:            None
 *****************************************************************************/

void Leer_veloc(void)
{
 velocL=cont_pulsos;
 velocH=cont_pulsos>>8;
 cont_pulsos=0;
}

/*****************************************************************************
 * Function:        void Denboragailu_0_on (void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Funcion para el comienzo de la aceleracion del motor paso a paso
 *
 * Note:            None
 *****************************************************************************/
void Denboragailu_0_on (struct CONTROLMOVIMIENTO *pointermovimiento)
{

mLED_4_On();

if (controlMovimiento.tipoMovimiento==MEDIOPASO)
{
OpenTimer0( TIMER_INT_ON &          //Interrupt enabled
                T0_16BIT &          //Set timer0 as 16 bit mode.
                T0_SOURCE_INT &     //choose Internal clock source (TOSC)
                T0_PS_1_64);          //Prescale Value: 1:64
}
else
{
OpenTimer0( TIMER_INT_ON &          //Interrupt enabled
                T0_16BIT &          //Set timer0 as 16 bit mode.
                T0_SOURCE_INT &     //choose Internal clock source (TOSC)
```

```
                T0_PS_1_128         //Prescale Value: 1:128
               );
}

contadorAceleracion=0; //Inicializamos el contador de pasos de aceleracion/deceleracion
aceleracion=1;      //Indicamos que el aceleracion(0 indica deceleracion)
WriteTimer0(60000); //Un simple retraso de 500ms hasta que entre en el bucle

}

//end Denboragailu_0_on



/*****************************************************************************
 * Function:        void Denboragailu_0_off (void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Funcion para el comienzo de la deceleracion del motor
 *
 * Note:            None
 *****************************************************************************/
void Denboragailu_0_off ()
{
contadorDeceleracion=controlMovimiento.pasosDeceleracion;
aceleracion=0; //Indicamos que ahora debe decelerar

}


/*****************************************************************************
 * Function:        void ProcessIO(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is a place holder for other user routines.
 *                  It is a mixture of both USB and non-USB tasks.
 *
 * Note:            None
 *****************************************************************************/
void ProcessIO(void)
{
BlinkUSBStatus();
```

```
    // User Application USB tasks
    if((usb_device_state < CONFIGURED_STATE)||(UCONbits.SUSPND==1)) return;

    ServiceRequests();

}//end ProcessIO


void ServiceRequests(void)
{

byte index;


    if(USBGenRead((byte*)&dataPacket,sizeof(dataPacket)))
    {
        counter = 0;

        switch(dataPacket.CMD)
        {
            case READ_VERSION:
                //dataPacket._byte[1] is len
                dataPacket._byte[2] = MINOR_VERSION;
                dataPacket._byte[3] = MAJOR_VERSION;
                counter=0x04;
                break;
             case ID_BOARD:
                counter = 0x01;
                if(dataPacket.ID == 0)
                {
                    mLED_3_Off();mLED_4_Off();
                }
                else if(dataPacket.ID == 1)
                {
                    mLED_3_Off();mLED_4_On();
                }
                else if(dataPacket.ID == 2)
                {
                    mLED_3_On();mLED_4_Off();
                }
                else if(dataPacket.ID == 3)
                {
                    mLED_3_On();mLED_4_On();
                }
                else
                    counter = 0x00;
                break;

            case DENBORAGAILUA_0_ON:

                ContadorInicialTemp0(dataPacket._byte+1);
                controlMovimiento.sentidoGiro=dataPacket._byte[5];
                controlMovimiento.pasosAceleracion=dataPacket._byte[6];//sólo 1 byte=256 pasos maximo
                controlMovimiento.pasosDeceleracion=dataPacket._byte[7];//sólo 1 byte=256 pasos maximo
                controlMovimiento.tipoMovimiento=dataPacket._byte[8];
                PWMdutyCorrienteFase(dataPacket._byte+9); //funcion para introducir los valores de las corrientes
```

```
                //Damos a pointerControlmovimiento la direccion de la variable controlMovimiento
                pointerControlmovimiento=&controlMovimiento;
                Denboragailu_0_on(pointerControlmovimiento);
                counter=0x01;

            break;
            case DENBORAGAILUA_0_OFF:
                Denboragailu_0_off();
                counter=0x01;
                break;

            case RESET:
                Reset();
                break;
            case VELOCIDAD:
                Leer_veloc();
                dataPacket._byte[1] = velocL;
                dataPacket._byte[2] = velocH;
                counter=0x03;
            default:
                break;

        }//end switch()
        if(counter != 0)
        {
            if(!mUSBGenTxIsBusy())
                USBGenWrite((byte*)&dataPacket,counter);
        }//end if
    }//end if

}//end ServiceRequests

/********************************************************************************
 * Function:        void BlinkUSBStatus(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        BlinkUSBStatus turns on and off LEDs corresponding to
 *                  the USB device state.
 *
 * Note:            mLED macros can be found in io_cfg.h
 *                  usb_device_state is declared in usbmmap.c and is modified
 *                  in usbdrv.c, usbctrltrf.c, and usb9.c
 ********************************************************************************/
void BlinkUSBStatus(void)
{
    static word led_count=0;

    if(led_count == 0)led_count = 10000U;
    led_count--;
```

```
    #define mLED_Both_Off()         {mLED_1_Off();mLED_2_Off();}
    #define mLED_Both_On()          {mLED_1_On();mLED_2_On();}
    #define mLED_Only_1_On()        {mLED_1_On();mLED_2_Off();}
    #define mLED_Only_2_On()        {mLED_1_Off();mLED_2_On();}

    if(UCONbits.SUSPND == 1)
    {
        if(led_count==0)
        {
            mLED_1_Toggle();
            mLED_2 = mLED_1;        // Both blink at the same time
        }//end if
    }
    else
    {
        if(usb_device_state == DETACHED_STATE)
        {
            mLED_Both_Off();

        }
        else if(usb_device_state == ATTACHED_STATE)
        {
            mLED_Both_On();
        }
        else if(usb_device_state == POWERED_STATE)
        {
            mLED_Only_1_On();
        }
        else if(usb_device_state == DEFAULT_STATE)
        {
            mLED_Only_2_On();
        }
        else if(usb_device_state == ADDRESS_STATE)
        {
            if(led_count == 0)
            {
                mLED_1_Toggle();
                mLED_2_Off();
            }//end if
        }
        else if(usb_device_state == CONFIGURED_STATE)
        {
            if(led_count==0)
            {
                mLED_1_Toggle();
                mLED_2 = !mLED_1;       // Alternate blink
            }//end if
        }//end if(...)
    }//end if(UCONbits.SUSPND...)

}//end BlinkUSBStatus

/** EOF user.c ***************************************************************/
```

5.6.3.- Encadain.c

```
/********************************************************************
 *
 *                Microchip USB C18 Firmware Version 1.0
 *
 ********************************************************************
 * FileName:         encadain.c
 * Dependencies:     See INCLUDES section below
 * Processor:        PIC18
 * Compiler:         C18 2.30.01+
 *
 *
 * Este programa contiene todas las diversas funciones que pueden
 *  ejecutarse cada vez que entra la interrupci¢n de la tarjeta o
 *  que impliquen el giro del motor paso a paso.     .
 ********************************************************************/

/** I N C L U D E S **************************************************/
#include <p18f4550.h>
#include <usart.h>
#include "system\typedefs.h"
#include <ADC.h>
#include "system\usb\usb.h"
#include "io_cfg.h"             // I/O pin mapping
#include "user\user.h"
#include <timers.h>
#include <portb.h>
#include <pwm.h>
/** V A R I A B L E S ************************************************/
#pragma udata

/** P R I V A T E  P R O T O T Y P E S *******************************/
void PWMdutyCorrientesFases(byte *);
void pasoenterolfase(void);
#pragma idata
static unsigned int contar=1;



/** D E C L A R A T I O N S ******************************************/
#pragma code
extern CONTROLMOVIMIENTO controlMovimiento;

//extern struct CONTROLMOVIMIENTO *_pointerCorrientefases;

/********************************************************************************
    Nombre:     pasoenterolfase
    Recibe:     nada
    Devuelve:   nada
    Esta función se encarga de excitar al motor paso a paso según
    una secuencia de pasos enteros (7.5 grados) en la cual esta solo
    1 fase excitada simultáneamente. Se puede girar
    en sentido horario y antihorario.
    Nota: Se utilizan los pins RB1 y RB3 del PIC para desinhibir
    los transistores de las Fase B y A respectivamente.
    Se inhiben con la señal a cero.
```

```
*****************************************************************************/

void pasoenterolfase()
{


if (controlMovimiento.sentidoGiro==0){ //Sentido horario

if (contar==1)  {
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}//PORTB =0b00011110;
if (contar==2) {
mLED_3_On();
mLED_4_Off();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}// PORTB =0b00011010;
if (contar==3){
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}//     PORTB =0b00001010;
if (contar==4)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}//PORTB =0b00001110;
}
else    //sentido antihorario
{
if (contar==1)  {
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1; //Inhibit_A
PORTAbits.RA5=0; //Inhibit_B
PORTCbits.RC6=0; //Phase_A
PORTBbits.RB4=1; //Phase_B
}//PORTB =0b00011110;
if (contar==2) {
mLED_3_On();
mLED_4_Off();
PORTBbits.RB1=0;
```

```
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}// PORTB =0b00011010;
if (contar==3){
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}//     PORTB =0b00001010;
if (contar==4)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}//PORTB =0b00001110;
}
contar++;
if (contar>4) contar=1;
}
//end pasoenterolfase()


/*********************************************************************
    Nombre:     pasoentero2fase
    Recibe:     nada
    Devuelve:   nada
    Esta función se encarga de excitar al motor paso a paso según
    una secuencia de pasos enteros (7.5 grados) en la cual siempre
    están las dos fases excitadas simultáneamente. Se puede girar
    en sentido horario y antihorario.
    Nota: Se utilizan los pins RB1 y RB3 del PIC para desinhibir
    los transistores de las Fase B y A respectivamente.
    Se inhiben con la señal a cero.
*********************************************************************/
void pasoentero2fase(){

//se mandan las corrientes de referencia para las dos fases del motor
// Timer2 ya esta inicializado, las corrientes enl a fases A y B puestas
//En user.c PORTB para controlar RB0=INT0=interrupciones encoder RB1=INHIBIT_A
//RC6=PWM_A RB3=INHIBIT_B RB4=PWM_B

/*Cuando haya que tomar datos del encoder para gráficos en pantalla PC
Enable interrupts for PORTB0 pin para la permitir las interrupciones del encoder.
*/
PORTBbits.RB1=1; //Habilitamos los mosfests de la rama superior de la fase A
PORTAbits.RA5=1; //Habilitamos los mosfests de la rama superior de la fase B

if (controlMovimiento.sentidoGiro==0){ //Sentido horario

if (contar==1) {
```

```
mLED_3_Off();
mLED_4_Off();
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}//PORTB =0b00011110;
if (contar==2) {
mLED_3_On();
mLED_4_Off()
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}   //PORTB =0b00011010;
if (contar==3){
mLED_3_On();
mLED_4_On();
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}   //PORTB =0b00001010;
if (contar==4){
mLED_3_Off();
mLED_4_On();
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}//      PORTB =0b00001110;
}
else    //sentido antihorario
{
if (contar==4)  {
mLED_3_Off();
mLED_4_Off();
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}//PORTB =0b00011110;
if (contar==3) {
mLED_3_On();
mLED_4_Off();
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}// PORTB =0b00011010;
if (contar==2){
mLED_3_On();
mLED_4_On();
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}//      PORTB =0b00001010;
if (contar==1)
{
mLED_3_On();
mLED_4_On();
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}//PORTB =0b00001110;
}
contar++;
if (contar>4) contar=1;
}
//end pasoentero2fase()
```

```
void mediopaso(){

/*************************************************************************
    Nombre:     mediopaso
    Recibe:     nada
    Devuelve:   nada
    Esta función se encarga de excitar al motor paso a paso según
    una secuencia de medios pasos (7.5/2 grados). Se puede girar
    en sentido horario y antihorario.
    Nota: Se utilizan los pins RB1 y RB3 del PIC para desinhibir
    los transistores de las Fase B y A respectivamente.
    Se inhiben con la señal a cero.
*************************************************************************/

if (controlMovimiento.sentidoGiro==0)    //Sentido horario
{

if (contar==1)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}

if (contar==2)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}
if (contar==3)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}
if (contar==4)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}
if (contar==5)
{
```

```
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}
if (contar==6)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}
if (contar==7)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}
if (contar==8)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}

}

else //Sentido antihorario
{

if (contar==1)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
}

if (contar==2)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
```

```
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}
if (contar==3)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=0;
}
if (contar==4)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}
if (contar==5)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=0;
PORTBbits.RB4=1;
}
if (contar==6)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=0;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}
if (contar==7)
{
mLED_3_On();
mLED_4_On();
PORTBbits.RB1=1;
PORTAbits.RA5=1;
PORTCbits.RC6=1;
PORTBbits.RB4=1;
}
if (contar==8)
{
mLED_3_Off();
mLED_4_Off();
PORTBbits.RB1=1;
PORTAbits.RA5=0;
PORTCbits.RC6=1;
PORTBbits.RB4=0;
```

```
}

}

contar++;
if (contar>8) contar=1;

}


void PWMdutyCorrienteFase(byte *fase){

                controlMovimiento.corrientefaseA= *(fase+1);
                controlMovimiento.corrientefaseA=
                controlMovimiento.corrientefaseA<<8;
                controlMovimiento.corrientefaseA=
                controlMovimiento.corrientefaseA| (unsigned int) *fase;
                controlMovimiento.corrientefaseB= *(fase+3);
                controlMovimiento.corrientefaseB=
                controlMovimiento.corrientefaseB<<8;
                controlMovimiento.corrientefaseB=
    controlMovimiento.corrientefaseB| (unsigned int) *(fase+2);
}
//end PWMdutyCorrienteFase
void ContadorInicialTemp0(byte *ptrcontadorTemp0){

        controlMovimiento.contadorAcelTemp0= *(ptrcontadorTemp0+1);
        controlMovimiento.contadorAcelTemp0=
        controlMovimiento.contadorAcelTemp0<<8;
        controlMovimiento.contadorAcelTemp0=
        controlMovimiento.contadorAcelTemp0| (unsigned int) *ptrcontadorTemp0;
        controlMovimiento.contadorDecelTemp0= *(ptrcontadorTemp0+3);
        controlMovimiento.contadorDecelTemp0=controlMovimiento.contadorDecelTemp0<<8;
        controlMovimiento.contadorDecelTemp0=
        controlMovimiento.contadorDecelTemp0| (unsigned int) *(ptrcontadorTemp0+2);
}
```

5.6.4.- User.h

```
/********************************************************************
 *
 *                Microchip USB C18 Firmware Version 1.0
 *
 ********************************************************************
 * FileName:         user.h
 * Dependencies:     See INCLUDES section below
 * Processor:        PIC18
 * Compiler:         C18 2.30.01+
 * Company:          Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Rawin Rojvanit       11/19/04    Original.
 ********************************************************************/

#ifndef PICDEM_FS_DEMO_H
#define PICDEM_FS_DEMO_H

/** I N C L U D E S **************************************************************/
#include "system\typedefs.h"

/** D E F I N I T I O N S ********************************************************/
/* PICDEM FS USB Demo Version */
#define MINOR_VERSION   0x00    //Demo Version 1.00
#define MAJOR_VERSION   0x01

/* Temperature Mode */
#define TEMP_REAL_TIME  0x00
#define TEMP_LOGGING    0x01

/** D E F I N I C I O N E S ******************************************************/
/* PRIVADAS*/

/*typedef enum _BOOL { FALSE = 0, TRUE } BOOL;
```

```
#define OK      TRUE
#define FAIL    FALSE*/
#define PASOSACELMAX 0x0F

/** S T R U C T U R E S ************************************************/
typedef union DATA_PACKET
{
    byte _byte[USBGEN_EP_SIZE];  //For byte access
    word _word[USBGEN_EP_SIZE/2];//For word access(USBGEN_EP_SIZE must be even)
    struct
    {
        enum
        {
            READ_VERSION    = 0x00,
            ID_BOARD        = 0x31,
            RESET           = 0xFF,
            DENBORAGAILUA_0_ON  = 0XA0,
            DENBORAGAILUA_0_OFF = 0XA1,
            VELOCIDAD       = 0XA3 //Captura de la velocidad
        }CMD;
        byte len;
    };
    struct
    {
        unsigned :8;
        byte ID;
    };
    struct
    {
        unsigned :8;
        byte led_num;
        byte led_status;
    };
    struct
    {
        unsigned :8;
        word word_data;
    };


} DATA_PACKET;
/****************************
*Look up MPLAB® C18 C COMPILER
*GETTING STARTED FAQ-3
****************************/

/* struct Corrientefases{
unsigned int
corrientefaseA,
corrientefaseB;
};
*/
typedef struct CONTROLMOVIMIENTO
{
word contadorAcelTemp0;
```

```
word contadorDecelTemp0;
byte sentidoGiro;
byte pasosAceleracion;
byte pasosDeceleracion;
//byte numeroVueltas;
word corrientefaseA;
word corrientefaseB;

enum {
            PASOENTERO1FASE     = 0x01,
            PASOENTERO2FASE     = 0x02,
            MEDIOPASO           = 0x32,
            MICROPASO           = 0xEE
      } tipoMovimiento;

}CONTROLMOVIMIENTO;



/** P U B L I C  P R O T O T Y P E S ***************************************/
void UserInit(void);
void ProcessIO(void);

void behitiko_etenaldi(void);
void PWM_on(struct CONTROLMOVIMIENTO *);
void PWMdutyCorrienteFase(byte *);

//

/* VARIABLES GLOBALES USADAS PARA EL CONTROL DE LA VELOCIDAD DEL MOTOR*/
extern unsigned char aceleracion;
extern unsigned int contadorAceleracion;
extern unsigned int contadorDeceleracion;
extern unsigned short cont_pulsos;

#endif //PICDEM_FS_DEMO_H
```

6.- Interfaz control motor paso a paso en C#

6.1.- Introducción al lenguaje C# y .NET Framework

C# es un lenguaje orientado a objetos que permite a los desarrolladores crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. Se puede utilizar este lenguaje para crear aplicaciones cliente para Windows tradicionales, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos, y muchas tareas más. Microsoft Visual C# proporciona un editor de código avanzado, diseñadores de interfaz de usuario prácticos, un depurador integrado y muchas otras herramientas para facilitar un rápido desarrollo de la aplicación basado en la versión 2.0 del lenguaje C# y en .NET Framework.

6.1.1.- Lenguaje C#

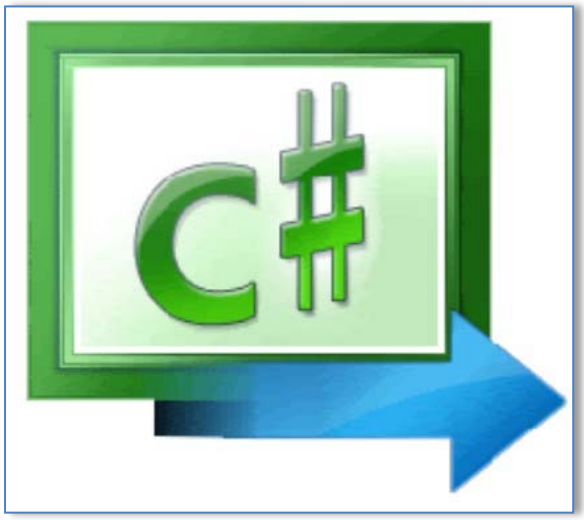


Figura 6.1

La sintaxis de C# cuenta con menos de 90 palabras clave; es sencilla y fácil de aprender. La sintaxis de C# basada en signos de llave podrá ser reconocida inmediatamente por cualquier persona familiarizada con C, C++ o Java. Los desarrolladores que conocen cualquiera de estos lenguajes pueden empezar a trabajar de forma productiva en C# en un plazo muy breve. La sintaxis de C# simplifica muchas de las complejidades de C++ y, a la vez, ofrece funciones eficaces tales como tipos de valores que aceptan valores NULL, enumeraciones, delegados, métodos anónimos y acceso directo a memoria, que no se encuentran en Java. C# también admite métodos y tipos genéricos, que proporcionan mayor rendimiento y seguridad de tipos, e iteradores, que permiten a los implementadores de clases de colección definir

comportamientos de iteración personalizados que el código de cliente puede utilizar fácilmente.

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método `Main` que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan a los métodos virtuales en una clase primaria requieren la palabra clave override como medio para evitar redefiniciones accidentales. En C#, una estructura es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia.

Figura 6.2

Además de estos principios básicos orientados a objetos, C# facilita el desarrollo de componentes de software a través de varias construcciones de lenguaje innovadoras, entre las que se incluyen:

- Firmas de métodos encapsulados denominadas delegados, que permiten notificaciones de eventos con seguridad de tipos.
- Propiedades, que actúan como descriptores de acceso para variables miembro privadas.
- Atributos, que proporcionan metadatos declarativos sobre tipos en tiempo de ejecución.
- Comentarios en línea de documentación XML.

Si necesita interactuar con otro software de Windows, como objetos COM o archivos DLL nativos de Win32, podrá hacerlo en C# mediante un proceso denominado "interoperabilidad". La interoperabilidad permite que los programas de C# realicen prácticamente lo mismo que una aplicación de C++ nativa. C# admite incluso el uso de punteros y el concepto de código "no seguro" en los casos en que el acceso directo a la memoria es absolutamente crítico.

El proceso de generación de C# es simple en comparación con el de C y C++, y es más flexible que en Java. No hay archivos de encabezado independientes, ni se requiere que los métodos y los tipos se declaren en un orden determinado. Un archivo de código fuente de C# puede definir cualquier número de clases, estructuras, interfaces y eventos.

6.1.2.- Arquitectura de la plataforma .NET Framework

Los programas de C# se ejecutan en .NET Framework, un componente que forma parte de Windows y que incluye un sistema de ejecución virtual denominado Common Language Runtime (CLR) y un conjunto unificado de bibliotecas de clases. CLR es la implementación comercial de Microsoft de Common Language Infrastructure (CLI), norma internacional que constituye la base para crear entornos de ejecución y desarrollo en los que los lenguajes y las bibliotecas trabajan juntos sin problemas.

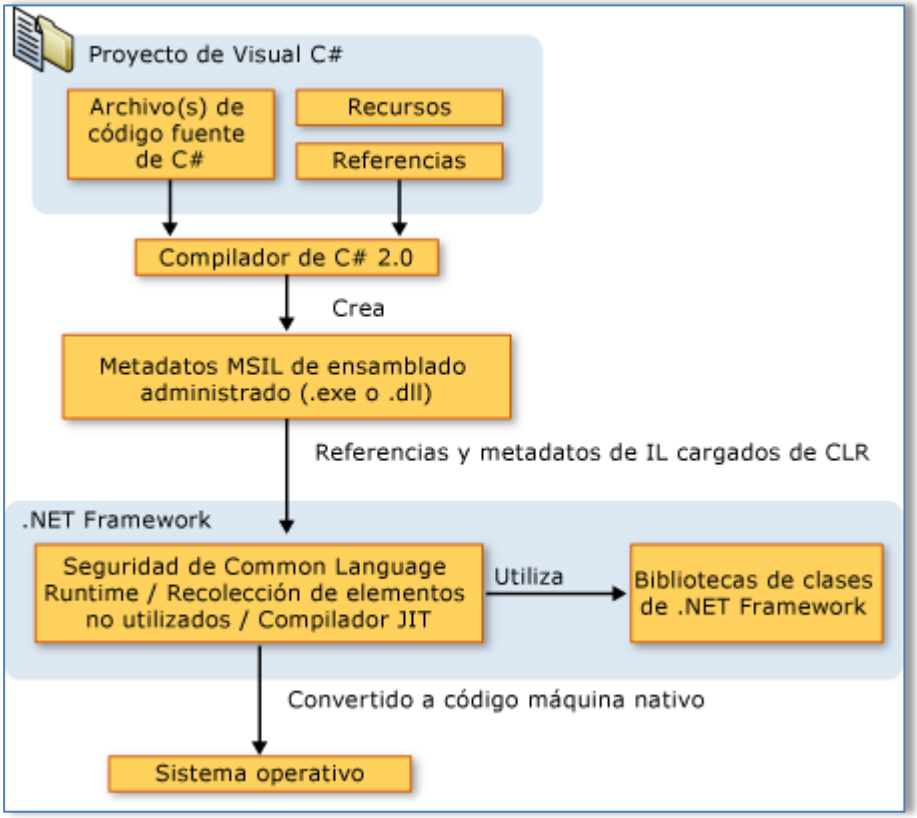


Figura 6.3

El código fuente escrito en C# se compila en un lenguaje intermedio (IL) conforme con la especificación CLI. El código de lenguaje intermedio, junto con

recursos tales como mapas de bits y cadenas, se almacena en disco en un archivo ejecutable denominado ensamblado, cuya extensión es .exe o .dll

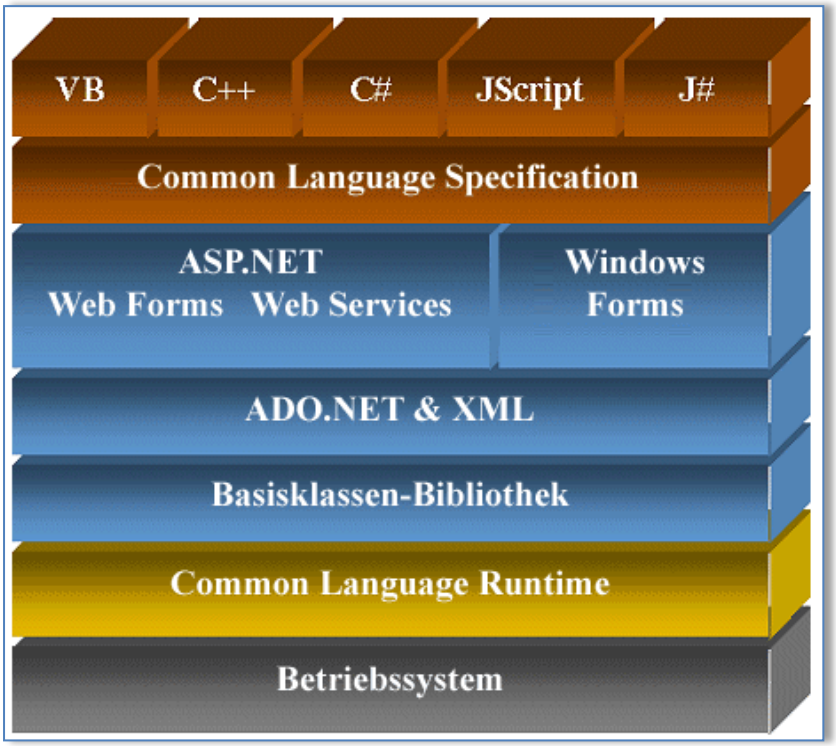


Figura 6.4

generalmente. Un ensamblado contiene un manifiesto que ofrece información sobre los tipos, la versión, la referencia cultural y los requisitos de seguridad del ensamblado.

Cuando se ejecuta un programa de C#, el ensamblado se carga en CLR, con lo que se pueden realizar diversas acciones en función de la información del manifiesto. A continuación, si se cumplen los requisitos de seguridad, CLR realiza una compilación Just In Time (JIT) para convertir el código de lenguaje intermedio en instrucciones máquina nativas. CLR también proporciona otros servicios relacionados con la recolección automática de elementos no utilizados, el control de excepciones y la administración de recursos. El código ejecutado por CLR se denomina algunas veces "código administrado", en contraposición al "código no administrado" que se compila en lenguaje máquina nativo destinado a un sistema específico. En el diagrama siguiente se muestran las relaciones en tiempo de compilación y tiempo de ejecución de los archivos de código fuente de C#, las bibliotecas de clases base, los ensamblados y CLR.

La interoperabilidad del lenguaje es una función clave de .NET Framework. Como el código de lenguaje intermedio generado por el compilador de C# cumple la especificación de tipos común (CTS), este código generado en C#

puede interactuar con el código generado en las versiones .NET de Visual Basic, Visual C++, Visual J# o cualquiera de los más de 20 lenguajes conformes con CTS. Un único ensamblado puede contener varios módulos escritos en diferentes lenguajes .NET, y los tipos admiten referencias entre sí como si estuvieran escritos en el mismo lenguaje.

Figura 6.5

Además de los servicios en tiempo de ejecución, .NET Framework también incluye una amplia biblioteca de más de 4.000 clases organizada en espacios de nombres que ofrecen una diversidad de funciones útiles para la entrada y salida de archivos, la manipulación de cadenas, el análisis de archivos XML y los controles de formularios Windows Forms. La aplicación de C# típica utiliza continuamente la biblioteca de clases de .NET Framework para el tratamiento de las tareas comunes de "infraestructura".

6.1.3.- Formulario Windows Forms

Los formularios Windows Forms son la tecnología que se utiliza en Visual C# para crear aplicaciones para clientes inteligentes basadas en Windows que se ejecutan en .NET Framework. Cuando crea un proyecto de aplicación para Windows, está creando una aplicación basada en formularios Windows Forms. Utilizará el Diseñador de Windows Forms para crear la interfaz de usuario y tendrá acceso a otras funciones de diseño y tiempo de ejecución:

- Implementación ClickOnce
- Compatibilidad enriquecida de bases de datos con el control DataGrisView
- Barras de herramientas y otros elementos de interfaz de usuario que pueden tener el aspecto y comportamiento de Microsoft® Windows® XP, Microsoft Office o Microsoft Internet Explorer.

En Visual C#, la forma más rápida y cómoda de crear la interfaz del usuario (UI) es hacerlo visualmente, con el Diseñador de Windows Forms y el Cuadro de herramientas. Hay tres pasos básicos para crear todas las interfaces de usuario:

- Agregar los controles a la superficie de diseño.
- Establecer las propiedades iniciales de los controles.
- Escribir los controladores para los eventos especificados.

Aunque también puede escribir su propio código para crear la UI, los diseñadores permiten hacer este trabajo mucho más rápidamente de lo que es posible mediante codificación manual.

6.2.- Funcionamiento del interfaz Control motor paso a paso

En primer lugar, para facilitar la comprensión del funcionamiento se explicará la interfaz de usuario, mostrada en la figura 6.6.:

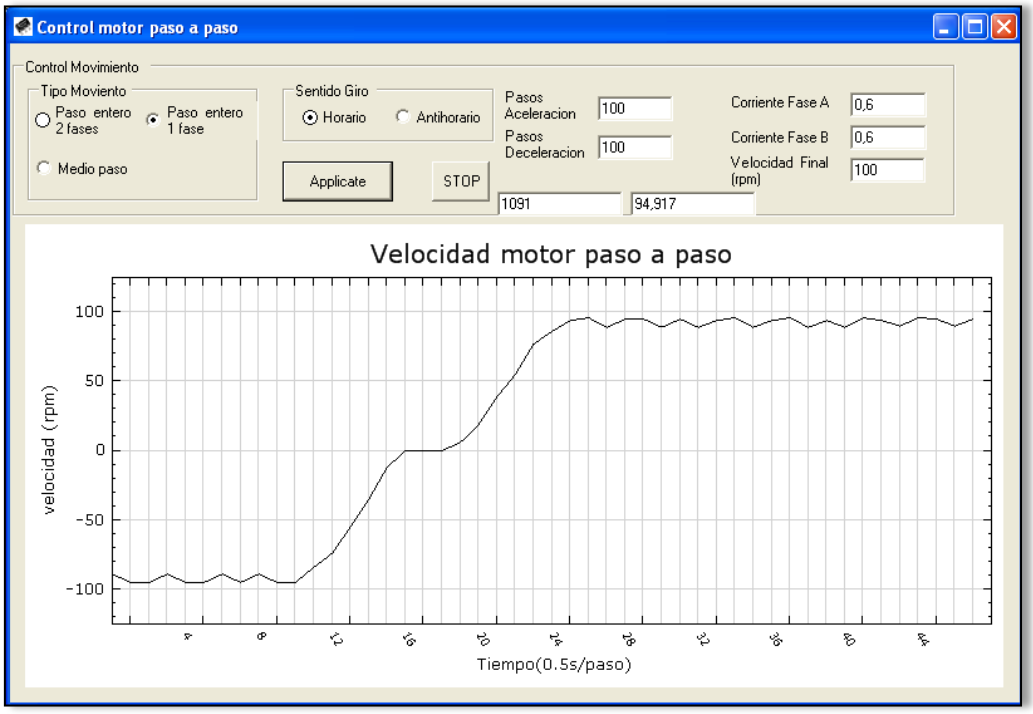


Figura 6.6

Desde esta ventana se tiene acceso a las diferentes configuraciones, y al control del motor paso a paso. Se divide en dos grandes partes, la parte superior desde donde se configura y controla el motor y la parte inferior donde se muestra la información recibida desde el encoder del motor

6.2.1.- Control del motor

En la parte superior de la interfaz, tal y como se puede observar figura 6.6, se en la parte de control, la cual se divide a su vez en secciones más reducidas. A continuación se explican las funciones de cada una de ellas.

6.2.1.1.- Tipo de movimiento

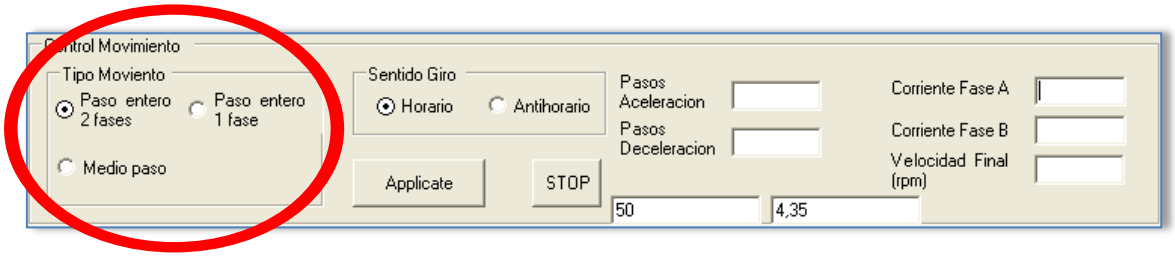


Figura 6.7 Sección de configuración control de movimiento

En esta sección se puede seleccionar uno de los tres tipos de movimiento (modo de excitación) para los que está preparado el sistema.

- *Paso entero (dos fases):* En este modo de excitación después de cada conmutación siempre resultan estar excitadas las dos fases
- *Paso entero (una fase):* En este modo de excitación, en cada secuencia de comunicación solamente una fase está excitada
- *Medio paso*: En este modo de excitación el motor se desplaza en cada conmutación la mitad del ángulo de paso

6.2.1.2.- Sentido de giro

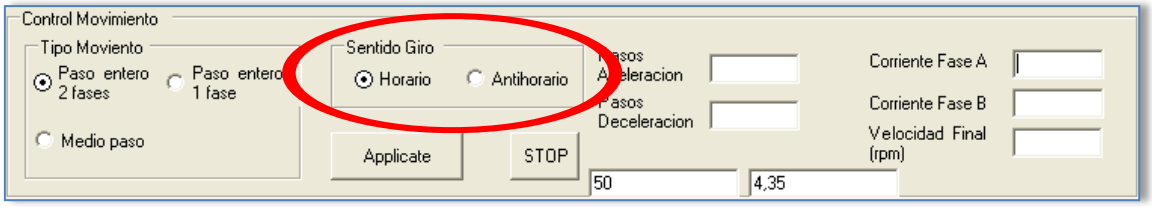


Figura 6.8 Sección de configuración sentido de giro

Aquí sencillamente se selecciona el sentido de giro del motor, horario o anti horario.

6.2.1.3.- Variables control motor paso a paso

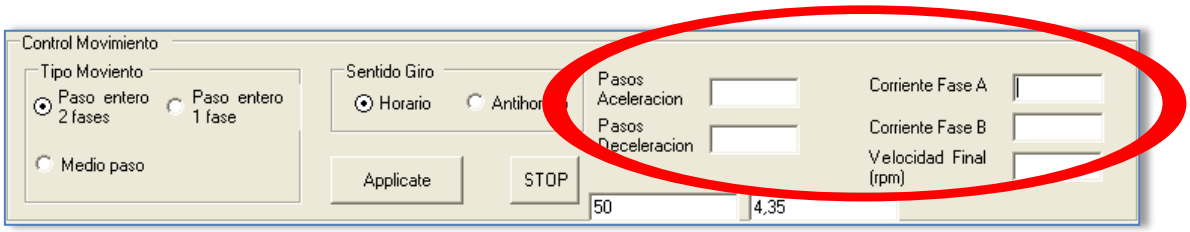


Figura 6.9 Sección de configuración

Este es lugar donde se deben introducir el resto de la variables que determinarán el funcionamiento del motor paso a paso.

- *Pasos de aceleración*: El número de pasos que dará el motor paso a paso durante la aceleración, desde el punto de parada hasta alcanzar la velocidad final.
- *Pasos de deceleración*: El número de pasos que dará el motor paso paso durante la deceleración, desde la velocidad de giro estacionaria hasta la parada.
- *Corriente Fase A*: Es la corriente en amperios que se desea que circule por la fase A del motor paso a paso.
- *Corriente Fase B*: Es la corriente en amperios que se desea que circule por la fase B del motor paso a paso.
- *Velocidad Final:* Es la velocidad estacionaria que debe alcanzar el motor, en revoluciones por minuto.

Se debe mencionar que los pasos de aceleración y deceleración no tienen por qué ser los mismos, el motor paso a paso puede configurarse para tener elevada aceleración y lenta deceleración o viceversa.

Algo parecido ocurre con las corrientes por la fase A o B, se pueden introducir diferentes corrientes en cada fase, pero se debe tener en cuenta que funcionando en modo paso entero con dos fases, si las corrientes son diferentes, el funcionamiento no será correcto. Hay que asegurarse que en cualquier caso la corriente que circula por cada fase sea, suficiente para superar el par que el motor deba generar para mover la carga.

Estas variables también tienen la peculiaridad de que los valores que se pueden introducir están restringidas, con lo cual no es posible introducir valores incoherentes o fuera de los márgenes que se han determinado:

	Valores límite
Pasos de aceleración	0-255
Pasos de deceleración	0-255
Corriente Fase A	0-0,6
Corriente Fase B	0-0,6
Velocidad final	0-100

6.2.1.4.- Impulsos del encoder

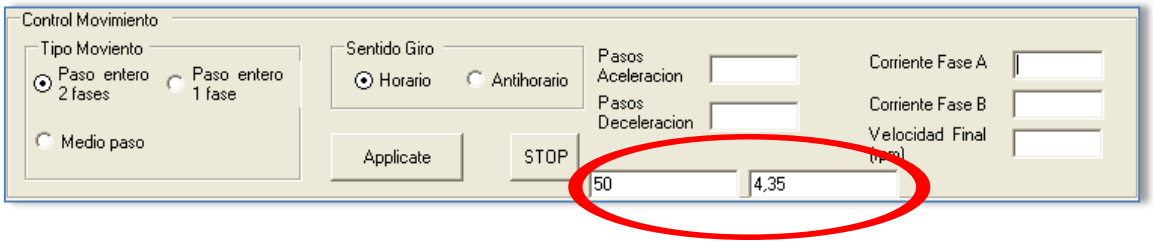


Figura 6.10

Estos dos valores, son valores de salida. El valor de la izquierda indica el número de impulsos que ha mandado el encoder del motor paso a paso cada un determinado tiempo (fijado dentro del programa de C#), mientras que el valor que se encuentra a la derecha es la transformación del valor anterior en

revoluciones por segundo, valor que posteriormente se mostrará en la gráfica de salida.

6.2.1.5.- Applicate & Stop

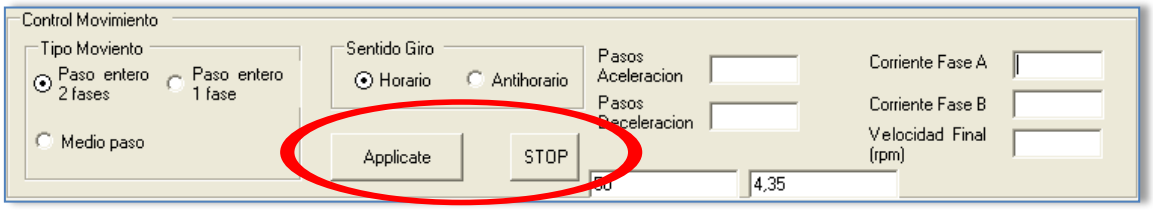


Figura 6.11

Applicate y Stop son los botones que controlan, dentro de la interfaz de control del motor paso a paso, la puesta en marcha y apagado del motor. La puesta en marcha sólo se efectuará si los valores introducidos en las variables son correctos y supondrá una aceleración desde una velocidad nula hasta la velocidad final, en todos los casos. Por otra parte el Stop es el accionamiento que detiene el giro del motor con la correspondiente deceleración determinada por las variables previamente introducidas.

6.2.2.-Gráfica velocidad (rpm)-Tiempo (s)

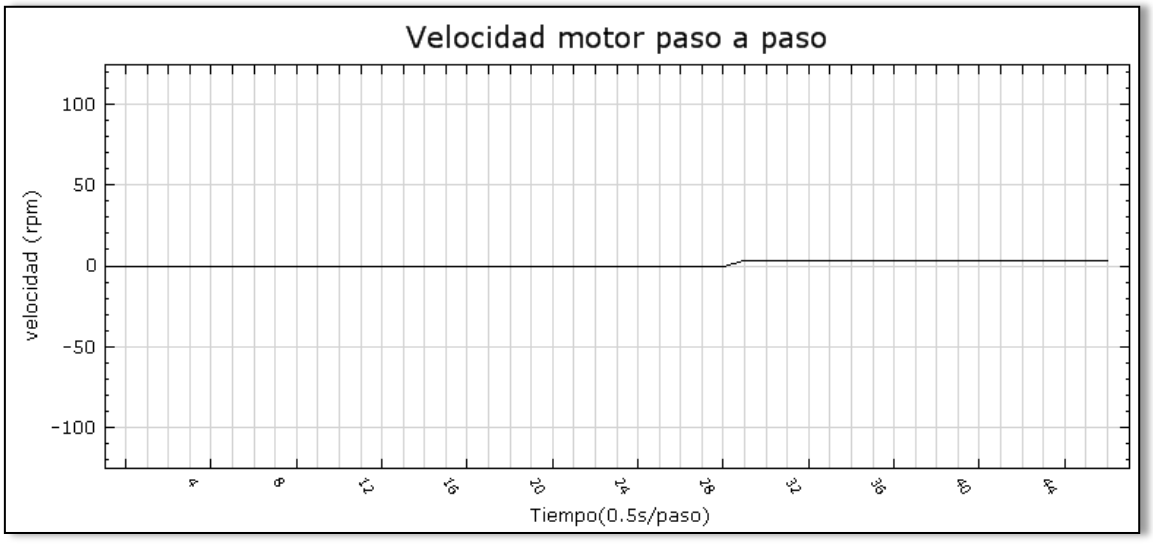


Figura 6.10

Esta gráfica muestra las revoluciones por minuto a las que gira el motor, en caso de sentido horario la gráfica será positiva y en caso de girar en sentido anti horario lo valores que mostrará la gráfica serán negativos.

Los valores de la gráfica son obtenidos a partir de los impulsos que envía el encoder, por lo que se puede afirmar que es la velocidad real de giro del motor en tiempo real (Hay un pequeño retraso entre el giro del motor y la información mostrada en la gráfica).

6.3.- Estructura interna del interfaz control motor paso a paso

Hasta ahora se ha visto una pequeña introducción al lenguaje de programación C# y el funcionamiento del interfaz de control del motor paso a paso, en este apartado se mostrará el código de ese interfaz, como funciona el programa. En

```
//Añadimos todos los namespaces necesarios
using System;
using System.Drawing;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using usb_api;
using NPlot;
using WindowsApplication3;

//Creamos nuestro namespace
namespace WindowsApplication3
{

    /// <summary>
    /// Summary description for Form1.
    /// </summary>

    //Creamos las clase USBdema herencia de la clase System.Windows.Forms.Form
   public class USBdemo : System.Windows.Forms.Form
   {
       usb_interface usb_int = new usb_interface();
        private Timer timer2;
        private IContainer components;
        private double[] PlotQEExampleValues2;
        private string[] PlotQEExampleTextValues2;
        private GroupBox groupBox2;
        private TextBox textBoxControlMovCorrienteA;
```

Figura 6.11

este apartado no pretendemos mostrar cada línea de código del interfaz, para ello se dispone del correspondiente apartado, sino explicar el flujo de información y la estructura que hay detrás.

Cuando se ejecuta el programa, lo primero que hace es realizar una instancia a una clase (USBDemo.cs), a su vez herencia de System.Windows.Forms.Form, creando un objeto, que será nuestra interfaz de control.

Para la creación de este objeto se ha inicializado y configurado todos sus elementos, como son los cuadros de texto, los botones, las etiquetas, incluso un temporizador, pero también a su vez se ha realizado una instancia a otra clase (usb_interface_inigo.cs) que prepara y activa la comunicación por USB entre el ordenador y el microprocesador.

6.3.1.- Temporizador interfaz control motor paso a paso

Uno de los elementos inicializados, al correr el programa, es el temporizador, el cual produce una interrupción que produce un evento cada 0.5 segundos.

```
// timer2
//
this.timer2.Enabled = true;
this.timer2.Interval = 500;
this.timer2.Tick += new System.EventHandler(this.timer2_Tick);
```

Figura 6.12

Por lo tanto una vez inicializado el programa, cada 0.5s se ejecutará un evento el cual consiste en mandar al microprocesador la orden de que nos mande el número de interrupciones que ha contado hasta ese momento, ese valor lo

```
private void timer2_Tick(object sender, EventArgs e) //Evento del timer2
{

        textBox1.Text = usb_int.datos_velocidad().ToString();
        double vel1 = double.Parse(textBox1.Text);
        vel1 = 0.087 * vel1;
        short vel_short = Convert.ToInt16(vel1);
        string vel_salida = Convert.ToString(vel1);
        textBox2.Text = vel_salida;



    plotSurface2.Clear(); //Limpiamos la gráfica 2
    Grid grid2 = new Grid();//Cuadricula para la gráfica 2
    grid2.VerticalGridType = Grid.GridType.Coarse;
    grid2.HorizontalGridType = Grid.GridType.Coarse;
    grid2.MajorGridPen = new Pen(Color.LightGray, 1.0f);
    plotSurface2.Add(grid2); //Introduciomos la cuadricula en la gráfica
```

Figura 6.13

adaptamos a revoluciones por minuto para después actualizar la gráfica, consiguiendo un flujo de valores (velocidad) en la gráfica, que se actualiza cada 0.5 segundos.

Es importante mencionar que en caso de que el ordenador no reciba respuesta por parte del microprocesador, por la razón que sea, el programa del interfaz de control está programado para devolver el valor 50, que es el que sacará por pantalla en ese caso.

6.3.2.- Applicate. Puesta en marcha del motor paso a paso

Una vez que todos los parámetros y modos de funcionamiento del motor están correctamente introducidos, se está en disposición de poner en marcha el motor mediante la pulsación del botón “Applicate”.

Este botón tiene asociado un evento que se ejecutará al clickear sobre él. En el evento se comprueba en primer lugar que todos los valores introducidos sean correctos, en caso contrario detenemos el evento y pedimos al usuario que repase la configuración.

```
        private void button1_Click(object sender, EventArgs e)
        {

            //Creamos el objeto controlmoviento de la clase ControlMovimiento
ControlMovimiento controlmovimiento = new ControlMovimiento(true);
                if (controlmovimiento.comprobarConvertirGrupoControMov(textBoxControlMovPasosAcel.Text,
                        textBoxControlMovPasosDecel.Text,
                        textBoxControlMovVelocidad.Text,
                        textBoxControlMovCorrienteA.Text,
                        textBoxControlMovCorrienteB.Text,
                        radioButtonDerecha.Checked,
                        radioButtonTipoMovPasoEnt2Fase.Checked,
                        radioButtonPasosEnteroUnafase.Checked,
                        radioButtonMedioPaso.Checked))
                {
                    usb_int.controlMovUsb(controlmovimiento);
                }
```

Figura 6.14

En caso de que todos los valores sean correctos, mandaremos a través del USB la orden de puesta en marcha del motor junto con todos los valores necesarios.

6.3.3.- Stop. Parada del motor paso a paso

Para detener el motor, se debe pulsar el botón de “Stop”, para que mediante la deceleración se efectúe esta acción.

```
private void button2_Click(object sender, EventArgs e)
{
    usb_int.Stop();
}
```

Figura 6.15

Esta acción es más sencilla que la anterior, debido a que en este caso no es necesario realizar ninguna comprobación, simplemente manda por USB la orden de detención del motor, los valores ya habían sido previamente cargados en la puesta en marcha

```
public uint Stop() {

    byte* send_buf = stackalloc byte[64];
    byte* receive_buf = stackalloc byte[64];

    DWORD RecvLength = 3;

    send_buf[0] = 0xA1;          //Command for TIMER0_OFF


    if (SendReceivePacket(send_buf, 1, receive_buf, &RecvLength) == 1)
    {
        if (RecvLength == 1 && receive_buf[0] == 0xA1)
        {
            return 0;
        }
        else
        {
            return 2;
        }
    }
    else
    {
            return 1;
    }
}
```

Figura 6.16-Envío por USB de la orden de detención del motor

6.3.4.- Estructura control de movimiento

Merece la pena hacer una mención especial a esta estructura, se ha hecho un breve comentario sobre ella en el apartado del evento ligado al botón "Applicate". Se trata de la estructura de datos que se envía al microprocesador a través del USB, junto con la orden de puesta en marcha del motor, estos datos componen toda la información que necesita el microprocesador para controlar el motor paso a paso.

```
public uint controlMovUsb(ControlMovimiento e)
{
    uint Temp=0;

    byte* send_buf = stackalloc byte[64];
    byte* receive_buf = stackalloc byte[64];

    byte[] contadorAcelTemp0 = new byte[2];
    byte[] contadorDecelTemp0 = new byte[2];
    byte[] sendcurrentA = new byte[2];
    byte[] sendcurrentB = new byte[2];
    contadorAcelTemp0 = BitConverter.GetBytes(e.contadorInicialAcelTemp0);
    contadorDecelTemp0 = BitConverter.GetBytes(e.contadorInicialDecelTemp0);
    sendcurrentA = BitConverter.GetBytes(e.valueCorrienteFaseA);
    sendcurrentB = BitConverter.GetBytes(e.valueCorrienteFaseB);

    DWORD RecvLength = (byte)3;
    send_buf[0] =0xA0;
    send_buf[1] = contadorAcelTemp0[0];
    send_buf[2] = contadorAcelTemp0[1];
    send_buf[3] = contadorDecelTemp0[0];
    send_buf[4] = contadorDecelTemp0[1];
    send_buf[5] = e.byteSentidoGiro;
    send_buf[6] = e.valuePasosAceleracion;
    send_buf[7] = e.valuePasosDeceleracion;
    send_buf[8] = e.byteTipoMovimiento;
    send_buf[9]  = sendcurrentA[0];
    send_buf[10] = sendcurrentA[1];
    send_buf[11] = sendcurrentB[0];
    send_buf[12] = sendcurrentB[1];
```

Figura 6.17

6.4.- Entorno de trabajo en C#

Al igual que en el apartado del microprocesador previamente a mostrar las líneas de código del programa, realizaremos un pequeña introducción al entorno de trabajo en el que se ha desarrollado el proyecto.

6.4.1.- Introducción al entorno IDE (Visual C#)

El entorno de desarrollo integrado (IDE) de Visual C# es un conjunto de herramientas de desarrollo expuestas a través de una interfaz de usuario común. Algunas de las herramientas se comparten con otros lenguajes de Visual Studio, y otras, como el compilador de C#, son únicas de Visual C#. La documentación de esta sección describe de forma general cómo utilizar las herramientas más importantes de Visual C# mientras se trabaja en el IDE en distintas fases del proceso de desarrollo.

6.4.2.- Herramientas de Visual C#

A continuación se detallan las herramientas y ventanas más importantes de Visual C#. Las ventanas de la mayoría de estas herramientas se pueden abrir desde el menú Ver.

- El Editor de código, para escribir código fuente.
- El compilador de C#, para convertir el código fuente de C# en un programa ejecutable.
- El depurador de Visual Studio, para probar el programa.
- El Cuadro de herramientas y el Diseñador, para desarrollar rápidamente interfaces de usuario con el mouse.
- El Explorador de soluciones, para ver y administrar archivos de proyecto y configuraciones.
- El Diseñador de proyectos, para configurar opciones del compilador, rutas de implementación, recursos, etc.
- La Vista de clases, para desplazarse por el código fuente según los tipos, no los archivos.

- La Ventana Propiedades, para configurar propiedades y eventos en los controles de la interfaz de usuario.
- El Examinador de objetos, para ver los métodos y clases disponibles en las bibliotecas de vínculos dinámicos, incluidos los ensamblados de .NET Framework y los objetos COM.
- Document Explorer, para explorar y buscar la documentación del producto en su equipo local y en Internet.

6.4.3.- Cómo expone las herramientas el IDE

Se puede interactuar con las herramientas a través de ventanas, menús, páginas de propiedades y asistentes en el IDE. El IDE básico tiene un aspecto similar al siguiente:

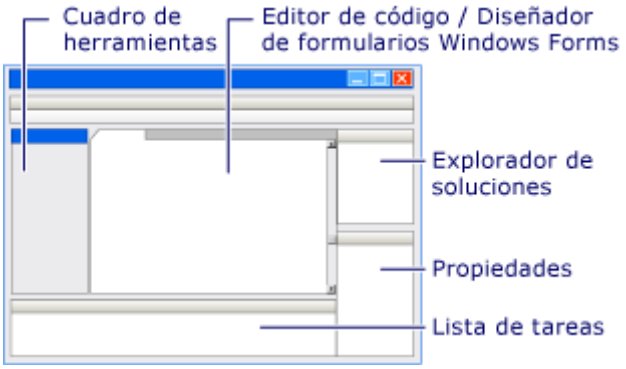


Figure 6.18

Puede tener acceso rápidamente a las ventanas de herramientas o archivos abiertos presionando CTRL + TAB.

6.4.3.1.- Ventanas del Editor y del Diseñador de Windows Forms

El Editor de código y el Diseñador de Windows Forms utilizan la ventana principal grande. Para alternar entre la vista de código y la vista Diseño, puede presionar F7 o hacer clic en Código o Diseñador en el menú Ver. En la vista Diseño, puede arrastrar controles a la ventana desde el Cuadro de

herramientas, que se puede hacer visible haciendo clic en la ficha Cuadro de herramientas del margen izquierdo.

La ventana Propiedades de la parte inferior derecha sólo contiene información en vista Diseño. Permite establecer las propiedades y enlazar los eventos para los controles de la interfaz de usuario, como botones, cuadros de texto, etc. Cuando esta ventana se establece como Ocultar automáticamente, se contrae en el margen derecho cada vez que se cambia a Vista Código.

6.4.3.2.- Explorador de soluciones y Diseñador de proyectos

La ventana de la parte superior derecha es el Explorador de soluciones, que muestra todos los archivos del proyecto en una vista de árbol jerárquica. Cuando se utiliza el menú Proyecto para agregar nuevos archivos al proyecto, se verán reflejados en el Explorador de soluciones. Además de los archivos, el Explorador de soluciones también muestra la configuración del proyecto y las referencias a las bibliotecas externas que necesita la aplicación.

Para obtener acceso a las páginas de propiedades del Diseñador de proyectos, haga clic con el botón secundario del mouse en el nodo Propiedades del Explorador de soluciones y, a continuación, haga clic en Abrir. Utilice estas páginas para modificar opciones de generación, requisitos de seguridad, detalles de implementación y muchas otras propiedades del proyecto.

6.4.3.3.- Ventanas Compilador, Depurador y Lista de errores

El compilador de C# no tiene ninguna ventana porque no es una herramienta interactiva, pero puede establecer sus opciones en el Diseñador de proyectos. Cuando se hace clic en Generar en el menú Generar, el IDE invoca el compilador de C#. Si la generación se realiza correctamente, el panel de estado muestra un mensaje Generación satisfactoria. Si se producen errores de generación, aparece la ventana Lista de errores aparece debajo de la ventana del editor/diseñador con una lista de errores. Haga doble clic en un error para ir a la línea de código fuente que presenta el problema. Presione F1 para consultar la documentación de Ayuda correspondiente al error resaltado.

El depurador tiene distintas ventanas que muestran valores de variables e información de tipos a medida que se ejecuta la aplicación. Puede utilizar la ventana Editor de código mientras depura el programa para especificar una línea en la que desee hacer una pausa durante la ejecución en el depurador y para recorrer el código línea a línea.

6.4.4.- Personalizar el IDE

En Visual C#, todas las ventanas pueden ser acoplables o flotantes, estar ocultas o visibles o cambiarse de ubicación. Para cambiar el comportamiento de una ventana, haga clic en el icono de flecha abajo o de pin de la barra de título y seleccione entre las opciones disponibles. Para mover una ventana acoplada a una nueva ubicación, arrastre la barra de título hasta que aparezcan los iconos de colocación de la ventana. Mantenga presionado el botón primario del mouse y mueva el puntero del mouse sobre el icono en la nueva ubicación. Coloque el puntero sobre los iconos izquierdo, derecho, superior o inferior para acoplar la ventana en el lado especificado. Coloque el puntero sobre el icono del medio para transformar la ventana en una ventana con fichas. Mientras coloca el puntero, aparece un rectángulo azul semitransparente que indica dónde se acoplará la ventana en la nueva ubicación.

Puede personalizar muchos otros aspectos del IDE haciendo clic en Opciones en el menú Herramientas

6.5.- Código C# en el ordenador

El PICusbdemo, este es el nombre del proyecto, está desarrollado a partir de una aplicación de Windows form, en este caso llamada USBdemo, es un entorno de ventanas tras la que corre el programa.

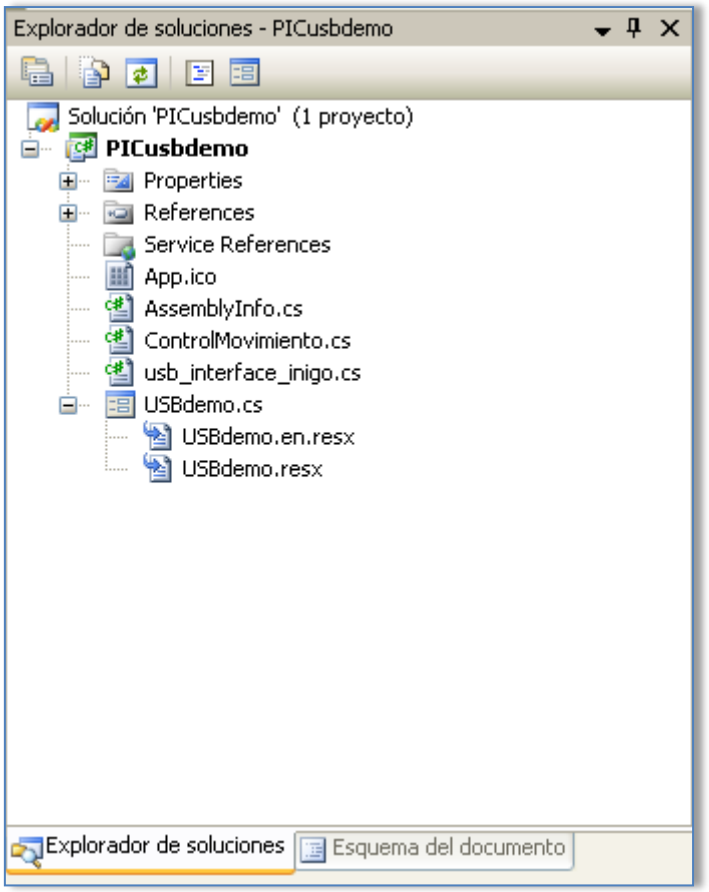


Figure 6.19

En el explorador de soluciones podemos ver claramente por qué clases está formado el programa.

- USBdemo.cs: Es una clase Windows form, sería el Main del programa, cuando lo corremos aparece la ventana aquí creada.
- ControlMovimiento.cs: En esta clase es la que da forma a las estructuras de información que se manda al microprocesador, el chequeo de toda la información que se ha introducido, la conversión de la información para poder ser enviada correctamente.
- Usb_interface_inigo: Esta es la clase encargada de enviar correctamente la información mediante el protocolo USB, aquí se encuentra la información específica que vamos a mandar por el USB

6.5.1.- USBdemo.cs

```
//Añadimos todos los namespaces necesarios
using System;
using System.Drawing;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using usb_api;
using NPlot;
using WindowsApplication3;

//Creamos nuestro namespace
namespace WindowsApplication3
{

    /// <summary>
    /// Summary description for Form1.
    /// </summary>

    //Creamos las clase USBdema herencia de la clase
System.Windows.Forms.Form
   public class USBdemo : System.Windows.Forms.Form
   {
       usb_interface usb_int = new usb_interface();
        private Timer timer2;
        private IContainer components;
        private double[] PlotQEExampleValues2;
        private string[] PlotQEExampleTextValues2;
        private GroupBox groupBox2;
        private TextBox textBoxControlMovCorrienteA;
        private RadioButton radioButtonIzquierda;
        private RadioButton radioButtonDerecha;
        private Label label3;
        private Label label6;
        private TextBox textBoxControlMovVelocidad;
        private TextBox textBoxControlMovPasosAcel;
        private Label label8;
        private GroupBox groupBox3;
        private RadioButton radioButtonTipoMovPasoEnt2Fase;
        private GroupBox groupBox4;
        private RadioButton radioButtonMedioPaso;
        private RadioButton radioButtonPasosEnteroUnafase;
        private Button buttonControlMovApplicate;
        private TextBox textBoxControlMovCorrienteB;
        private Label label5;
        private TextBox textBoxControlMovPasosDecel;
        private Label label7;
        private Button button2;
        static public short[] DatuBalioak = new short[24];
        private NPlot.Windows.PlotSurface2D plotSurface2;
        private TextBox textBox1;
        private TextBox textBox2;
        static public short[] DatosVelocidad = new short[48];



        public USBdemo()
```

```
        {
            InitializeComponent();
            //DatuBalioak.Initialize();

        }

        #region Clean up any resources being used.
        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose(disposing);
        }

        #endregion



        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources =
new System.ComponentModel.ComponentResourceManager(typeof(USBdemo));
            this.timer2 = new
System.Windows.Forms.Timer(this.components);
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.button2 = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBoxControlMovPasosDecel = new
System.Windows.Forms.TextBox();
            this.label7 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.textBoxControlMovCorrienteB = new
System.Windows.Forms.TextBox();
            this.buttonControlMovApplicate = new
System.Windows.Forms.Button();
            this.groupBox4 = new System.Windows.Forms.GroupBox();
            this.radioButtonMedioPaso = new
System.Windows.Forms.RadioButton();
            this.radioButtonPasosEnteroUnafase = new
System.Windows.Forms.RadioButton();
            this.radioButtonTipoMovPasoEnt2Fase = new
System.Windows.Forms.RadioButton();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.radioButtonIzquierda = new
System.Windows.Forms.RadioButton();
            this.radioButtonDerecha = new
System.Windows.Forms.RadioButton();
```

```
            this.textBoxControlMovPasosAcel = new
System.Windows.Forms.TextBox();
            this.label8 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.textBoxControlMovVelocidad = new
System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.textBoxControlMovCorrienteA = new
System.Windows.Forms.TextBox();
            this.plotSurface2 = new NPlot.Windows.PlotSurface2D();
            this.groupBox2.SuspendLayout();
            this.groupBox4.SuspendLayout();
            this.groupBox3.SuspendLayout();
            this.SuspendLayout();
            //
            // timer2
            //
            this.timer2.Enabled = true;
            this.timer2.Interval = 500;
            this.timer2.Tick += new
System.EventHandler(this.timer2_Tick);
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.textBox2);
            this.groupBox2.Controls.Add(this.button2);
            this.groupBox2.Controls.Add(this.textBox1);

this.groupBox2.Controls.Add(this.textBoxControlMovPasosDecel);
            this.groupBox2.Controls.Add(this.label7);
            this.groupBox2.Controls.Add(this.label5);

this.groupBox2.Controls.Add(this.textBoxControlMovCorrienteB);

this.groupBox2.Controls.Add(this.buttonControlMovApplicate);
            this.groupBox2.Controls.Add(this.groupBox4);
            this.groupBox2.Controls.Add(this.groupBox3);

this.groupBox2.Controls.Add(this.textBoxControlMovPasosAcel);
            this.groupBox2.Controls.Add(this.label8);
            this.groupBox2.Controls.Add(this.label6);

this.groupBox2.Controls.Add(this.textBoxControlMovVelocidad);
            this.groupBox2.Controls.Add(this.label3);

this.groupBox2.Controls.Add(this.textBoxControlMovCorrienteA);
            this.groupBox2.Location = new System.Drawing.Point(2, 12);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(751, 126);
            this.groupBox2.TabIndex = 18;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Control Movimiento";
            //
            // textBox2
            //
            this.textBox2.Location = new System.Drawing.Point(492,
106);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(100, 20);
            this.textBox2.TabIndex = 24;
            //
```

```
            // button2
            //
            this.button2.Location = new System.Drawing.Point(334, 82);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(46, 32);
            this.button2.TabIndex = 24;
            this.button2.Text = "STOP";
            this.button2.Click += new
System.EventHandler(this.button2_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(386,
106);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(100, 20);
            this.textBox1.TabIndex = 23;
            //
            // textBoxControlMovPasosDecel
            //
            this.textBoxControlMovPasosDecel.Location = new
System.Drawing.Point(466, 61);
            this.textBoxControlMovPasosDecel.Name =
"textBoxControlMovPasosDecel";
            this.textBoxControlMovPasosDecel.Size = new
System.Drawing.Size(60, 20);
            this.textBoxControlMovPasosDecel.TabIndex = 23;
            //
            // label7
            //
            this.label7.AutoSize = true;
            this.label7.Location = new System.Drawing.Point(390, 55);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(70, 26);
            this.label7.TabIndex = 22;
            this.label7.Text = "Pasos\r\nDeceleracion";
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Location = new System.Drawing.Point(570, 56);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(85, 13);
            this.label5.TabIndex = 21;
            this.label5.Text = "Corriente Fase B";
            //
            // textBoxControlMovCorrienteB
            //
            this.textBoxControlMovCorrienteB.Location = new
System.Drawing.Point(667, 53);
            this.textBoxControlMovCorrienteB.Name =
"textBoxControlMovCorrienteB";
            this.textBoxControlMovCorrienteB.Size = new
System.Drawing.Size(61, 20);
            this.textBoxControlMovCorrienteB.TabIndex = 20;
            //
            // buttonControlMovApplicate
            //
            this.buttonControlMovApplicate.Location = new
System.Drawing.Point(215, 82);
```

```
            this.buttonControlMovApplicate.Name =
"buttonControlMovApplicate";
            this.buttonControlMovApplicate.Size = new
System.Drawing.Size(88, 32);
            this.buttonControlMovApplicate.TabIndex = 19;
            this.buttonControlMovApplicate.Text = "Applicate";
            this.buttonControlMovApplicate.Click += new
System.EventHandler(this.button1_Click);
            //
            // groupBox4
            //
            this.groupBox4.Controls.Add(this.radioButtonMedioPaso);

this.groupBox4.Controls.Add(this.radioButtonPasosEnteroUnafase);

this.groupBox4.Controls.Add(this.radioButtonTipoMovPasoEnt2Fase);
            this.groupBox4.Location = new System.Drawing.Point(12,
19);
            this.groupBox4.Name = "groupBox4";
            this.groupBox4.Size = new System.Drawing.Size(184, 95);
            this.groupBox4.TabIndex = 17;
            this.groupBox4.TabStop = false;
            this.groupBox4.Text = "Tipo Moviento";
            //
            // radioButtonMedioPaso
            //
            this.radioButtonMedioPaso.AutoSize = true;
            this.radioButtonMedioPaso.Location = new
System.Drawing.Point(7, 60);
            this.radioButtonMedioPaso.Name = "radioButtonMedioPaso";
            this.radioButtonMedioPaso.Size = new
System.Drawing.Size(80, 17);
            this.radioButtonMedioPaso.TabIndex = 17;
            this.radioButtonMedioPaso.Text = "Medio paso";
            this.radioButtonMedioPaso.UseVisualStyleBackColor = true;
            //
            // radioButtonPasosEnteroUnafase
            //
            this.radioButtonPasosEnteroUnafase.AutoSize = true;
            this.radioButtonPasosEnteroUnafase.Location = new
System.Drawing.Point(94, 15);
            this.radioButtonPasosEnteroUnafase.Name =
"radioButtonPasosEnteroUnafase";
            this.radioButtonPasosEnteroUnafase.Size = new
System.Drawing.Size(82, 30);
            this.radioButtonPasosEnteroUnafase.TabIndex = 16;
            this.radioButtonPasosEnteroUnafase.Text = "Paso
entero\r\n1 fase";
            this.radioButtonPasosEnteroUnafase.UseVisualStyleBackColor
= true;
            //
            // radioButtonTipoMovPasoEnt2Fase
            //
            this.radioButtonTipoMovPasoEnt2Fase.AutoSize = true;
            this.radioButtonTipoMovPasoEnt2Fase.BackColor =
System.Drawing.SystemColors.Control;
            this.radioButtonTipoMovPasoEnt2Fase.Checked = true;

this.radioButtonTipoMovPasoEnt2Fase.FlatAppearance.CheckedBackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))),
((int)(((byte)(192)))), ((int)(((byte)(0)))));
```

```
            this.radioButtonTipoMovPasoEnt2Fase.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.radioButtonTipoMovPasoEnt2Fase.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.radioButtonTipoMovPasoEnt2Fase.Location = new
System.Drawing.Point(6, 15);
            this.radioButtonTipoMovPasoEnt2Fase.Name =
"radioButtonTipoMovPasoEnt2Fase";
            this.radioButtonTipoMovPasoEnt2Fase.Size = new
System.Drawing.Size(81, 30);
            this.radioButtonTipoMovPasoEnt2Fase.TabIndex = 10;
            this.radioButtonTipoMovPasoEnt2Fase.TabStop = true;
            this.radioButtonTipoMovPasoEnt2Fase.Text = "Paso
entero\r\n2 fases";

this.radioButtonTipoMovPasoEnt2Fase.UseVisualStyleBackColor = false;
            //
            // groupBox3
            //
            this.groupBox3.Controls.Add(this.radioButtonIzquierda);
            this.groupBox3.Controls.Add(this.radioButtonDerecha);
            this.groupBox3.Location = new System.Drawing.Point(215,
19);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(169, 48);
            this.groupBox3.TabIndex = 15;
            this.groupBox3.TabStop = false;
            this.groupBox3.Text = "Sentido Giro";
            //
            // radioButtonIzquierda
            //
            this.radioButtonIzquierda.AutoSize = true;
            this.radioButtonIzquierda.BackColor =
System.Drawing.SystemColors.Control;
            this.radioButtonIzquierda.Checked = true;
            this.radioButtonIzquierda.FlatAppearance.BorderSize = 2;
            this.radioButtonIzquierda.FlatAppearance.CheckedBackColor
= System.Drawing.Color.Lime;
            this.radioButtonIzquierda.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.radioButtonIzquierda.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.radioButtonIzquierda.Location = new
System.Drawing.Point(16, 19);
            this.radioButtonIzquierda.Name = "radioButtonIzquierda";
            this.radioButtonIzquierda.Size = new
System.Drawing.Size(58, 17);
            this.radioButtonIzquierda.TabIndex = 9;
            this.radioButtonIzquierda.TabStop = true;
            this.radioButtonIzquierda.Text = "Horario";
            this.radioButtonIzquierda.UseVisualStyleBackColor = true;
            //
            // radioButtonDerecha
            //
            this.radioButtonDerecha.AutoSize = true;
            this.radioButtonDerecha.ForeColor =
System.Drawing.SystemColors.MenuText;
            this.radioButtonDerecha.Location = new
System.Drawing.Point(90, 19);
            this.radioButtonDerecha.Name = "radioButtonDerecha";
```

```
            this.radioButtonDerecha.Size = new System.Drawing.Size(75,
17);
            this.radioButtonDerecha.TabIndex = 8;
            this.radioButtonDerecha.Text = "Antihorario";
            this.radioButtonDerecha.UseVisualStyleBackColor = true;
            //
            // textBoxControlMovPasosAcel
            //
            this.textBoxControlMovPasosAcel.Location = new
System.Drawing.Point(466, 30);
            this.textBoxControlMovPasosAcel.Name =
"textBoxControlMovPasosAcel";
            this.textBoxControlMovPasosAcel.Size = new
System.Drawing.Size(60, 20);
            this.textBoxControlMovPasosAcel.TabIndex = 14;
            //
            // label8
            //
            this.label8.AutoSize = true;
            this.label8.Location = new System.Drawing.Point(390, 24);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size(63, 26);
            this.label8.TabIndex = 13;
            this.label8.Text = "Pasos\r\nAceleracion";
            //
            // label6
            //
            this.label6.AutoSize = true;
            this.label6.Location = new System.Drawing.Point(570, 76);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(79, 26);
            this.label6.TabIndex = 11;
            this.label6.Text = "Velocidad Final\r\n(rpm)";
            //
            // textBoxControlMovVelocidad
            //
            this.textBoxControlMovVelocidad.Location = new
System.Drawing.Point(667, 79);
            this.textBoxControlMovVelocidad.Name =
"textBoxControlMovVelocidad";
            this.textBoxControlMovVelocidad.Size = new
System.Drawing.Size(60, 20);
            this.textBoxControlMovVelocidad.TabIndex = 10;
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(570, 28);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(85, 13);
            this.label3.TabIndex = 4;
            this.label3.Text = "Corriente Fase A";
            //
            // textBoxControlMovCorrienteA
            //
            this.textBoxControlMovCorrienteA.Location = new
System.Drawing.Point(667, 27);
            this.textBoxControlMovCorrienteA.Name =
"textBoxControlMovCorrienteA";
            this.textBoxControlMovCorrienteA.Size = new
System.Drawing.Size(61, 20);
```

```
            this.textBoxControlMovCorrienteA.TabIndex = 3;
            //
            // plotSurface2
            //
            this.plotSurface2.AutoScaleAutoGeneratedAxes = false;
            this.plotSurface2.AutoScaleTitle = false;
            this.plotSurface2.BackColor =
System.Drawing.SystemColors.ControlLightLight;
            this.plotSurface2.DateTimeToolTip = false;
            this.plotSurface2.Legend = null;
            this.plotSurface2.LegendZOrder = -1;
            this.plotSurface2.Location = new System.Drawing.Point(12,
144);
            this.plotSurface2.Name = "plotSurface2";
            this.plotSurface2.RightMenu = null;
            this.plotSurface2.ShowCoordinates = true;
            this.plotSurface2.Size = new System.Drawing.Size(779,
369);
            this.plotSurface2.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.None;
            this.plotSurface2.TabIndex = 22;
            this.plotSurface2.Text = "plotSurface2D1";
            this.plotSurface2.Title = "";
            this.plotSurface2.TitleFont = new
System.Drawing.Font("Arial", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Pixel);
            this.plotSurface2.XAxis1 = null;
            this.plotSurface2.XAxis2 = null;
            this.plotSurface2.YAxis1 = null;
            this.plotSurface2.YAxis2 = null;
            //
            // USBdemo
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.AutoSizeMode =
System.Windows.Forms.AutoSizeMode.GrowAndShrink;
            this.ClientSize = new System.Drawing.Size(804, 525);
            this.Controls.Add(this.plotSurface2);
            this.Controls.Add(this.groupBox2);
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Name = "USBdemo";
            this.Text = "Control motor paso a paso";
            this.Load += new System.EventHandler(this.USBdemo_Load);
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.groupBox4.ResumeLayout(false);
            this.groupBox4.PerformLayout();
            this.groupBox3.ResumeLayout(false);
            this.groupBox3.PerformLayout();
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        public static void Main()
        {
```

```
            Application.Run(new USBdemo());

        }

        private void USBdemo_Load(object sender, System.EventArgs e)
        {

        }

        private void timer2_Tick(object sender, EventArgs e) //Evento
del timer2
        {

                textBox1.Text = usb_int.datos_velocidad().ToString();
                double vel1 = double.Parse(textBox1.Text);
                vel1 = 0.087 * vel1;
                short vel_short = Convert.ToInt16(vel1);
                string vel_salida = Convert.ToString(vel1);
                textBox2.Text = vel_salida;



            plotSurface2.Clear(); //Limpiamos la gráfica 2
            Grid grid2 = new Grid();//Cuadricula para la gráfica 2
            grid2.VerticalGridType = Grid.GridType.Coarse;
            grid2.HorizontalGridType = Grid.GridType.Coarse;
            grid2.MajorGridPen = new Pen(Color.LightGray, 1.0f);
            plotSurface2.Add(grid2); //Introduciomos la cuadricula en
la gráfica

            int len2 = 48; //Longitud Eje abscisas
            string[] s2 = new string[len2]; //Creamos un string del
mismo tamaño que el eje de abscisas
            PlotQEExampleValues2 = new double[len2]; //Double-64 bits
            PlotQEExampleTextValues2 = new string[len2];
            short[] Dato_velocidad = new short[len2]; //Short->int de
16 bits. Aqui irá el valor de la velocidad

            // Representa una lista de objetos con establecimiento
inflexible de tipos a la que se puede
            //obtener acceso por índice. Proporciona métodos para
buscar, ordenar y manipular listas.
            List<short> intList2 = new List<short>();

            intList2.Add(vel_short); //Añadimos a la lista el valor
devuelto por la funcion datos
            //_velocidad, perteneciente a la clase usb_int


            //Copia un intervalo de elementos de una matriz Array
comenzando en el índice de origen
            //especificado y los pega en otra matriz Array comenzando
en el índice de destino especificado
            // La longitud y los índices se especifican como enteros
de 32 bits.
            Array.Copy(DatosVelocidad, 1, Dato_velocidad, 0, 47);
            Dato_velocidad[47] = vel_short; //Guardamos el dato
devuelto por la funcion y los guardamos
            //de recoger el dato de potenciometro y lo reenvia al PC
de nuevo
            Dato_velocidad.CopyTo(DatosVelocidad, 0);
```

```
            for (int i = 0; i < len2; i++)
            {
                s2[i] = i.ToString("0");
            }


            LinePlot pp2 = new LinePlot();
            pp2.DataSource = DatosVelocidad;
            plotSurface2.Add(pp2);

            LabelPointPlot tp2 = new LabelPointPlot();
            tp2.DataSource = PlotQEExampleValues2;
            tp2.TextData = PlotQEExampleTextValues2;
            tp2.LabelTextPosition =
LabelPointPlot.LabelPositions.Above;
            tp2.Marker = new Marker(Marker.MarkerType.None, 10);
            plotSurface2.Add(tp2);

            LabelAxis la2 = new LabelAxis(plotSurface2.XAxis1);
            for (int i = 0; i < len2; ++i)
            {
                la2.AddLabel(s2[i], i);
            }
            FontFamily ff2 = new FontFamily("Verdana");
            la2.TickTextFont = new Font(ff2, 7);
            la2.PhysicalSpacingMin = 51;
            plotSurface2.XAxis1 = la2;



            plotSurface2.Title = "Velocidad motor paso a paso";
            plotSurface2.TitleFont = new Font(ff2, 15);
            plotSurface2.XAxis1.WorldMin = 0;
            plotSurface2.XAxis1.WorldMax = len2;
            plotSurface2.XAxis1.LabelFont = new Font(ff2, 10);
            plotSurface2.XAxis1.Label = "Tiempo(0.5s/paso)";
            plotSurface2.YAxis1.Label = "velocidad (rpm)";
            plotSurface2.YAxis1.LabelFont = new Font(ff2, 10);
            plotSurface2.YAxis1.TickTextFont = new Font(ff2, 10);

            plotSurface2.YAxis1.WorldMin = -125.0;
            plotSurface2.YAxis1.WorldMax = 125.0;//Valores máximos y
mínimos de la gráfica

            plotSurface2.XAxis1.TicksLabelAngle = 60.0f;

            plotSurface2.Refresh();
        }

       //Al clickar sobre el boton de applicate realizaremos las
siguientes acciones

        private void button1_Click(object sender, EventArgs e)
        {

            //Creamos el objeto controlmoviento de la clase
ControlMovimiento
ControlMovimiento controlmovimiento = new ControlMovimiento(true);
```

```
                if
(controlmovimiento.comprobarConvertirGrupoControMov(textBoxControlMovP
asosAcel.Text,
                        textBoxControlMovPasosDecel.Text,
                        textBoxControlMovVelocidad.Text,
                        textBoxControlMovCorrienteA.Text,
                        textBoxControlMovCorrienteB.Text,
                        radioButtonDerecha.Checked,
                        radioButtonTipoMovPasoEnt2Fase.Checked,
                        radioButtonPasosEnteroUnafase.Checked,
                        radioButtonMedioPaso.Checked))
                {
                    usb_int.controlMovUsb(controlmovimiento);
                }

                else
                {

                }
            }

        private void button2_Click(object sender, EventArgs e)
        {
            usb_int.Stop();
        }

    }

   }
```

6.5.2.- ControlMovimiento.cs

```
using System;
using System.Collections.Generic;
//using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using usb_api;

namespace WindowsApplication3
{
    public class ControlMovimiento
    {

        public UInt16
contadorInicialAcelTemp0,contadorInicialDecelTemp0;
        public byte valuePasosAceleracion,valuePasosDeceleracion;
        public UInt16 valueCorrienteFaseA, valueCorrienteFaseB;
        public enum tipoMovimiento : byte
        {
            PASOENTERO1FASE = 0x01,
            PASOENTERO2FASE = 0x02,
            MEDIOPASO = 0x32,
            MICROPASO = 0xEE
        }
        public byte byteTipoMovimiento;
        public byte byteSentidoGiro;
        private bool chekEntradas = true;
        private float valueVelocidadFinal;
        private const int Fosc=31250;        //frequencia del
oscilador del pic del temp Temp0
        private const byte GiroDerechas = 1, GiroIzquierdas = 0;

        private const double ConstContadorAcelDecel=0.26;


        bool[] casoFalloChekEntradas = new bool[3] { true, true, true
};



        const string txantioiCorrientes = @"\b^[0][,][0-5][0-
9]$|\b^[0]$|\b^[0][,][0-6]$";
        const UInt16 PWMdutyMax = 1020;
        const Double CorrienteMaxima = 0.6;
        const int NumPasosAcelDecelMax = 256;
        const string txantioiPasosAcelDecel = @"\b^[0-9]$|\b^[1-9][0-
9]$|\b^1[0-9][0-9]$|\b^2[0-4][0-9]$|\b^25[0-5]$";
        const string txantioiVelocidadFinal = @"\b^[0-9]$|\b^[1-9][0-
9]$|\b^1[0][0]$";

        float valueCorrienteFaseAfloat;
        float valueCorrienteFaseBfloat;
        //This is the constructor
        public ControlMovimiento(bool chekEntradas)
        {
            this.chekEntradas = chekEntradas;
        }
```

```
        //Corrientes entre 0 y 0,6 A con dos decimales y positivas
        public bool comprobarConvertirGrupoControMov(string
stringPasosAceleracion,string stringPasosDeceleracion,string
stringVelocidadFinal,string stringCorrientefaseA, string
stringCorrientefaseB, bool boolSentidoGiroDerechas, bool
boolPasoEnt2Fases, bool boolPasoEnt1Fase, bool boolMedioPaso)

        {

            Regex rgx1 = new Regex(txantioiCorrientes);
            MatchCollection matchesCurrentFaseA =
rgx1.Matches(stringCorrientefaseA);
            MatchCollection matchesCurrentFaseB =
rgx1.Matches(stringCorrientefaseB);
            Regex rgx2 = new Regex(txantioiPasosAcelDecel);
            MatchCollection matchespasosAceleracion =
rgx2.Matches(stringPasosAceleracion);
            MatchCollection matchespasosDeceleracion =
rgx2.Matches(stringPasosDeceleracion);
            Regex rgx3 = new Regex(txantioiVelocidadFinal);
            MatchCollection matchesVelocidadFinal =
rgx3.Matches(stringVelocidadFinal);

            if (matchesCurrentFaseA.Count == 0 ||
matchesCurrentFaseB.Count == 0)
            {
                /*MessageBox.Show("Expresión numérica no válida." +
               "\rExpresiones válidas son del tipo:" +
               "\r 0; 0,1;0,15;...;0,6");*/
                chekEntradas = false;// Operation Failed
                casoFalloChekEntradas[0] = false;

            }
            else casoFalloChekEntradas[0] = true;

            if (matchespasosAceleracion.Count == 0 ||
matchespasosDeceleracion.Count == 0)
            {
                /*bi bihurketa horiek azkenean dena ongi badoa
                 string newValueCorrienteFaseA = stringCorrientefaseA;
                 string newValueCorrienteFaseB = stringCorrientefaseB;
                 */

                chekEntradas = false;// Operation Failed
                casoFalloChekEntradas[1] = false;
            }
            else casoFalloChekEntradas[1] = true;

            if (matchesVelocidadFinal.Count == 0)
            {
                chekEntradas = false;// Operation Failed
                casoFalloChekEntradas[2] = false;
            }
            else casoFalloChekEntradas[2] = true;

            /* Create the event args object (BAINA EZ ORAIN MOTA
HONETAN).

             KorronteenValueUpdatedEventArgs korronteenValueArgs =
```

```
                    new
KorronteenValueUpdatedEventArgs(newValueCorrienteFaseA,
newValueCorrienteFaseB);
                // Inform the observer. Orain egikaratzen da ondoko EVENT
                KorronteenValueUpdated(this, korronteenValueArgs);
                 */
               //to insert values to convert and then pass to the USB
               if (chekEntradas == true)
               {
                   //adapta las corrientes para pasarlas el buffer USB
                   valueCorrienteFaseAfloat =
float.Parse(stringCorrientefaseA);
                   valueCorrienteFaseBfloat =
float.Parse(stringCorrientefaseB);
                   valueCorrienteFaseA =
(UInt16)(valueCorrienteFaseAfloat * PWMdutyMax / CorrienteMaxima);
                   valueCorrienteFaseB =
(UInt16)(valueCorrienteFaseBfloat * PWMdutyMax / CorrienteMaxima);
                   //adapta el núemro de pasos de acel y decel para
pasarlas el buffer USB
                   valuePasosAceleracion =
byte.Parse(stringPasosAceleracion);
                   valuePasosDeceleracion =
byte.Parse(stringPasosDeceleracion);
                   ////adapta la velo final desde rpm para pasarlas el
buffer USB
                   valueVelocidadFinal =
                               (float) (float.Parse(stringVelocidadFinal)
/ 60 * (2 * Math.PI));
                   contadorInicialAcelTemp0 = (UInt16)(0.676*Fosc *
ConstContadorAcelDecel * Math.Sqrt(
                       valuePasosAceleracion / (valueVelocidadFinal *
valueVelocidadFinal)));
                   //Multiplicamos por 0.676  para compensar el error que
se produce en le calculo de C0
                   contadorInicialDecelTemp0 = (UInt16)(0.676*Fosc *
ConstContadorAcelDecel * Math.Sqrt(
                       valuePasosDeceleracion /
(valueVelocidadFinal*valueVelocidadFinal)));
                   if (boolSentidoGiroDerechas == true) byteSentidoGiro =
GiroDerechas;
                   else byteSentidoGiro = GiroIzquierdas;
                   if (boolPasoEnt2Fases == true) byteTipoMovimiento =
(byte)tipoMovimiento.PASOENTERO2FASE;
                   if (boolPasoEnt1Fase == true) byteTipoMovimiento =
(byte)tipoMovimiento.PASOENTERO1FASE;
                   if (boolMedioPaso == true) byteTipoMovimiento =
(byte)tipoMovimiento.MEDIOPASO;
                   MessageBox.Show("bingo");

                   return true;
               }
               else
               {


                       if (casoFalloChekEntradas[0] == false)
                           MessageBox.Show("CorrientefaseAoB");

                       if (casoFalloChekEntradas[1] == false)
```

```
MessageBox.Show("pasosAceleracionDeceleracion");

                    if (casoFalloChekEntradas[2] == false)
                        MessageBox.Show("velocidad máxima superada");

                return false;


            }
        }
    }
}
```

6.5.3.- usb_interface_inigo.cs

```
//Añadimos los namespace que necesitaremos
using System;
using System.Runtime.InteropServices;
using WindowsApplication3;
using PVOID = System.IntPtr;
using DWORD = System.UInt32;


//Creamos nuiiestro namespace usb_api
namespace usb_api
{
    //Creamos la clase usb_interface, la definimos como unsafe para
poder utilizar
    //punteros en ella
    unsafe public class usb_interface
    {
        #region  String Definitions of Pipes and VID_PID
        //string vid_pid_boot= "vid_04d8&pid_000b";    // Bootloader
vid_pid ID
        string vid_pid_norm = "vid_04d8&pid_000c";

        string out_pipe = "\\MCHP_EP1"; // Define End Points
        string in_pipe = "\\MCHP_EP1";
        #endregion

        #region Imported DLL functions from mpusbapi.dll
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDLLVersion();
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDeviceCount(string
pVID_PID);
        [DllImport("mpusbapi.dll")]
        private static extern void* _MPUSBOpen(DWORD instance, string
pVID_PID, string pEP, DWORD dwDir, DWORD dwReserved);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBRead(void* handle, void*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBWrite(void* handle, void*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
```

```
        private static extern DWORD _MPUSBReadInt(void* handle, DWORD
dwLen, DWORD* pLength, DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern bool _MPUSBClose(void* handle);
        #endregion

        void* myOutPipe;
        void* myInPipe;

        private void OpenPipes()
        {
            DWORD selection = 0; // Selects the device to connect to,
in this example it is assumed you will only have one device per
vid_pid connected.

            myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe,
0, 0);
            myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1,
0);
        }

        private void ClosePipes()
        {
            _MPUSBClose(myOutPipe);
            _MPUSBClose(myInPipe);
        }
        //SendReceivePacket(send_buf,4,receive_buf,&RecvLength) == 1
        private DWORD SendReceivePacket(byte* SendData, DWORD
SendLength, byte* ReceiveData, DWORD* ReceiveLength)
        {
            uint SendDelay = 1000;
            uint ReceiveDelay = 1000;

            DWORD SentDataLength;
            DWORD ExpectedReceiveLength = *ReceiveLength;

            OpenPipes();

            if (_MPUSBWrite(myOutPipe, (void*)SendData, SendLength,
&SentDataLength, SendDelay) == 1)
            {

                if (_MPUSBRead(myInPipe, (void*)ReceiveData,
ExpectedReceiveLength, ReceiveLength, ReceiveDelay) == 1)
                {
                    if (*ReceiveLength == ExpectedReceiveLength)
                    {
                        ClosePipes();
                        return 1;   // Success!
                    }
                    else if (*ReceiveLength < ExpectedReceiveLength)
                    {
                        ClosePipes();
                        return 2;   // Partially failed, incorrect
receive length
                    }
                }
            }
            ClosePipes();
            return 0;  // Operation Failed
        }
```

```
        public DWORD GetDLLVersion()
        {
            return _MPUSBGetDLLVersion();
        }

        public DWORD GetDeviceCount(string Vid_Pid)
        {
            return _MPUSBGetDeviceCount(Vid_Pid);
        }

        public short datos_velocidad()
        {
            byte* send_buf = stackalloc byte[64];
            byte* receive_buf = stackalloc byte[64];

            short ValorVelocidad;
            byte[] byteArray2 = { 0, 0 };
            DWORD RecvLength = (byte)3;
            send_buf[0] = 0xA3;                 //Orden al micro de que
coja la velocidad
            //send_buf[1] = (byte)num1;
            //send_buf[2] = (byte)num2;

            if (SendReceivePacket(send_buf, 1, receive_buf,
&RecvLength) == 1)
            {

                byteArray2[0] = *(receive_buf + 1);
                byteArray2[1] = *(receive_buf + 2);
                ValorVelocidad = BitConverter.ToInt16(byteArray2, 0);
                //ValorVelocidad= 100;
                return ValorVelocidad; //Valor devuelto por el micro
en formato int16
            }

            else   return 50;


        }

        public uint controlMovUsb(ControlMovimiento e)
        {
            uint Temp=0;

            byte* send_buf = stackalloc byte[64];
            byte* receive_buf = stackalloc byte[64];

            byte[] contadorAcelTemp0 = new byte[2];
            byte[] contadorDecelTemp0 = new byte[2];
            byte[] sendcurrentA = new byte[2];
            byte[] sendcurrentB = new byte[2];
            contadorAcelTemp0 =
BitConverter.GetBytes(e.contadorInicialAcelTemp0);
            contadorDecelTemp0 =
BitConverter.GetBytes(e.contadorInicialDecelTemp0);
            sendcurrentA =
BitConverter.GetBytes(e.valueCorrienteFaseA);
            sendcurrentB =
BitConverter.GetBytes(e.valueCorrienteFaseB);
```

```
            DWORD RecvLength = (byte)3;
            send_buf[0] =0xA0;
            send_buf[1] = contadorAcelTemp0[0];
            send_buf[2] = contadorAcelTemp0[1];
            send_buf[3] = contadorDecelTemp0[0];
            send_buf[4] = contadorDecelTemp0[1];
            send_buf[5] = e.byteSentidoGiro;
            send_buf[6] = e.valuePasosAceleracion;
            send_buf[7] = e.valuePasosDeceleracion;
            send_buf[8] = e.byteTipoMovimiento;
            send_buf[9]  = sendcurrentA[0];
            send_buf[10] = sendcurrentA[1];
            send_buf[11] = sendcurrentB[0];
            send_buf[12] = sendcurrentB[1];

            if (SendReceivePacket(send_buf, 13, receive_buf,
&RecvLength) == 1)
            {
                Temp = (uint)receive_buf[1];
            }
            return Temp;
        }

        public uint Stop() {

            byte* send_buf = stackalloc byte[64];
            byte* receive_buf = stackalloc byte[64];

            DWORD RecvLength = 3;

            send_buf[0] = 0xA1;                //Command for
TIMER0_OFF


            if (SendReceivePacket(send_buf, 1, receive_buf,
&RecvLength) == 1)
            {
                if (RecvLength == 1 && receive_buf[0] == 0xA1)
                {
                    return 0;
                }
                else
                {
                    return 2;
                }
            }
            else
            {
                    return 1;
            }
        }

        }
    }
```

7.- Funcionamiento global de sistema de control de un motor paso a paso

Hasta ahora se ha estudiado cada una de las diferentes partes que forman el sistema, pero para tener una idea clara y precisa de la forma de trabajar del sistema, se debe tener una perspectiva sencilla y externa.

En primer lugar para tener una visión clara del funcionamiento global del sistema recordemos lo elementos que lo componen.

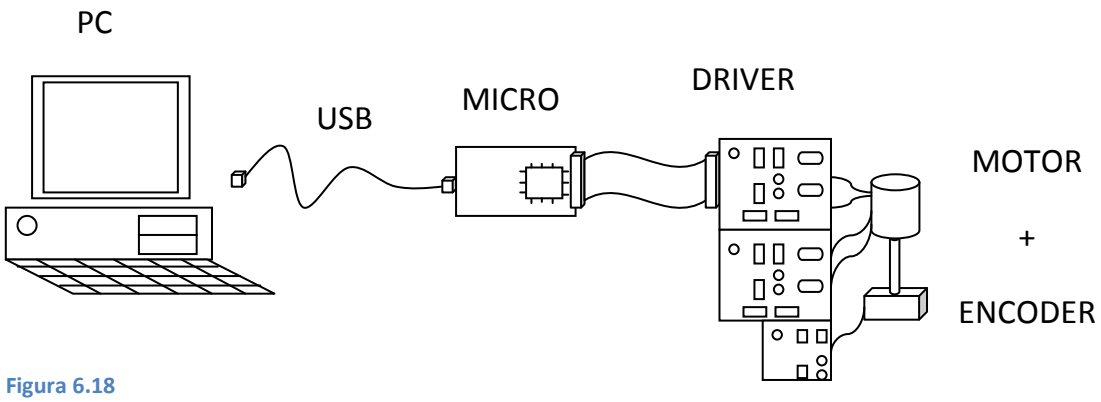


Figura 6.18

- *Ordenador*: Es el centro de control. En él se hace correr el programa que controla la configuración, mando y visualización del motor paso a paso.

- *Comunicación USB*: Es el protocolo de comunicación entre el ordenador y el microprocesador.

- *Microprocesador*: Es el interlocutor entre el ordenador y los drivers del motor paso a paso.

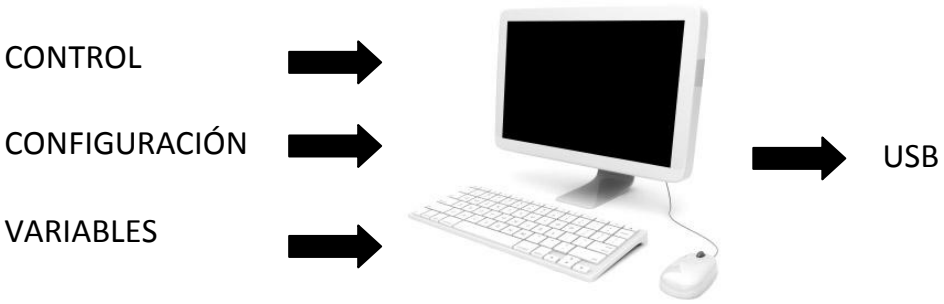
- *Driver*: Es la circuitería electrónica que crea las señales necesarias para el control del motor.

- *Motor & Encoder*: Un motor paso a paso y su encoder.

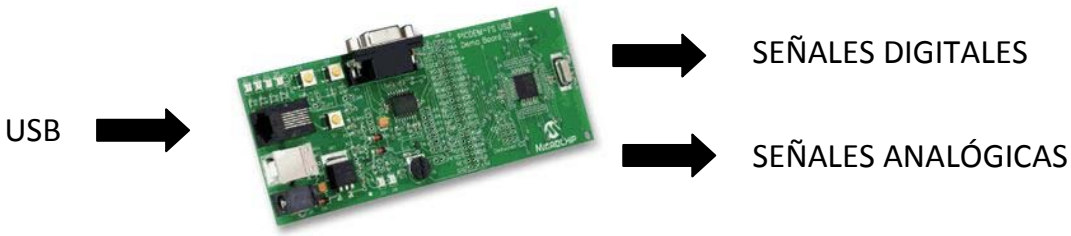
Este apartado se estructura siguiendo el flujo de información, primero desde el ordenador hasta el motor y posteriormente desde el encoder hasta la visualización de la información en el ordenador.

7.1.- Flujo de datos en el encendido

Como anteriormente se ha mencionado el control del motor la realizamos en el ordenador, al igual que la configuración y carga de variables.



Cuando la configuración y variables están listas, mandamos la orden de puesta marcha, lo que supone el envío por USB de la orden y la información necesaria.



Llegados a este punto el microprocesador ha recibido la orden de puesta en marcha del motor y toda la información necesaria.

El microprocesador actuará en primer lugar configurando las señales PWM, que serán las que, tras filtrar, constituyan las dos señales analógicas que indiquen a los drivers la corriente que debe circular por cada fase.

Tras lo cual, mediante la utilización de un temporizador, creará las correspondientes señales de salida digitales, dependiendo de la configuración del motor, que irán incrementándose en el tiempo hasta llegar a la frecuencia indicada por la velocidad final del motor.

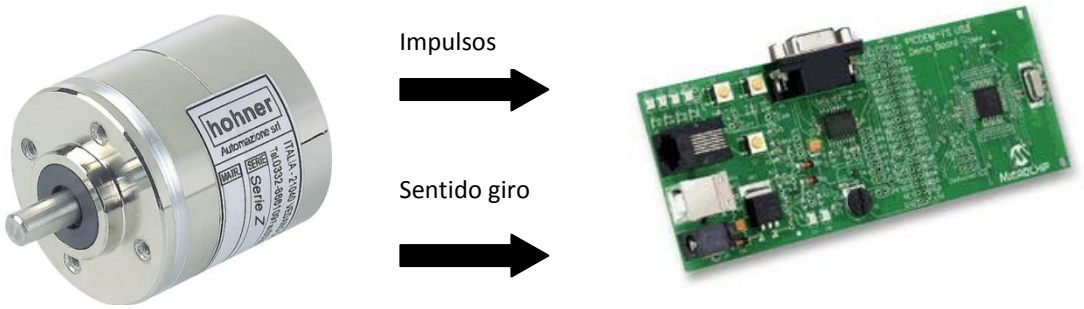
7.2.- Flujo de datos en el apagado

Como se ha explicado en el apartado 6.3.3. cuando desde el ordenador se pulsa sobre la orden detención “Stop” el programa mandará la correspondiente orden al micro procesador a través del puerto USB.

Cuando el microprocesador reciba esa orden comenzará la reducción de la frecuencia que controla los pasos del motor, independientemente del modo en el que esté trabajando, en el número de pasos de deceleración que se le haya indicado previamente, llegando a la detención de las señales digitales, tras ello se apagarán las señales PWM

7.3.- Flujo de datos desde el encoder

Se dispone de un encoder acoplado al motor que genera 345 impulsos por vuelta, estos impulsos son enviados al driver del encoder que multiplica esta cantidad de impulsos por cuatro, logrando una resolución de 1380 impulsos por vuelta.

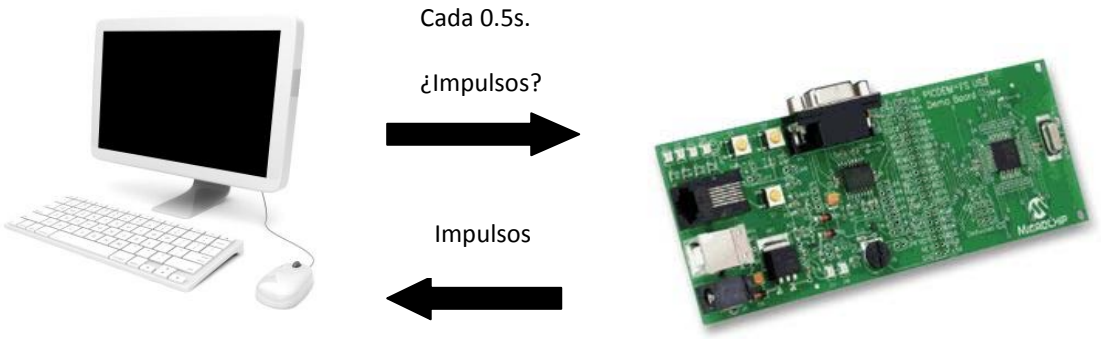


Cada impulso supone un flanco de bajada que es enviado al microprocesador, después de pasar por el driver del encoder, encargado de aislar la señal y cuadriplicar la señal, junto a esta señal enviamos también una señal de sentido de giro, “0” en caso de sentido anti horario y “1” (5).

Cuando un impulso proveniente del encoder llega al microprocesador se produce una interrupción y en función del sentido de giro que recibe el

microprocesador en ese momento sumaremos o restaremos un valor en el contador de impulsos, una variable del microprocesador.

Por otro lado se encuentra el temporizador del interfaz de control del motor paso a paso en el ordenador, el cual cada medio segundo pregunta al microprocesador el número de pasos que lleva contados.



De esta forma el microprocesador envía cada medio segundo el número de pulsos que ha contado, para finalmente en el programa de interfaz transformar este valor a revoluciones por minuto y poder mostrarlo a través de la gráfica.

8.- Conclusiones

El proyecto realizado es el resultado de la idea de controlar un motor paso a paso desde un ordenador. ¿Cómo lo hacemos? ¿Qué herramientas utilizamos? ¿Por dónde empezamos? Estas y muchas más son las preguntas que se plantean inicialmente, resolverlas son algunos de los pasos más importantes y enriquecedores del proyecto.

En todo momento se ha intentado usar herramientas actuales y fácilmente actualizables, lo que nos ha permitido adquirir o mejorar conocimientos de diferentes ramas de la ingeniería.

Por un lado los motores paso a paso, previamente teníamos unos conocimientos básicos sobre este tipo de motores, pero el hecho de tener que trabajar con uno de ellos exige un mayor detalle de estudio sobre este en particular, pero también sobre otros modelos, a la vez que asienta los conocimientos previos, como estructura de montaje, configuraciones, modos de control, funcionamiento…

El diseño de los drivers no ha sido realizado por nosotros, pero su estudio y comprensión en profundidad ha sido imprescindible para la realización del proyecto, lo que nos ha obligado a poner a prueba nuestros conocimientos de electrónica digital y analógica.

El microprocesador PIC18F4550 surgió como respuesta a la pregunta de cómo mandar las correspondientes consignas a los drivers y poder recibir señales por parte del encoder a la vez que fuese capaz de comunicarse con el ordenador mediante una comunicación actual como es el protocolo USB 2.0, este microprocesador era la respuesta perfecta. Los microprocesadores son uno de los grandes protagonistas de nuestras vidas en este siglo XXI, su uso en la vida cotidiana e industrial esta a la orden del día, mediante este proyecto nos hemos acercado a su uso práctico, a su programación, a su funcionamiento, pero sobre todo nos ha dado una perspectiva clara de sus posibilidades y limitaciones, un conocimiento realmente útil.

USB, no pasa un día sin que oigamos esta palabra, se encuentra en nuestras vidas tanto o más que los microprocesadores, pero realmente no sabemos sobre él más allá de que es un puerto del ordenador para conectar dispositivos,

y aunque es un protocolo complejo este proyecto nos ha permitido abrir sus puertas y poder trabajar con él a un nivel más profundo que el de un usuario.

Por último tenemos el framework Microsoft .NET, un conjunto de herramientas para el desarrollo de software enfocado principalmente al creciente mercado de los negocios en entornos Web. Nosotros hemos trabajado con una herramienta en particular (Microsoft visual C#), por lo que hemos aprendido un lenguaje de programación moderno y con perspectiva de futuro, que asimismo se encuentra dentro de un marco mucho más amplio, lo cual aparte del conocimiento específico de programación, nos ha ofrecido una visión mucho más extensa y clara del entramado Microsoft .NET.

El proyecto ha supuesto un desarrollo técnico en electrónica, programación en diferentes lenguajes y comunicación digital, pero también ha sido un trabajo de búsqueda de información, de enfocar y solucionar problemas, de organizar y optimizar la presentación, sin duda todo esto hace de este proyecto una completa aplicación práctica de conocimientos de ingeniería.

9.- Líneas futuras

Este proyecto ha sido fundamentalmente técnico y enfocado al manejo de un motor paso a paso específico, pero la velocidad de avance de la tecnología es altísima, por lo que hemos intentado facilitar futuras ampliaciones o actualizaciones del conjunto del proyecto o de cualquiera de sus partes.

Futuras ampliaciones como el control de diferentes motores simultáneamente, o el control de otro tipo de motor las cuales supondrían cambios mínimos, debido a que las herramientas serían las mismas, ordenador con su aplicación basada en Microsoft Visual C#, comunicación USB y microprocesador.

Las posibilidades de un ordenador, junto con la aplicación en Visual C#, más la comunicación USB y el microprocesador son casi infinitas, se pueden encontrar aplicaciones en el control de otro tipo de motores, en la domótica , en la robótica, etc..

10.- Bibliografía

- **Microchip USB device Firmware Framework User's Guide**

- **Mplab_c18_libraries_51297f**

- **PIC18F4550 Data Sheet**

- **PICDEM FS USB user's guide**

- **Ansi C Programming Language**
  Brian W. Kernghan / Dennis M. Ritchie
  Prentice hall software series

- **C: MANUAL DE REFERENCIA**
  Herbert Schildt
  Ed McGraw-Hill

- **Stepping motors and their microprocessor controls**
  Takashi Kenjo
  Oxford Science Publications

- **Visual C# .NET Programación**
  Francisco Charte Ojeda
  Anaya

- **http://www.microsoft.com/express/windowsdevelopment/**

- **Pro C# 2008 and the.NET 3.5 Platform Fourth Edition**
  Andrew Troelsen