E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

Control motor brushless sensorless

Grado en Ingeniería Eléctrica y Electrónica

Trabajo Fin de Grado

Gonzalo Solchaga Pérez de Lazárraga

Jesús María Corres Sanz

Junio 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Me gustaría agradecer el apoyo técnico ofrecido por el director del proyecto y otros profesores de la Escuela ya que han aportado en mayor o menor medida nuevas experiencias y formas de abordar los problemas que han ido surgiendo durante el transcurso del proyecto.

También agradezco el apoyo de la familia y antes de comenzar citar una frase muy útil en una ingeniería:

"Keep It Simple, Stupid"- Kelly Johnson

Abstract

The project involves the creation of a speed controller circuit for a brushless sensorless motor. The caracteristic factor of this type of electric motors is that they don't have brush to change the polarity of the windings and they have no sensors to know the real position of the rotor. The speed controller circuit is specific for these electric motors that his application is the Unmanned Aircraft and they need a different design from a biggest electric brushless motor. The micro controller that will be used at this project will be from a PIC Microchip family. Thus the designed card could be used in practice of Microprocessors for learning control of electric motors such subject.

Resumen

El proyecto consiste en la creación de un circuito capaz de controlar la velocidad de un motor brushless sensorless. Este tipo de motores eléctricos tienen como característica que no tienen escobillas para cambiar la polaridad del bobinado de su interior y tampoco precisan de un sensor que indique que ha realizado una vuelta. Los motores brushless que son controlados por este tipo de circuitos son específicos para aeronaves no tripuladas y requieren un diseño diferente a un motor brushless pero de mayor tamaño. El micro controlador que se empleará en este proyecto pertenece a la familia PIC de Microchip. De esta forma la tarjeta diseñada podría ser empleada en prácticas de la asignatura Microprocesadores para el aprendizaje del control de este tipo de motores.

Key words: ESC, BRUSHLESS, SENSORLESS, H BRIDGE.

Índice

Introducción

Este proyecto surge a raíz de la idea de crear una aeronave no tripulada de cuatro motores brushless la cual necesita un circuito capaz de regular la velocidad de los motores. El diseño y control de un motor brushless sensorless se ha realizado íntegramente en la universidad, haciendo uso del laboratorio de electrónica básica para el montaje y prueba de los circuitos.

Este tipo de motores eléctricos "brushless", traducido al castellano "sin escobillas", no poseen escobillas para cambiar la polaridad del bobinado que hay en su interior. A su vez, son característicos por no tener un sensor, como por ejemplo un sensor efecto hall, que indique cuándo el eje del motor ha realizado una vuelta completa. Por ello se denominan motores "sensorless", traducido al castellano "sin sensor". Como la aplicación práctica de los motores son las aeronaves no tripuladas, las tensiones de alimentación rondan los 10-15 voltios. Esto implica un control de los motores ligeramente diferente a un control de motores de mayor tamaño. Se empleará un micro controlador junto con un puente en H para crear las formas de onda que alimenten al motor y controlar su velocidad.

Justificación y objetivos

En vista de lo indicado en la introducción, un circuito de control de velocidad es imprescindible a la hora de montar una aeronave no tripulada y debido a que el micro controlador que los gobierna pertenece a la familia PIC de Microchip, este proyecto podrá servir como prácticas para la asignatura de Microprocesadores. En este proyecto los objetivos serán:

- Creación de un circuito controlador de un motor brushless sensorless que lo gobierne un micro controlador perteneciente a la familia PIC de Microchip.
- Control en lazo cerrado de un motor brushless sensorless.
- Comunicación del micro controlador con el ordenador.

Contexto tecnológico

Hoy en día en el mercado existen multitud de circuitos de control de estos motores burshless sensorless para aeronaves no tripuladas.

Este proyecto ha tomado como referencia el circuito de control del siguiente producto comercial:

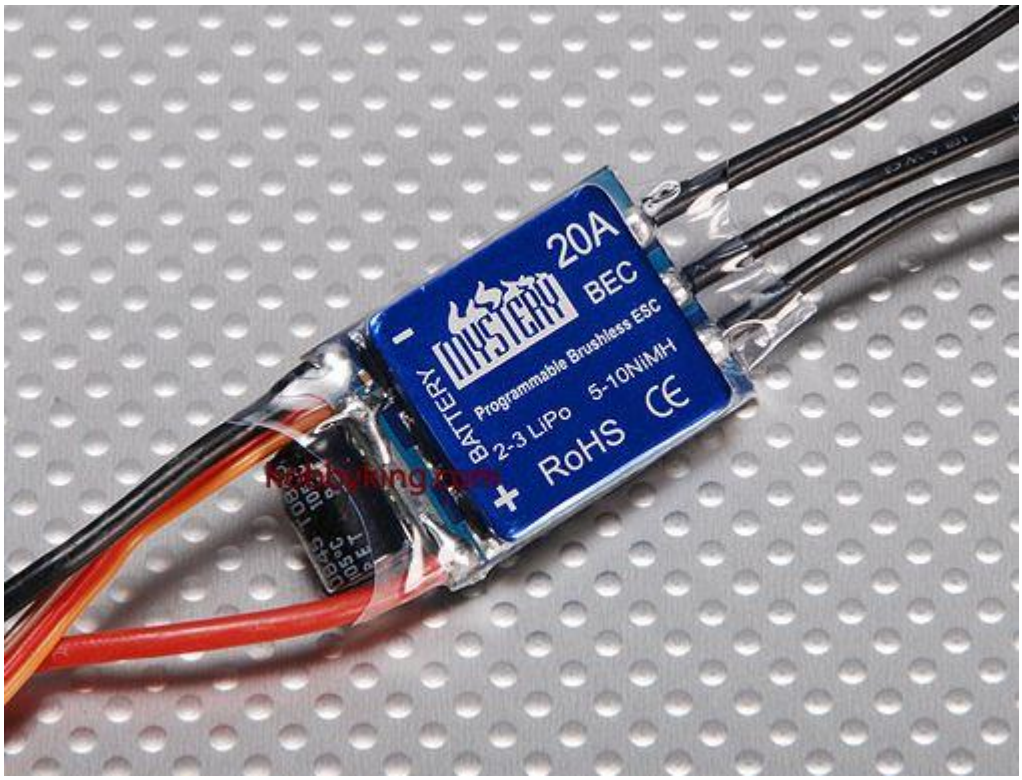

Imagen del circuito de control del motor brushless comercial. [1]

Es un circuito de control de un motor brushless sensorless con una corriente máxima permitida de 20A. Este circuito lleva integrado un regulador lineal que permite alimentar el circuito de control de la aeronave no tripulada a 5V y 2A, a los circuitos que llevan integrado este regulador se les llama BEC, Battery Eliminator Circuit. Otros circuitos de control en cambio no lo llevan. Para finalizar con el hardware, este circuito de control activa los transistores del puente en H que lleva integrado, directamente desde el micro controlador. El micro controlador suele ser de tecnología ARM y de la familia ATMEL.

En cuanto a software, estos circuitos llevan implementado un control en lazo cerrado de la velocidad del motor brushless sensorless por medio del control de la fuerza contra electromotriz inducida en el bobinado del motor cuando se encuentra en movimiento. No controlan la corriente en lazo cerrado pero sí detectan si la batería se está agotando para no dañarla. También al conectarse con la emisora la aeronave no tripulada, mandan un pulso al motor para que emita un pitido cuando reciba correctamente la señal al inicio.

El caso comentado es un circuito de control genérico ya que la diferencia básica de este con otros modelos reside en los amperios que permiten conducir. En modelos más costosos se pueden encontrar mejoras en el software como controlar un frenado del motor o incluso frenados regenerativos.

Todos los circuitos se encuentran preparados para recibir una PWM por un pin desde el circuito de control de la aeronave no tripulada y de ahí calcular la velocidad necesaria que debe llevar el motor.

Metodología empleada

Para el diseño del circuito ESC se hace uso del software, proporcionado por la empresa RS Components, DesignSpark. En él se realiza el diseño del circuito impreso, también llamado "PCB" donde se emplazarán los componentes.

El software empleado para programar el micro controlador es el MPLAB IDE v8.86 con la extensión MPLAB C18 ya que con el software original sin la extensión no se pueden programar micro controladores de gamas superiores al PIC 16.

Para realizar las pruebas del diseño se emplean un osciloscopio de dos canales, una fuente de alimentación con 2A de corriente máxima y otra fuente de alimentación con 5A de corriente máxima. Se requiere de un programador ICD 3 para programar el micro controlador de la familia PIC de Microchip con el ordenador.

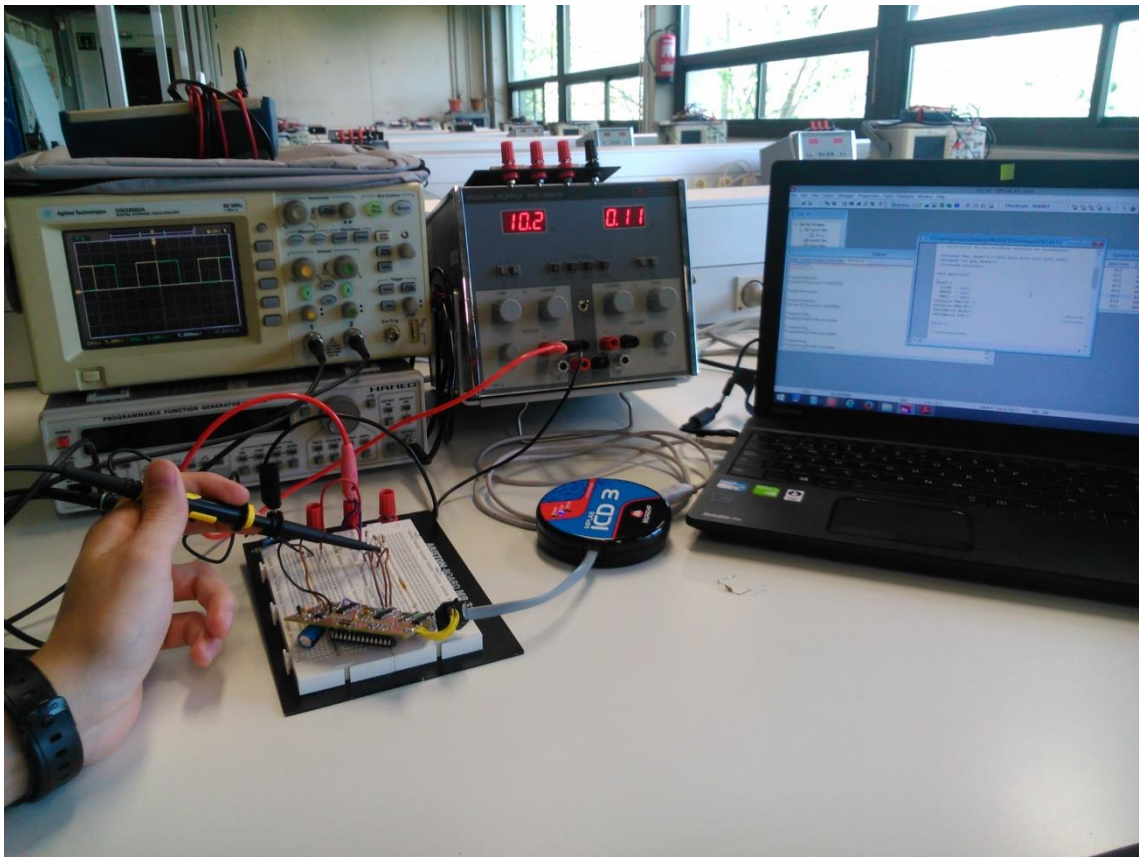

Imagen 1. El laboratorio donde se ha realizado el proyecto junto con los materiales empleados.

Cuerpo del trabajo

1. Motor Brushless

Los motores brushless son motores eléctricos. Constan de un rotor formado por una serie de imanes permanentes y un estator bobinado. Los imanes permanentes en el rotor es la característica principal de estos motores y por los que se les llama “sin escobillas” o brushless. Al tener un campo magnético constante creado por los imanes, no hay necesidad de un bobinado y escobillas para conducir la corriente al rotor. En la Figura 1 se puede observar el despiece del motor eléctrico brushless. Cabe destacar que en estos motores el estator que es estático se encuentra en el interior y el rotor que es móvil es la carcasa unida al eje del motor. El estator tiene 3 bobinados independientes que son las 3 fases del motor.

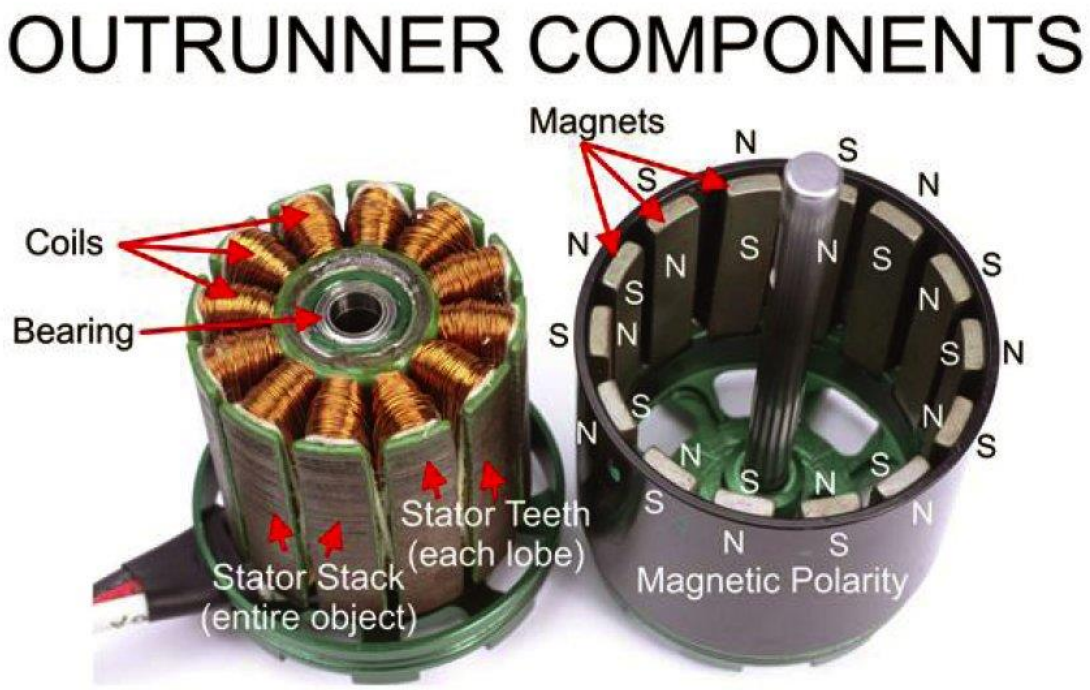

Figura 1. Motor brushless.

El funcionamiento de estos motores es similar a otros motores eléctricos de imanes permanentes. Al energizar una bobina, esta crea un campo magnético. El rotor que tiene un campo magnético constante, detecta la variación y tiende a alinear el campo creado por el estator y el propio haciendo girar el rotor ya que es la parte móvil del motor. Para lograr que el rotor siga girando, antes de que se alinee por completo la bobina energizada con el rotor, se energiza la bobina que le sigue y a la anterior se la deja de alimentar. Esto provoca que el campo magnético del rotor siga al campo magnético del estator, que va variando en el tiempo, haciendo que el rotor gire.

A continuación se mostrarán en las Ilustraciones 1 y 2 unas imágenes que representan el funcionamiento del motor al transmitir la serie de pulsos por las fases. Los pulsos que envía el micro controlador sirven para encender o apagar los transistores del puente en H de tal forma que la tensión en cada una de las tres bobinas del motor varíe creando un campo magnético giratorio con una velocidad determinada por la frecuencia de los pulsos que se alimenta cada fase.

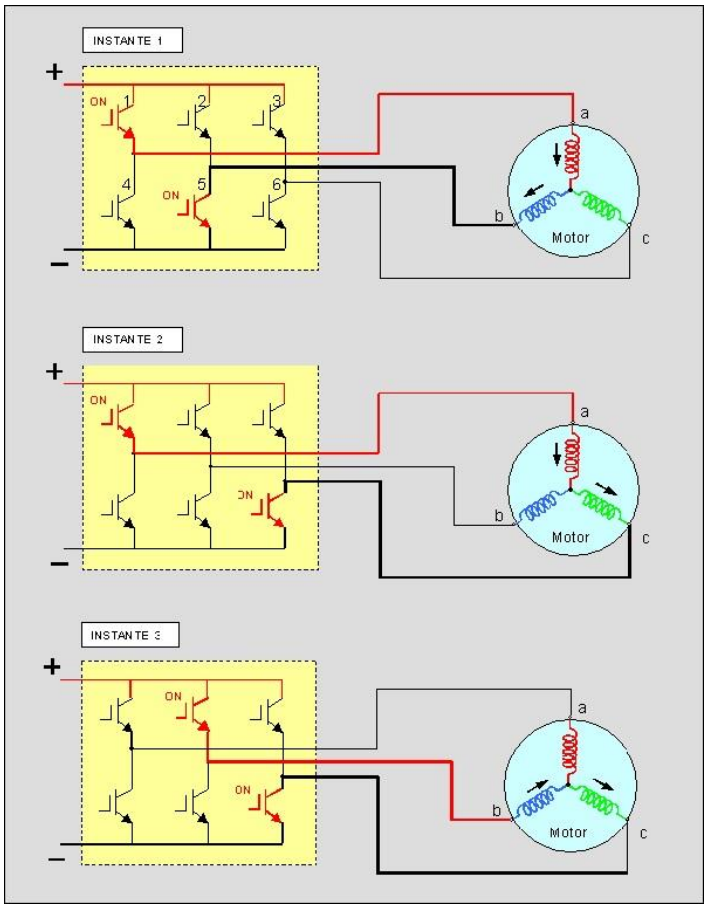

Ilustración 1. Funcionamiento del motor.

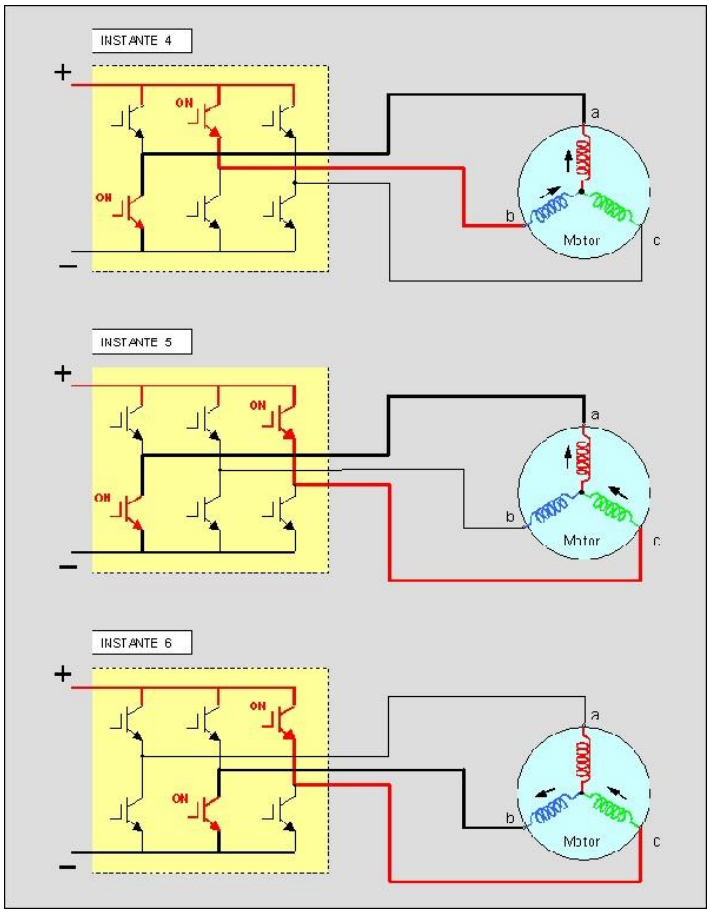

Ilustración 2. Funcionamiento del motor.

Si se desea que el motor gire a una velocidad determinada, se alimentan las fases del motor a una frecuencia determinada por la Ecuación 1 donde Ω es la frecuencia mecánica en radianes por segundo, *f* es la frecuencia eléctrica en Herzios y *p* es en número de pares de polos del rotor. Para alimentar de forma correcta las fases, se deben crear una serie de pulsos para cada fase que serán iguales pero con un desfase de 120º con respecto al anterior según indica la Figura 2.

$$\Omega = \frac{2\pi * f}{p}$$

Ecuación 1. Relación entre frecuencia mecánica y eléctrica.

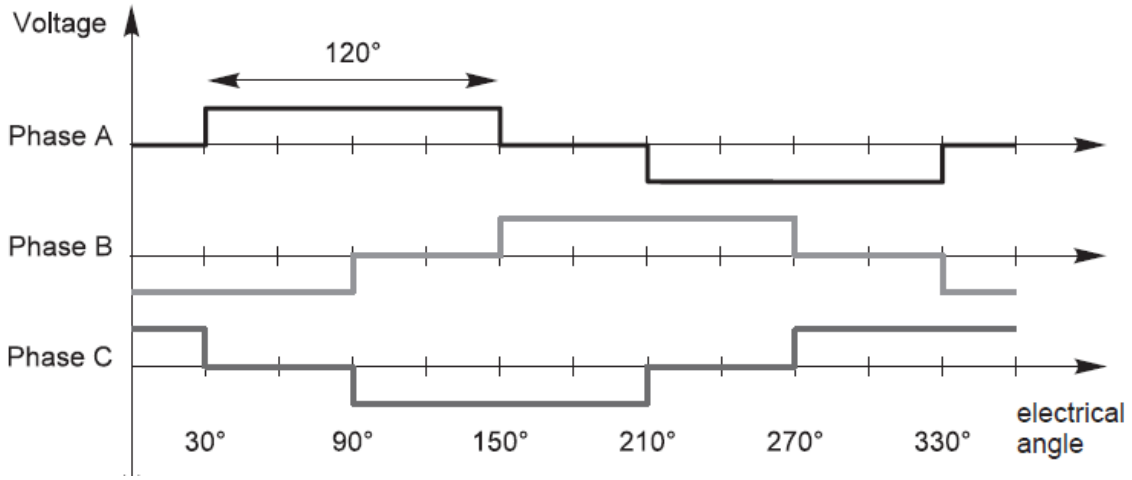

Figura 2. Forma de los pulsos que alimentan el motor.

En el arranque del motor, se debe ir incrementando gradualmente la frecuencia de los pulsos que alimentan al estator para que el motor no pierda el sincronismo. A su vez, la corriente en el arranque es muy grande debido a la baja impedancia de los bobinados y debe controlarse para evitar posibles daños en el motor.

Cuando el motor se encuentra en movimiento, los imanes permanentes del rotor generan una corriente en los bobinados que no se encuentran energizados debido a la variación de flujo magnético, según la Ley de Faraday-Lenz. Esa corriente en el bobinado generada crea una tensión denominada Fuerza Contra electromotriz Inducida, en inglés Back Electromotive Force (BEMF), que es directamente proporcional a la velocidad del motor mediante un coeficiente Kv. Por medio de la Ecuación 2, se puede relacionar en régimen permanente la velocidad del motor con la fuerza contra electromotriz.

$$Velocidad(RPM) = Kv * Tensión(V)$$

$$BEMF = \frac{Velocidad(RPM)}{Kv}$$

Ecuación 2. Relación entre velocidad y tensión en la bobina.

El coeficiente Kv es característico de cada motor y en este tipo de motores se proporciona con las unidades de revoluciones por minuto por voltio de tal forma que conociendo la tensión a la que se va a alimentar el motor, se conocerá la velocidad del mismo.

A continuación se mostrarán los datos que se proporcionan del motor que ha sido escogido para el proyecto:

- Coeficiente Kv(RPM/V): 935.
- Batería que puede alimentar el motor: 2-4s.
- Potencia máxima: 200W.
- Máxima corriente (10s): 15A.
- Resistencia interna: 0.180Ω.
- Número de polos: 14.
- Corriente sin carga: 0.4A.
- Dimensiones: 27.9 x 25.7 mm.

Aplicando la Ecuación 2 y conocidos el coeficiente Kv y la tensión de alimentación nominal de la batería se puede obtener la velocidad máxima:

$$V(RPM) = 935 * 11.1 = 10.378{,}5\ RPM$$

Por medio de la Ecuación 1 se obtiene:

$$f = \frac{7 * 10.378{,}5}{60} = 1.210{,}825\ Hz$$

De esta forma se conoce la frecuencia máxima que deben tener los pulsos que alimentan las fases del motor.

Imagen 2. Motor brushless sensorless.

2. Inversor puente en H

Para poder alimentar el motor con corrientes trifásicas se emplea un inversor de puente en h. El puente en h es un inversor trifásico que consta de tres células elementales de conmutación.

Cada célula de conmutación consta de dos transistores MOSFET, en la parte superior un transistor MOSFET de canal P y en la parte inferior uno de canal N como se muestra en la Figura 4. Célula elemental de conmutación. El empleo de dos tecnologías diferentes permite que la corriente circule en un sentido u otro en función de qué transistor se encuentra activado. En la Figura 3 se representan los esquemáticos de los transistores MOSFET canal P y N y el sentido de la corriente. La S se llama fuente, la D drenador y la G puerta. Por medio de la tensión que se aplica a la puerta del transistor MOSFET se logra que éste entre en la región de conducción donde permite el paso de la corriente o la región de corte donde no permite el paso de la corriente.

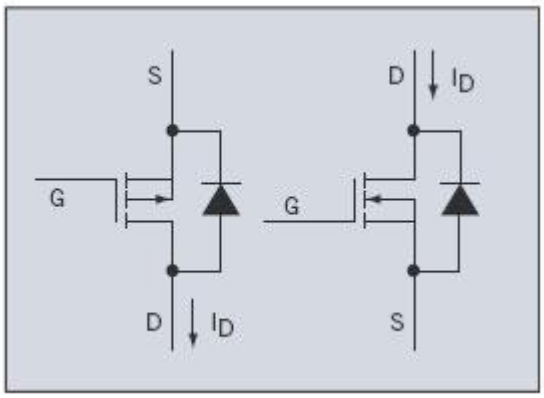

Figura 3. Esquemático de un transistor MOSFET canal P izquierda y canal N derecha.

Se emplean este tipo de transistores y no los transistores bipolares debido a que los transistores MOSFET se controlan por medio de tensión y los transistores bipolares por medio de corriente. Como se desea que el micro controlador sea quien active o desactive los transistores y no es capaz de entregar más de 20mA de corriente, este tipo de transistores son la mejor opción para realizar el puente en H.

En esta célula elemental se conecta la parte superior a la alimentación y la inferior a masa. La parte intermedia se conecta a una fase, de esta forma, juntando tres células de conmutación en paralelo se completa el puente alimentando las tres fases del motor.

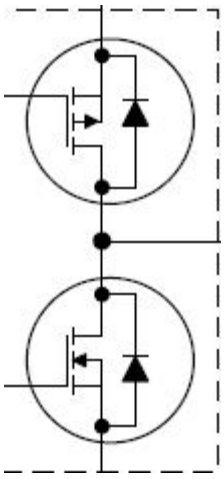

Figura 4. Célula elemental de conmutación.

Estos motores consumen grandes corrientes de tal forma que por mayor seguridad y disminuir el esfuerzo de los componentes, para cada fase del motor se colocan dos células elementales en paralelo quedando el puente en h tal y como se muestra en la Figura 5. 2 Células elementales de conmutación en paralelo.

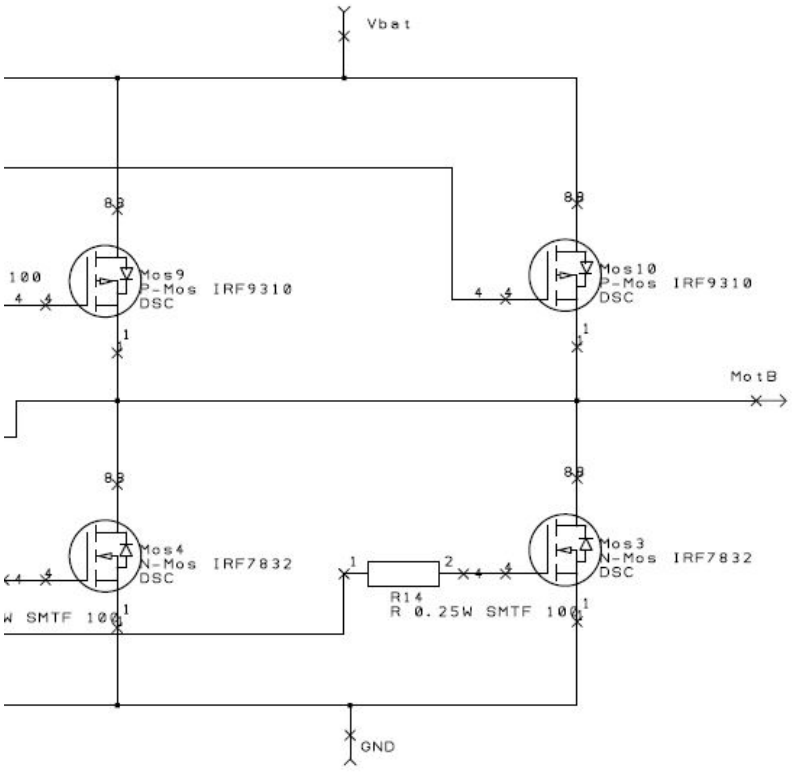

Figura 5. 2 Células elementales de conmutación en paralelo.

Por lo general entre el micro controlador que genera las señales de encendido y apagado de los transistores y los transistores hay unos drivers. Son dispositivos que proporcionan la corriente y tensión necesaria a las puertas de los transistores para que estos se activen ya que en muchas ocasiones la corriente y tensión que proporciona la

salida digital de un micro controlador no es suficiente para llevar a los transistores a la activación completa. En este primer diseño, no se han introducido estos componentes sino que se ha tomado directamente la salida del micro controlador para activar los transistores MOSFET de canal N, Figura 6.Circuito encendido transistor MOSFET canal N., y para los transistores MOSFET de canal P se ha introducido un transistor bipolar entre el micro controlador y la puerta del transistor como se puede apreciar en la Figura 7. Circuito encendido transistor MOSFET canal P.

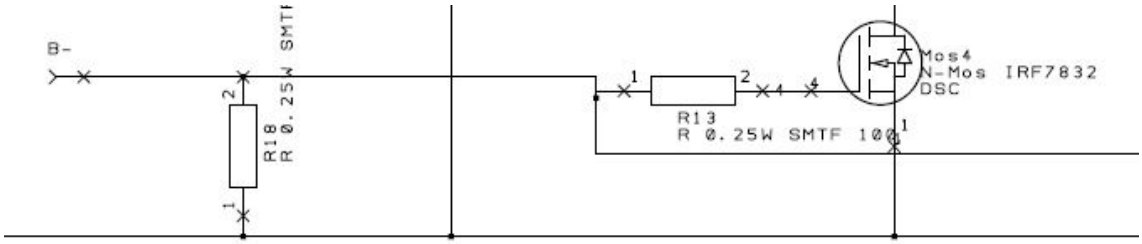

Figura 6.Circuito encendido transistor MOSFET canal N.

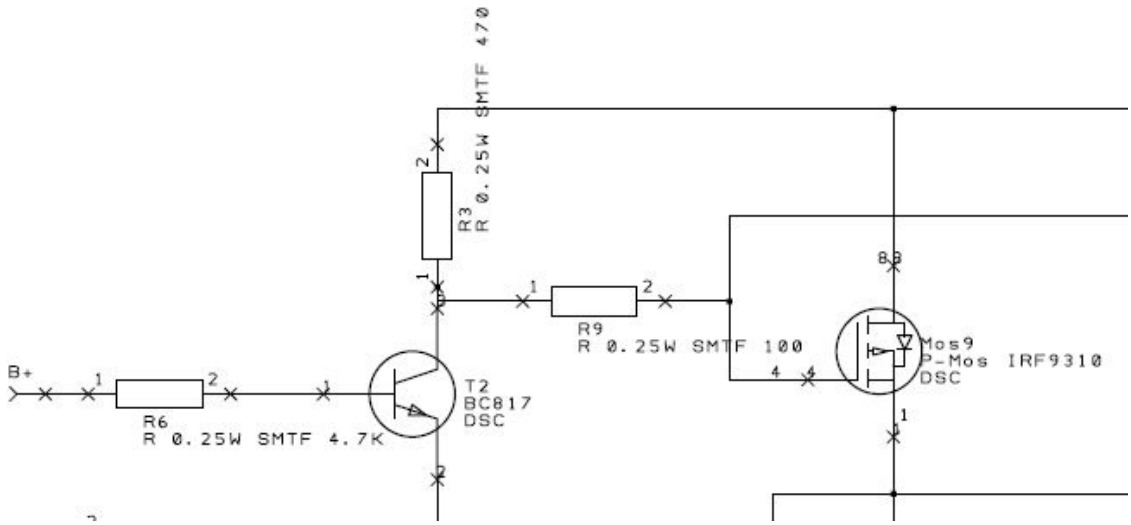

Figura 7. Circuito encendido transistor MOSFET canal P.

El motor puede llegar a consumir una corriente de 15 Amperios, por ello es importante que el dimensionamiento del puente en H sea el correcto para soportar la corriente que requiera el motor. Con transistores capaces de soportar entre 3 y 5 Amperios por encima del valor máximo del motor el circuito se encontrará bien dimensionado.

3. Micro controlador

Para iniciar el proyecto se ha tomado como referencia un circuito de control de un motor brushless comercial. En este caso se sustituye el micro controlador del modelo comercial por uno de la familia PIC de Microchip. Haciendo hincapié en las

características del comercial, se obtiene que el micro controlador PIC debe de poseer como mínimo estas características:

- Microprocesador de 8 bits.
- 16 MIPS, Mega Instrucciones Por Segundo.
- 8K Bytes de memoria flash programable.
- 512 Bytes de memoria EEPROM.
- 3 PWM.
- Timers de 8 y 16 bits.
- Conversores ADC de 10 bits.
- Alimentación a 5V.

El micro controlador que se ha escogido es el PIC 18F25K80 que en relación a las características anteriormente citadas tiene:

- Microprocesador de 8 bits.
- 16 MIPS, Mega Instrucciones Por Segundo.
- 64K Bytes de memoria flash programable.
- 1024 Bytes de memoria EEPROM.
- 5 PWM.
- 1 Timer 8/16 bits, 2 Timer 8 bits y 2 Timer 16 bits.
- Conversores ADC de 12 bits.
- Alimentación 5V.

Cabe destacar que la tecnología del micro controlador comercial y el PIC son diferentes de tal forma que en el micro controlador comercial realiza 1 instrucción por cada ciclo de reloj mientras que el PIC realiza 1 instrucción por cada 4 ciclos de reloj. Esto provoca que para obtener el mismo número de instrucciones por segundo, el PIC precisa de un reloj externo de 64MHz mientras que el micro controlador, con un reloj de 16MHz es suficiente.

4. Batería

Como el motor va instalado en una aeronave no tripulada, la alimentación de los circuitos electrónicos y los motores viene suministrada por una batería. Se supondrá que la batería debe alimentar a la aeronave no tripulada con 4 motores brushless.

Las baterías que se emplean en este tipo de aplicaciones son de Ion-Litio y polímero, también llamadas LI-PO. Las baterías se clasifican en función de tres parámetros, la tensión, la capacidad de descarga y la capacidad de la batería en miliamperios por hora (mA/h).

4.1.Tensión

Estas baterías se dividen en celdas, cada una con una tensión nominal de 3.7V. Para lograr baterías con mayor tensión, se colocan estas celdas en paralelo. En este caso la batería que se emplea tiene 3 celdas colocadas en paralelo obteniendo así una tensión nominal de 11.1V. La denominación de esta batería será 3s.

4.2.Capacidad de descarga

Una batería es capaz de entregar una mayor o menor intensidad en función de su capacidad de descarga. Un mayor índice de descarga proporciona picos de corriente más altos pero agota antes su carga, a la inversa, un menor índice de descarga proporciona una corriente más pequeña pero tarda más en descargarse. En este caso se trata de una batería 30C, para calcular la corriente máxima que puede dar la batería se multiplica ese coeficiente, 30C, a los miliAmperios de la batería. Como en este caso la batería es 30C y 2800mA/h, se obtiene una corriente máxima de:

Imax=30x2.8=84 A

Esos 84A que puede dar la batería son inferiores a la corriente máxima que pueden demandar los motores (4x15=60A) y los circuitos de control (1A a 2A) por lo que la batería se encuentra bien dimensionada.

4.3.Capacidad de la batería, mA/h

Es la capacidad de carga que posee la batería. En este caso, es de 2800mA/h, es decir, proporciona una corriente de 2800mA en 1h. El cálculo del tiempo estimado se realiza así:

t(min)=(mA Batería)/(mA Circuito) x60min

Realizando el cálculo, la duración de la batería es mucho menor siendo la corriente del circuito de 61A en potencia máxima y de media aproximadamente 40A. Esto supondría un tiempo de vuelo de 2.75 min a máxima potencia y a potencia media de 4min.

4.4.Precauciones

La descarga máxima de la batería no puede bajar nunca de los 3.3V por celda ya que eso supondría la destrucción de la misma.

5. Diseño I

Una vez conocidos los elementos fundamentales a la hora de realizar el proyecto, se procede a hacer un primer diseño.

5.1.Esquemático

Para la creación del circuito de control del motor brushless, inicialmente se toma como modelo un esquemático recogido en internet, Figura 8. Esquemático de un circuito comercial., que representa un modelo comercial.

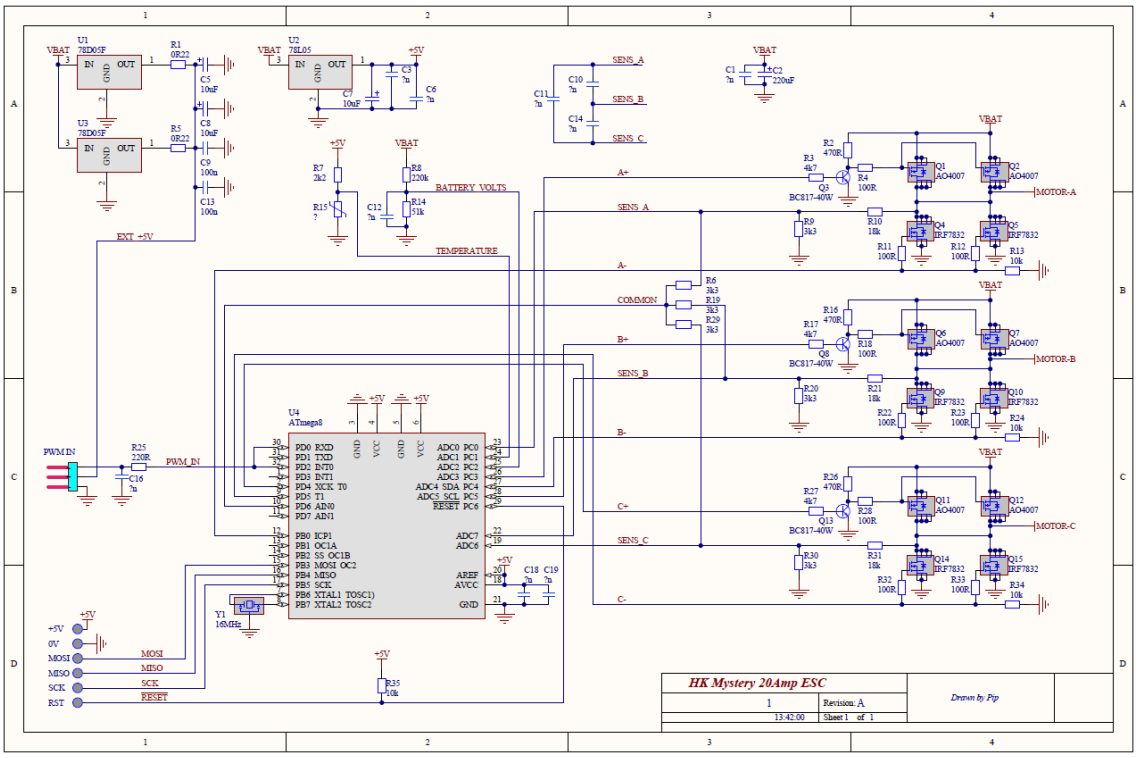

Figura 8. Esquemático de un circuito comercial.

En este proyecto se ha optado por sustituir el micro procesador del esquemático por otro con características similares pero de la familia PIC de Microchip. El esquemático realizado corresponde a la Figura 9.

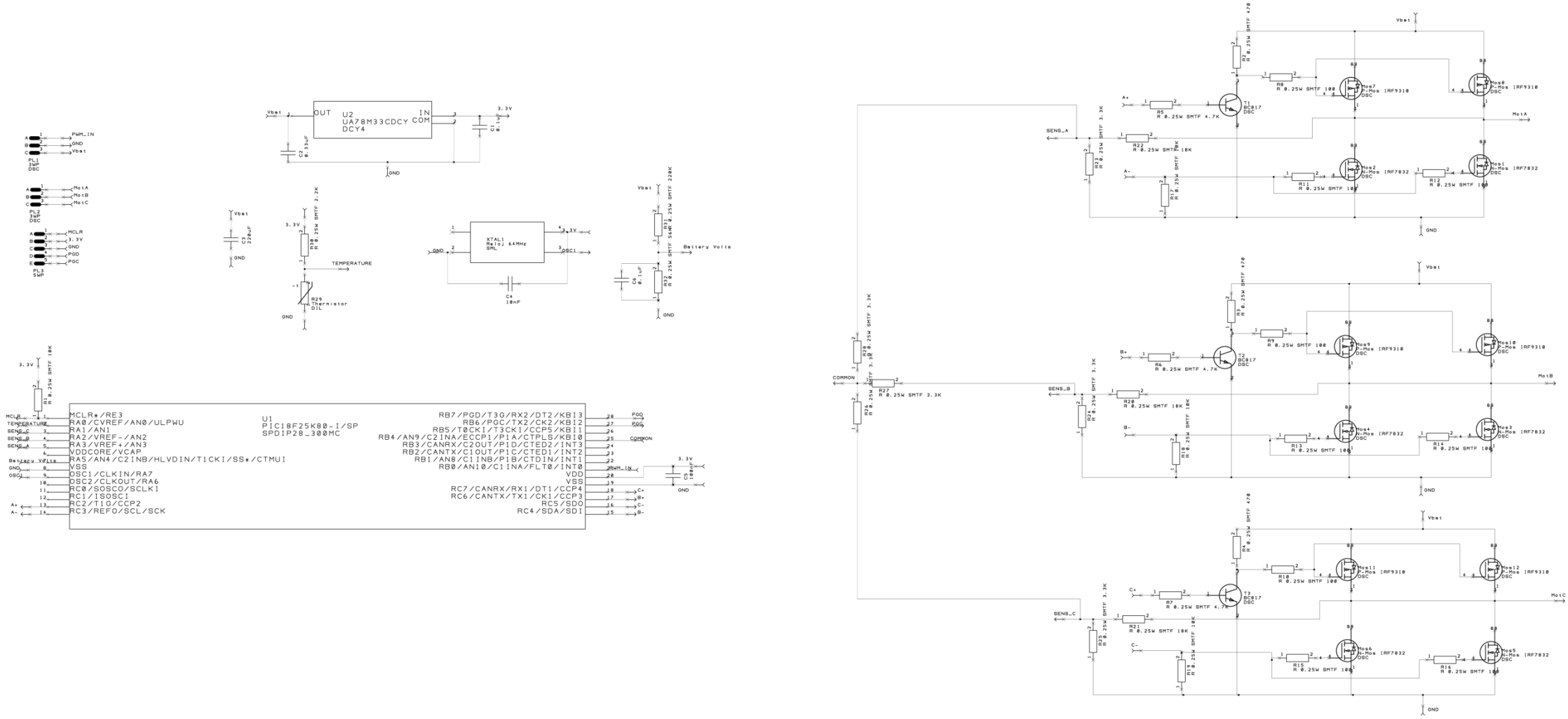

Figura 9. Esquemático del circuito del proyecto.

5.2.Circuito impreso

Una vez implementado el diseño del esquemático en el programa Design Spark, se procede a realizar el circuito impreso sobre el que irán montados los componentes.

Para las pistas de señal se ha tomado como valor nominal de anchura de la pista 0.7mm y para las pistas de potencia 1.3mm. El diseño se ha realizado dividiendo la tarjeta en dos secciones, la correspondiente a potencia (transistores, regulador lineal) y otra para la señal (micro controlador, reloj). En la sección de potencia, el plano superior corresponde a la alimentación y el inferior a masa. En la sección de señal, el plano superior corresponde a la alimentación de 3.3V o 5V y la superior a masa. Cabe destacar que las masas de ambas secciones se encuentran unidas en un solo tramo. Esto se realiza para poder disminuir el ruido de las conmutaciones de los transistores en el plano de masa y alimentación de los circuitos de señal.

Las dimensiones de la tarjeta de circuito impreso son de 50 mm de ancho por 70 mm de largo. Se ha buscado un diseño lo más compacto posible.

A continuación, las Figuras 10 y 11 representan la parte superior de la tarjeta y la inferior respectivamente.

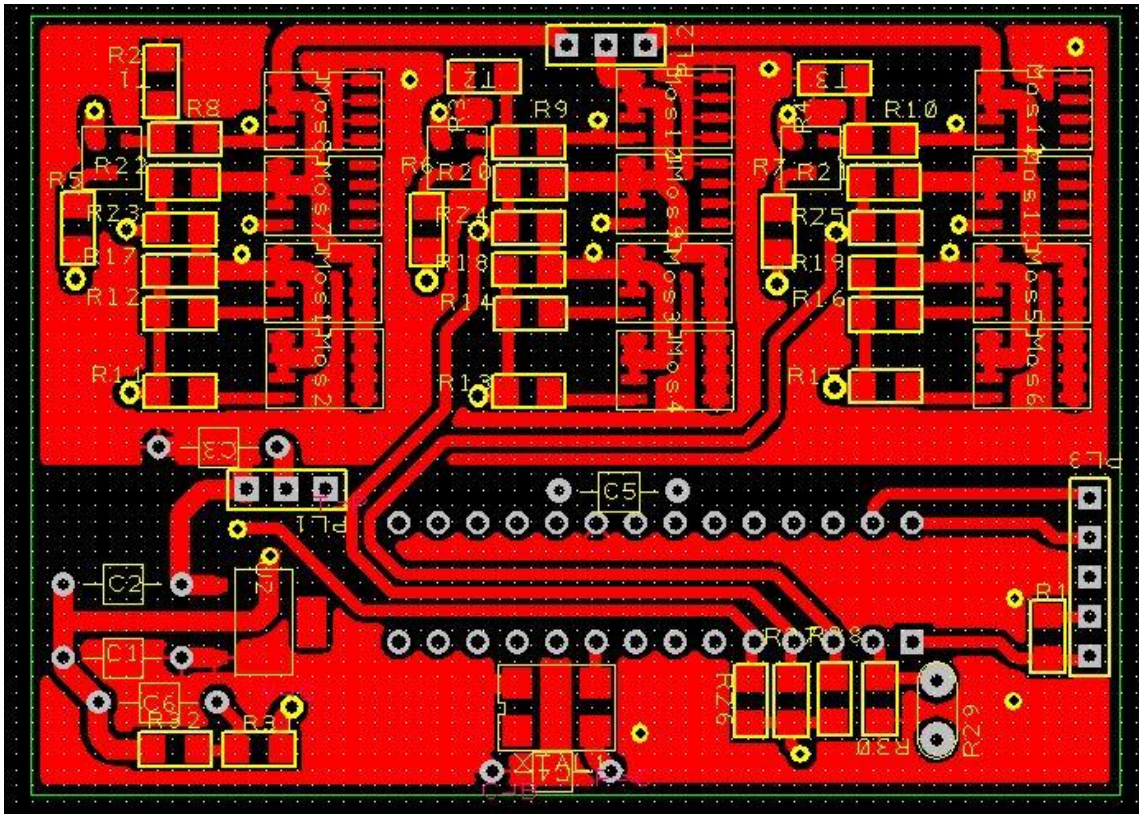

Figura 10. Imagen superior del circuito impreso.

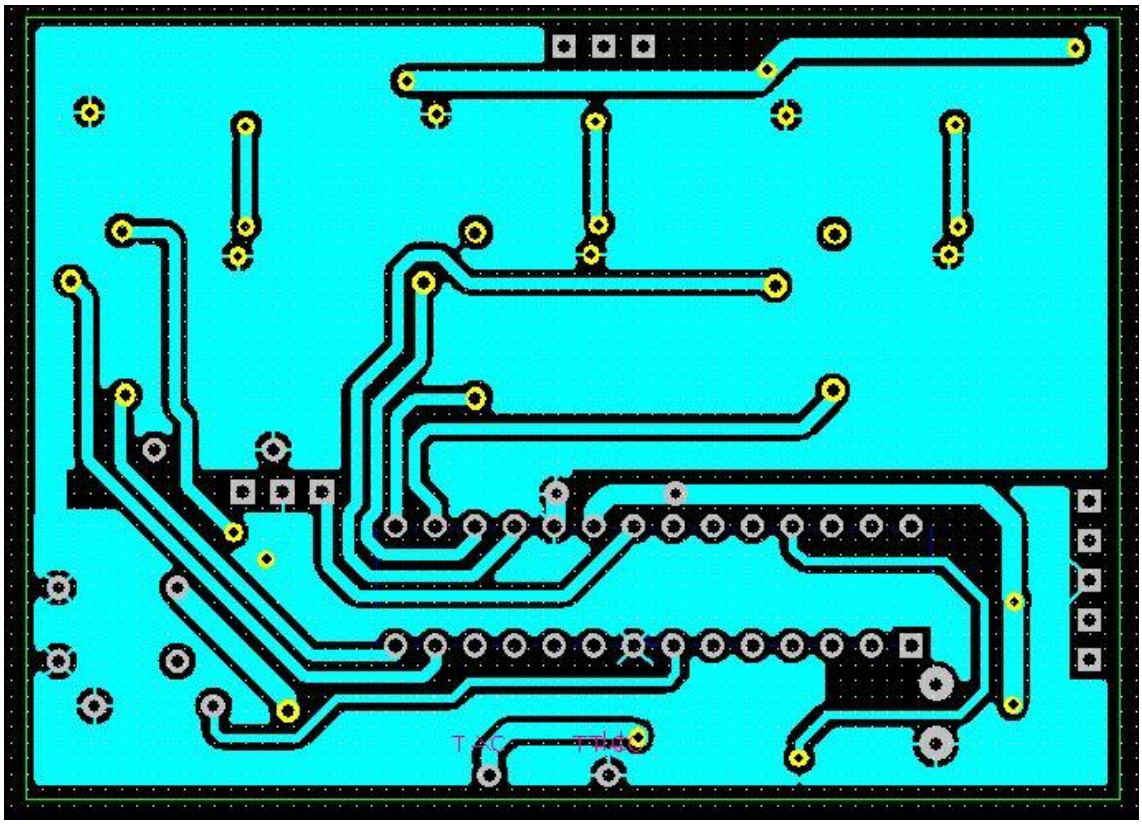

Figura 11. Imagen inferior del circuito impreso.

El software Design Spark proporciona también una imagen en tres dimensiones del circuito impreso para poder observar una imagen del acabado de la tarjeta más real. Estas imágenes se muestran en las Figuras 12 y 13.

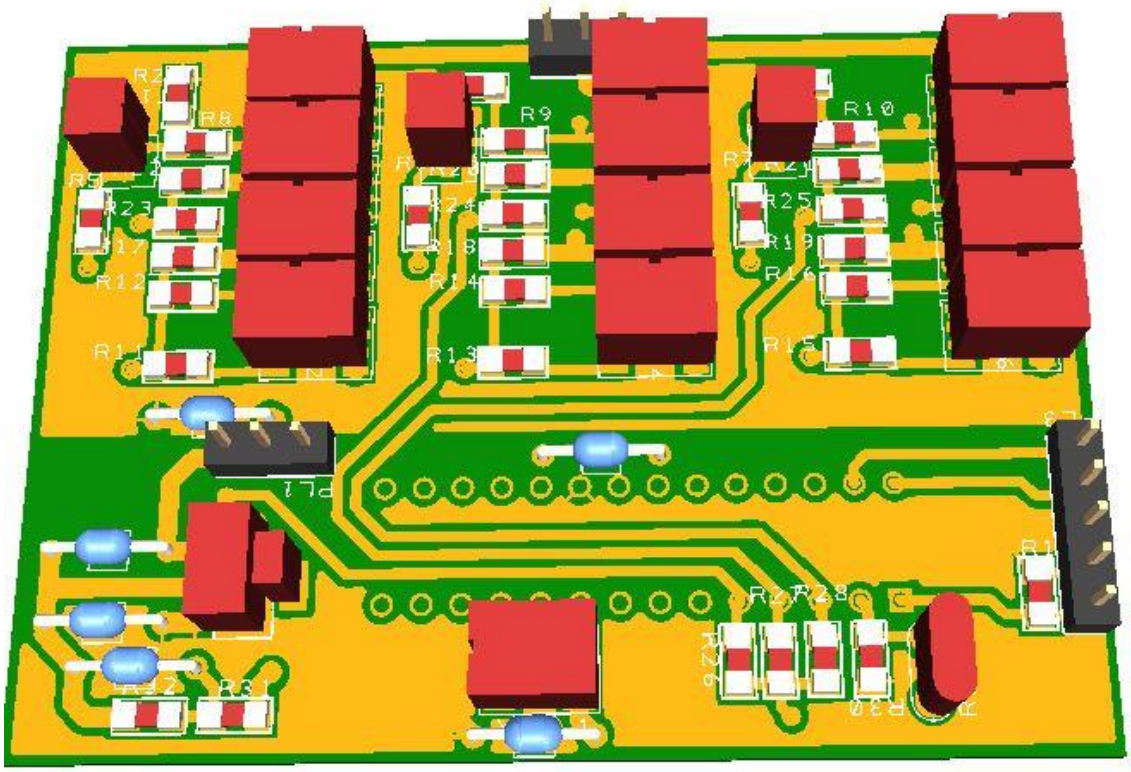

Figura 12. Circuito impreso 3D cara superior.

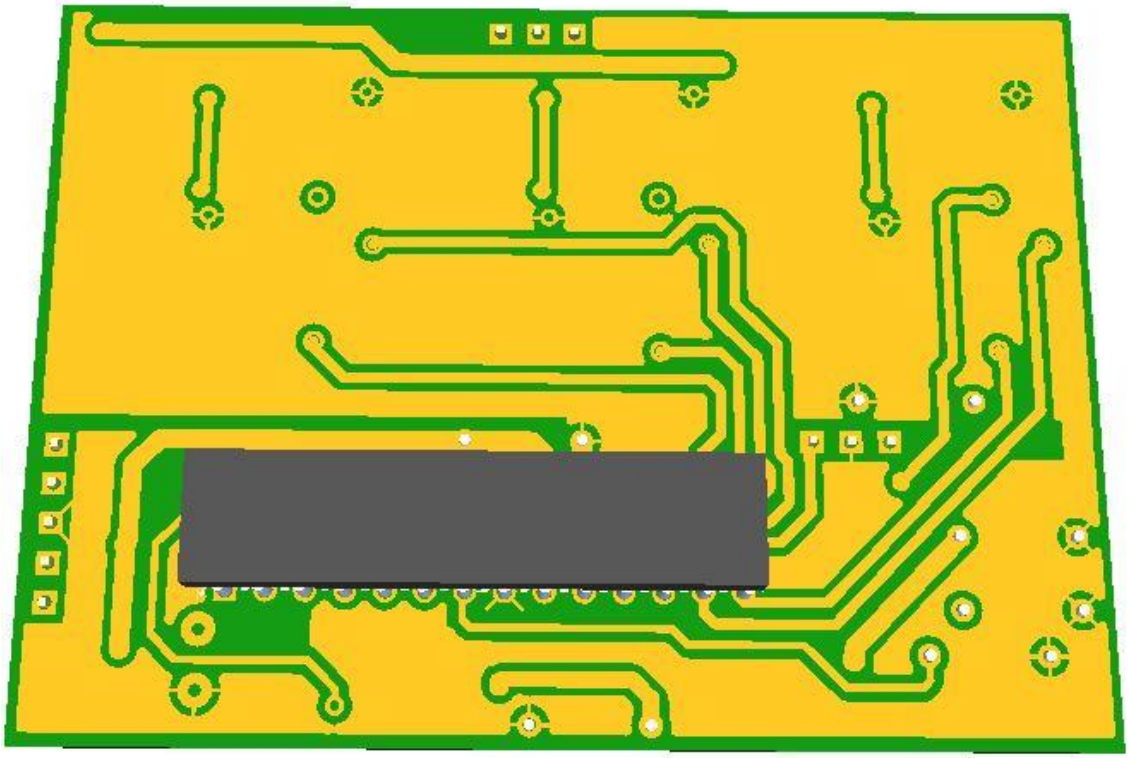

Figura 13. Circuito impreso 3D cara inferior.

5.3.Análisis

Una vez realizado el diseño y montaje inicial del circuito impreso que controlará el motor brushless, se prueba el circuito impreso sin conectar el motor para observar su funcionamiento.

Para poder comenzar a visualizar el comportamiento del puente en H se programa el micro procesador mediante el software MPLAB C18 para que envíe los pulsos de encendido y apagado a cada transistor a una frecuencia próxima a la máxima de funcionamiento del motor, 1KHz. De esta forma el control del puente en H es en lazo abierto.

Las primeras pruebas se realizaron con la alimentación del micro controlador a 3.3V para observar su funcionamiento. Debido a que la salida del micro controlador va conectada directamente a la puerta de los transistores MOSFET de potencia, en las primeras imágenes tomadas de las formas de onda de la señal a la entrada de los transistores, Figuras 14 y 15, destaca una curva en el flanco de bajada de cada fase. Esto se produce porque esa fase se encuentra flotante y hasta que no conduce el transistor inferior no disminuye su valor hasta 0V. Se ha observado también que el transistor MOSFET canal P necesita una corriente mayor para cargar la capacidad de la puerta y entrar en corte pero las resistencias entre la puerta y la alimentación limitan la corriente. Para agilizar las puertas de los transistores MOSFET y permitirles una conmutación más rápida, se disminuyen los valores de las resistencias que limitan la corriente en la puerta de los transistores de tal forma que circule una mayor corriente para cargar y descargar la capacidad de las puertas. En el caso de los transistores MOSFET canal N disminuir el valor de la resistencia de la puerta no ha supuesto un cambio relevante a la frecuencia deseada.

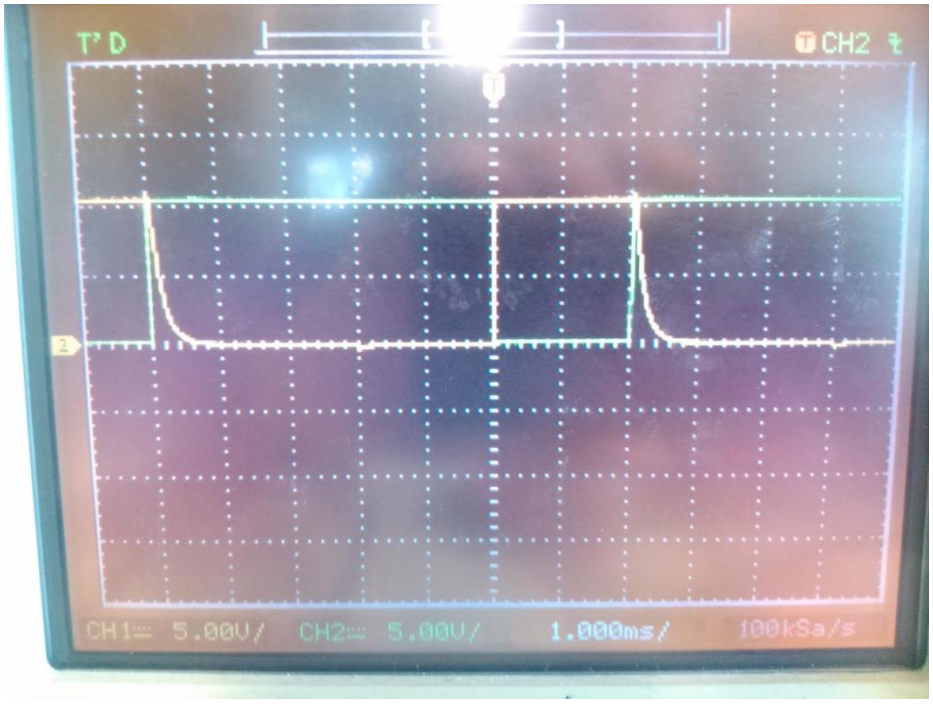

Figura 14. Tensión de una fase respecto a la tensión de la puerta de los MOSFET canal P.

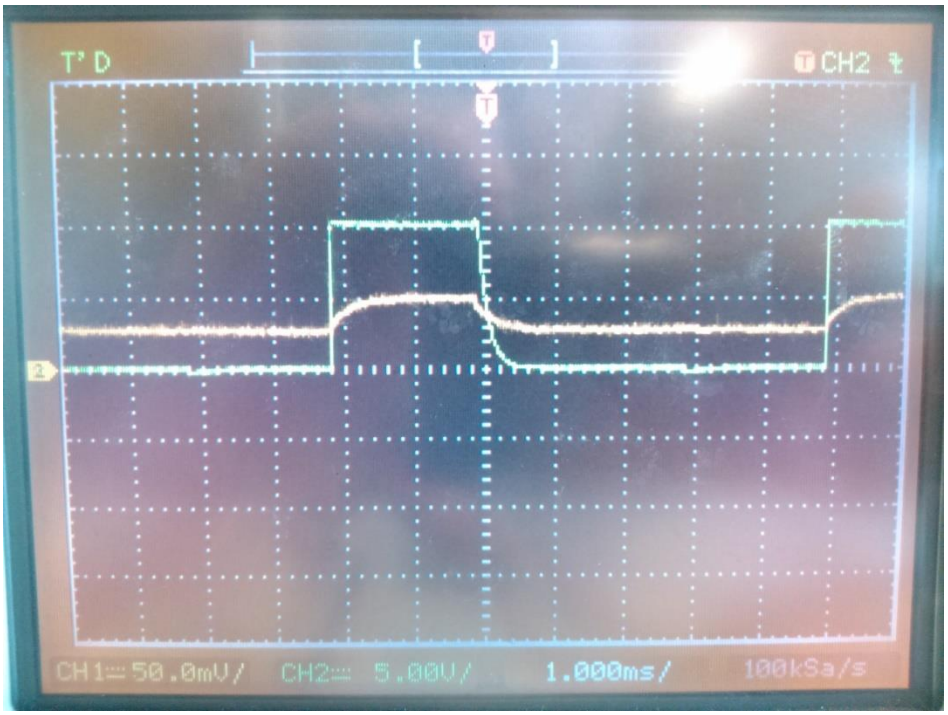

Figura 15. Tensión de una fase con respecto a la corriente total del circuito.

En la Figura 16 se puede visualizar que la corriente media consumida por el circuito se encuentra entre los 50 y 100mA conduciendo una sola fase. Parte del aumento de corriente en el circuito es debido a la conducción de los transistores MOSFET. La tensión umbral de las puertas de los transistores de canal N es de 2V aproximadamente, un valor muy próximo al de la tensión de salida del micro procesador, 3.3V. Para que los transistores puedan entrar en la zona de corte con un buen margen de seguridad se decide elevar la tensión de alimentación del micro procesador a 5V.

Alimentando el micro procesador a 5V los transistores MOSFET entran mejor en la región de saturación disminuyendo las pérdidas en el circuito.

En la Figura 16 donde aparecen dos fases consecutivas, se pueden observar claramente los pulsos desfasados 120º. La rampa que se produce es consecuencia de que la fase correspondiente se encuentra flotante y hasta que no conduce el MOSFET inferior que disminuya la tensión de la fase a 0V permanece con una tensión que va disminuyendo debido a las pérdidas.

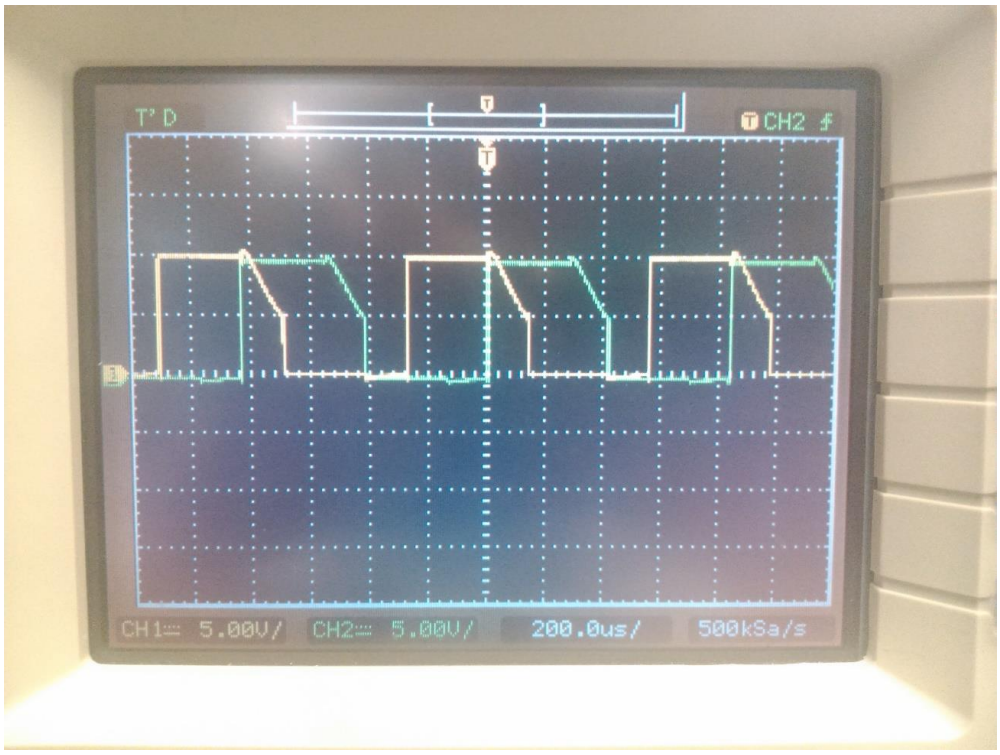

Figura 16. Fase A y B.

Las Figuras 17 y 18 muestran la fase con respecto a la tensión en la puerta del MOSFET canal P y canal N consecutivamente. Aunque la tensión en la puerta del MOSFET canal P sigue siendo la misma ya que se controla por medio de un transistor bipolar, la tensión de la puerta del MOSFET canal N es de 5V, la tensión de salida que proporciona el micro controlador.

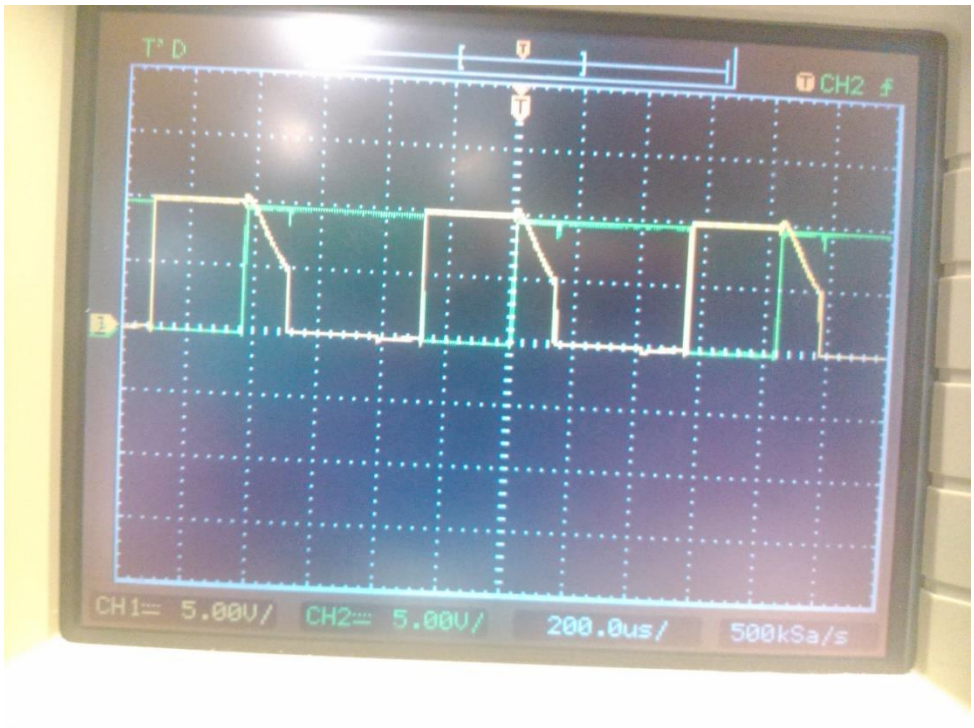

Figura 17. Tensión de una fase respecto a la tensión de la puerta de los MOSFET canal P.

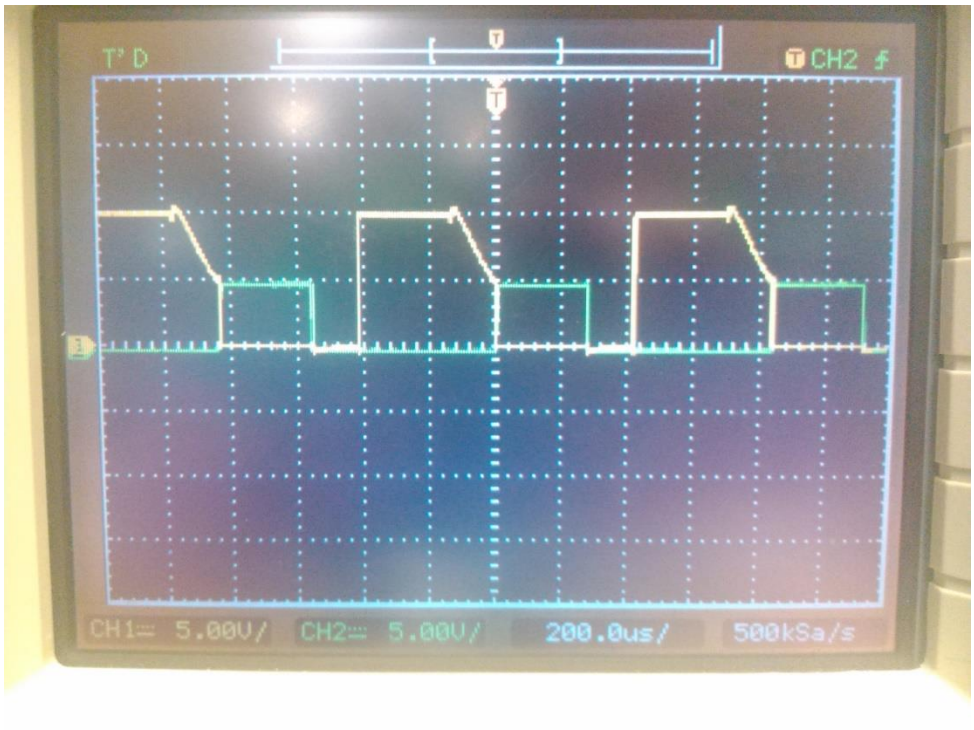

Figura 18. Tensión de una fase respecto a la tensión de la puerta de los MOSFET canal N.

La Figura 19 muestra una fase con respecto a la corriente total del circuito. Comparando esta imagen con la corriente consumida cuando la tensión de alimentación del micro controlador era de 3.3V, viendo únicamente el valor de la corriente, se puede interpretar que este resultado es negativo ya que la corriente media se encuentra entre los 150 y 200mA. Teniendo en cuenta que el valor obtenido de la corriente consumida con alimentación del micro controlador es de 100mA con una sola fase activa que consume 50mA y en el caso de la alimentación del micro controlador la corriente es de 150mA con las tres fases activas, la mejora es considerable justificando así el cambio de tensión de alimentación del micro controlador para que los transistores MOSFET se encuentren con seguridad en la región de saturación.

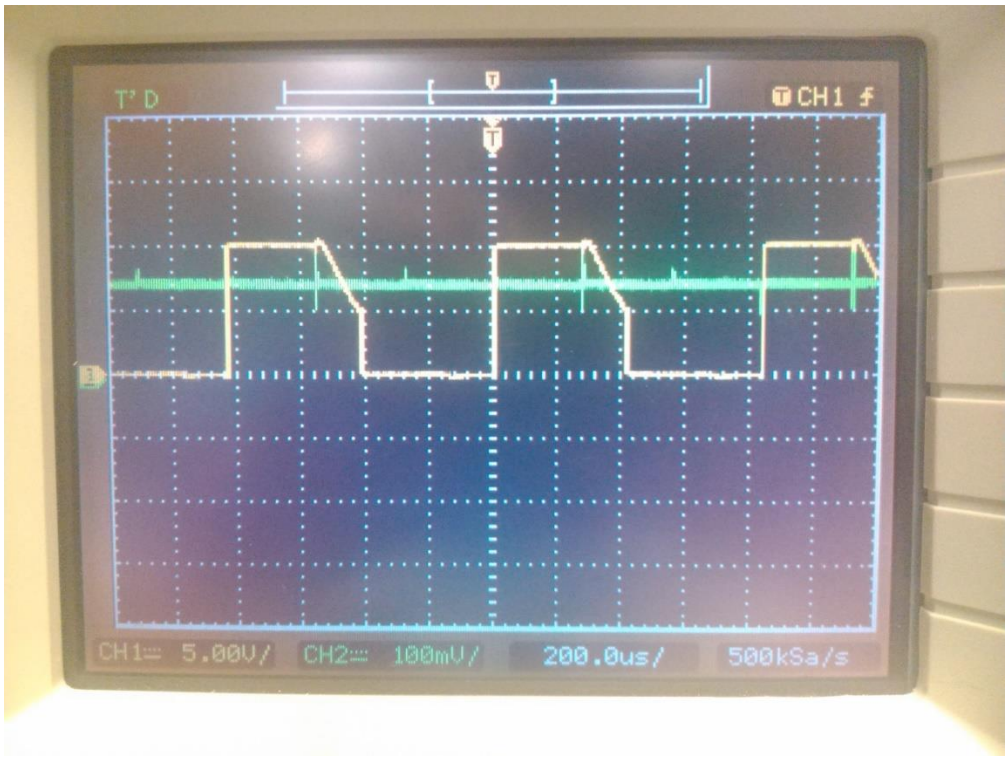

Figura 19. Tensión de una fase respecto a la corriente total del circuito.

Finalmente en la Figura 18 se aprecian picos de corriente en momentos determinados, esto es debido a que los transistores cuando conmutan de corte a saturación requieren de una corriente que cargue la capacidad de las puertas.

5.4.Conclusiones y mejoras

Mediante las pruebas se ha llegado a la conclusión de que la tensión de alimentación del micro controlador debe de ser de 5V debido a que favorece que los transistores MOSFET entren en la región de saturación disminuyendo las pérdidas.

Para agilizar las conmutaciones de los transistores se debe permitir que la corriente que necesitan para cargar las capacidades de las puertas sea mayor y eso se logra con resistencias de menor valor y dispositivos que permitan ese pico de corriente que precisan para conmutar.

Se presenta como mejora la inserción de drivers entre el micro procesador y los transistores MOSFET del puente en H ya que este circuito no tiene estos dispositivos preparados específicamente para mejorar la conmutación de los transistores MOSFET recibiendo las señales de encendido y apagado desde el micro controlador.

6. Diseño II

Esta segunda parte del diseño busca mejorar el circuito anterior de tal forma que el control del motor sea más eficiente y seguro.

En este diseño se implementan las mejoras que se han comentado en el diseño I, la inserción de drivers entre el puente en H y el micro controlador.

Los drivers escogidos tienen unas características que se ajustan al puente en H implementado ya que se alimentan con tensiones entre 5 y 18 V y la corriente que puede dar a los transistores MOSFET para su activación supera los 2A.

También se ha agregado un circuito que permita la conexión mediante RS-232 a un ordenador donde poder visualizar y controlar diversas variables del sistema.

Para poder conectar con un ordenador RS-232 el micro controlador se conecta mediante USART con el integrado MAX232 para que aumente la tensión de la transmisión desde los 5V de alimentación del micro controlador a 12V de transmisión RS-232.

6.1.Esquemático

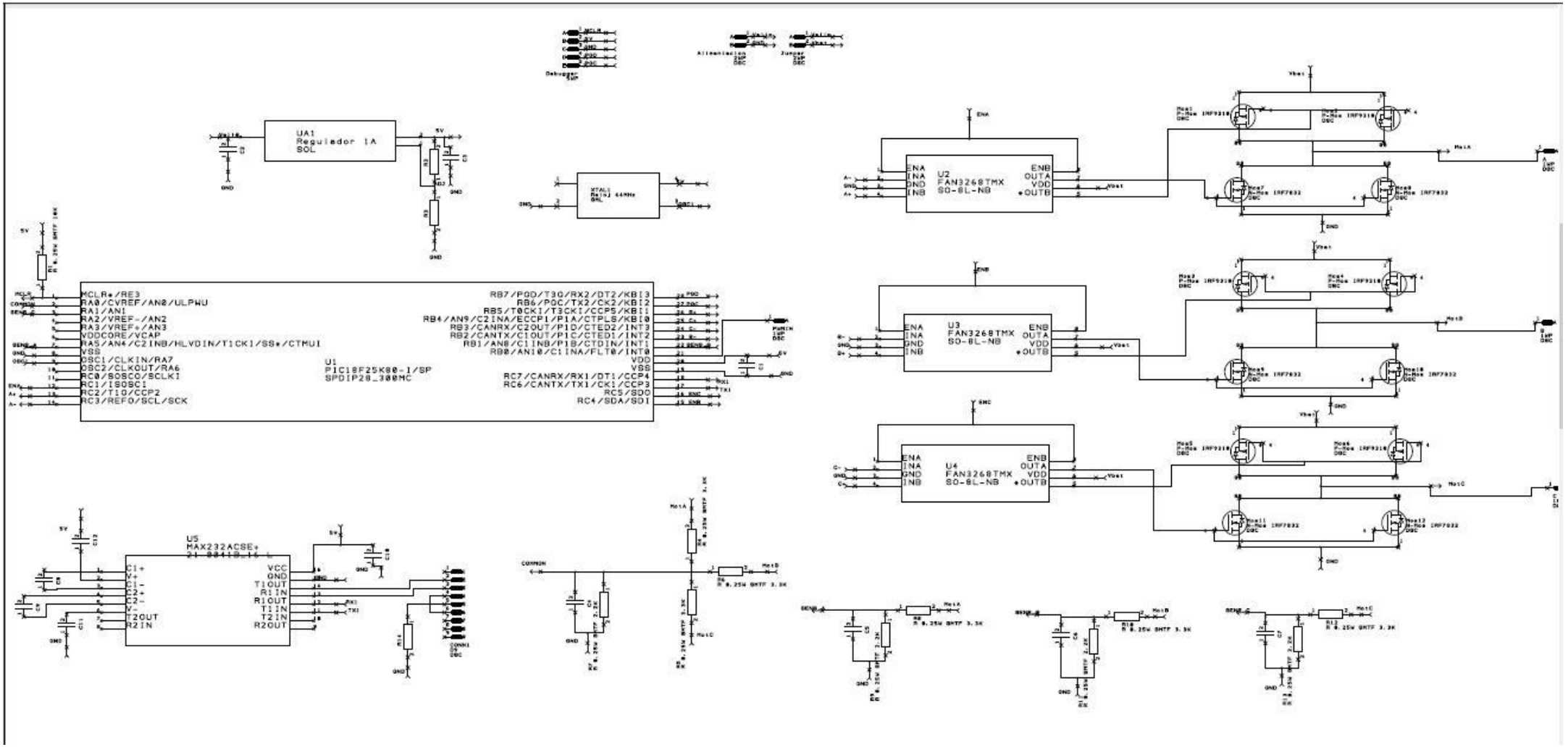

Figura 20. Esquemático del diseño II.

6.2.Circuito impreso

Para el diseño del circuito de la segunda versión, se han distribuido los componentes de las fases de diferente forma de tal forma que el camino de la corriente a cada fase del motor sea lo más corto posible.

Las dimensiones de la tarjeta de circuito impreso son de 60mm de ancho por 60mm de largo. En el caso de que el micro controlador y el integrado específico para la transmisión RS-232 fueran de montaje superficial, se podrían llegar a ajustar algo más las dimensiones de la tarjeta.

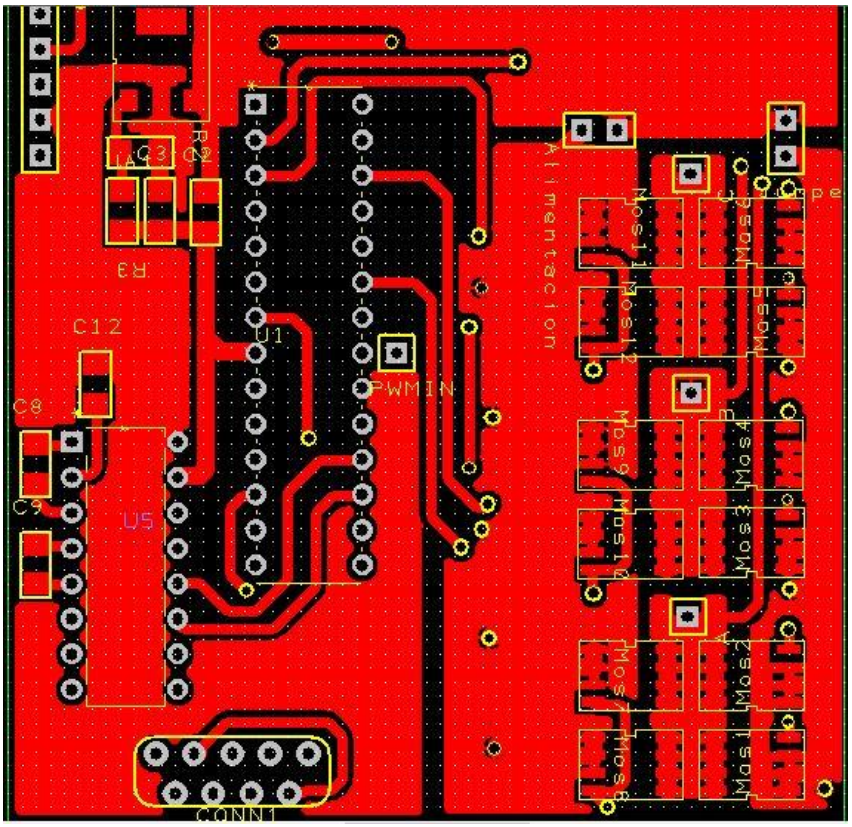

Figura 21. Imagen superior del circuito impreso.

Al igual que en el anterior diseño, el conector para programar el micro controlador no se introduce dentro de la tarjeta sino que se facilita el acceso a los pines requeridos y se acopla externamente el conector. A su vez, para el conector RS-232 se facilitan los pines requeridos pero con la forma del conector ya que el conector empleado encaja en la tarjeta directamente pero la carcasa no se encuentra dentro de la tarjeta sino hacia el exterior.

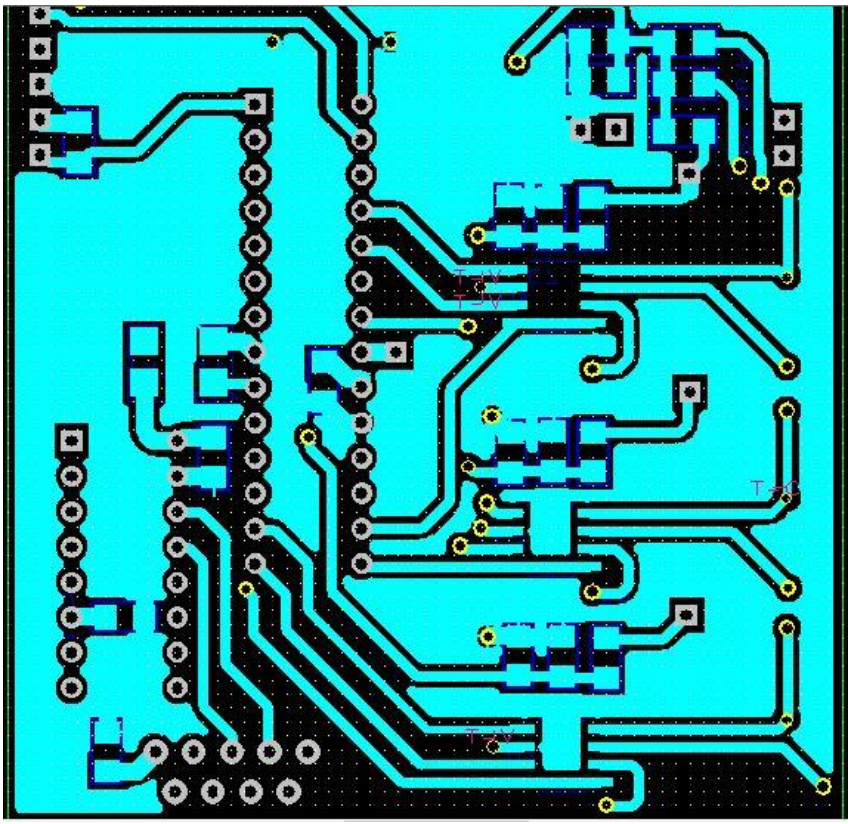

Figura 22. Imagen inferior del circuito impreso.

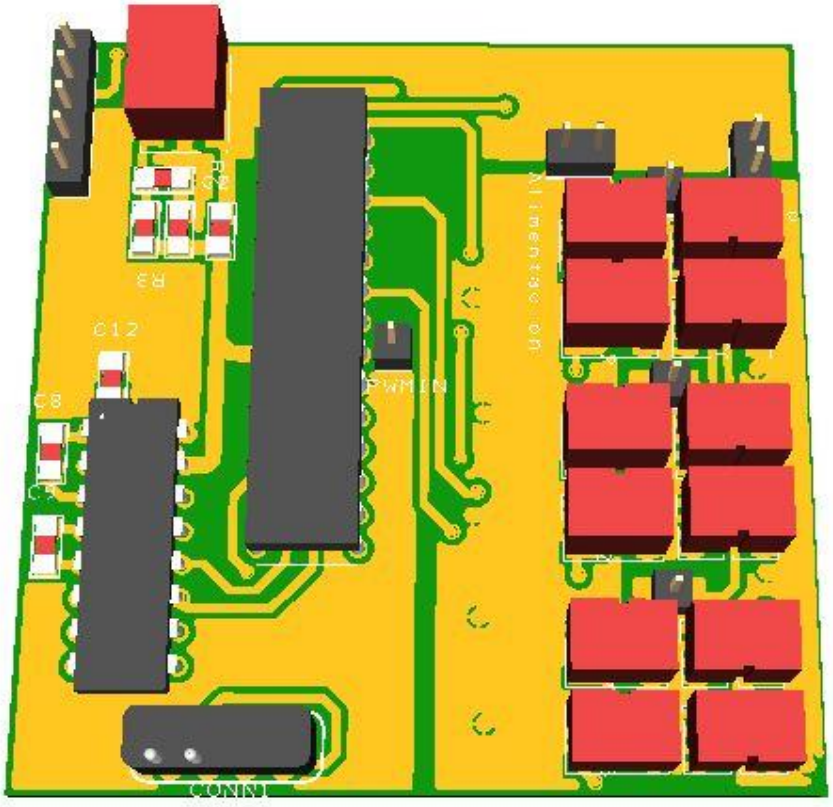

Figura 23. Circuito impreso en 3D cara superior.

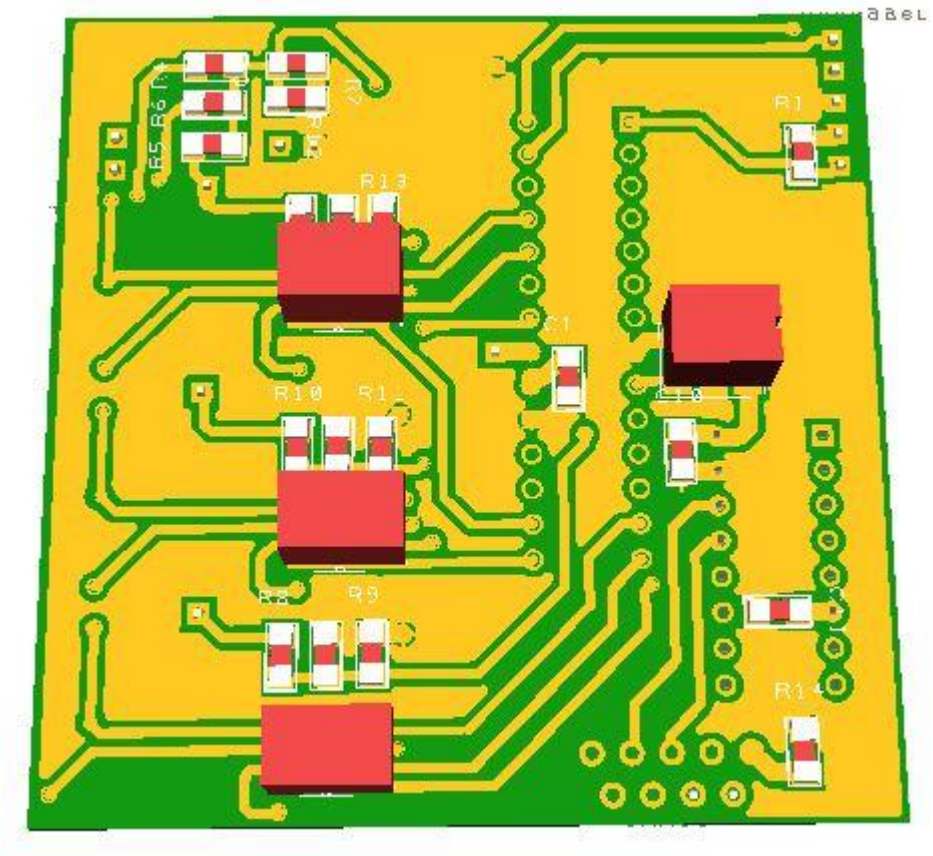

Figura 24. Circuito impreso en 3D cara inferior.

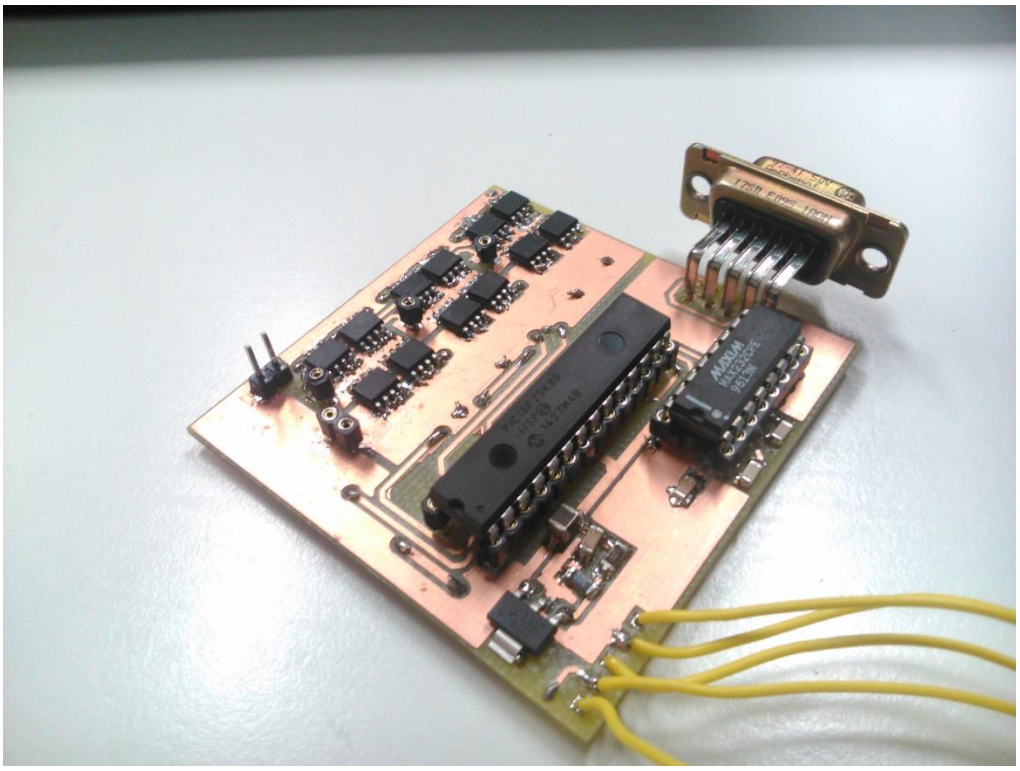

Figura 25. Imagen real del circuito impreso ya montado.

6.3.Análisis

En este diseño se logran las mejoras previstas con respecto al primer diseño con la introducción de los drivers.

La corriente que consume el circuito cuando e encuentra en reposo es de 50mA y cuando conduce sin el motor acoplado sube a 60mA. Una mejora notable ya que en el anterior diseño cuando conducían los transistores sin el motor acoplado la corriente requerida por el circuito era de 150mA.

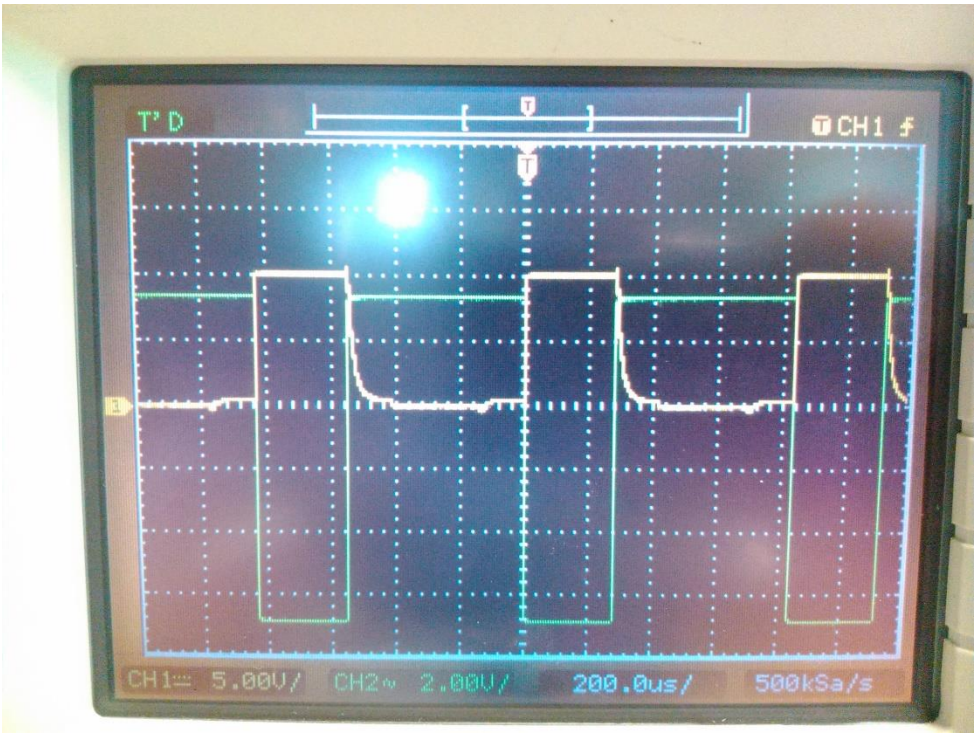

Figura 26. Tensión de una fase con respecto la tensión de puerta de un transistor MOSFET canal P.

En las Figuras 26 y 27 se muestran las formas de onda de una fase y las tensiones en las puertas de los transistores MOSFET de canal P y de canal N respectivamente. En este caso se aprecia que ambas tensiones en las puertas tienen un rango de 0 a 10V, esto es debido a los drivers, circuitos integrados encargados de saturar y cortar los MOSFET. Por ello la conmutación de los transistores es mejor y producen menos pérdidas disminuyendo la corriente que el circuito requiere.

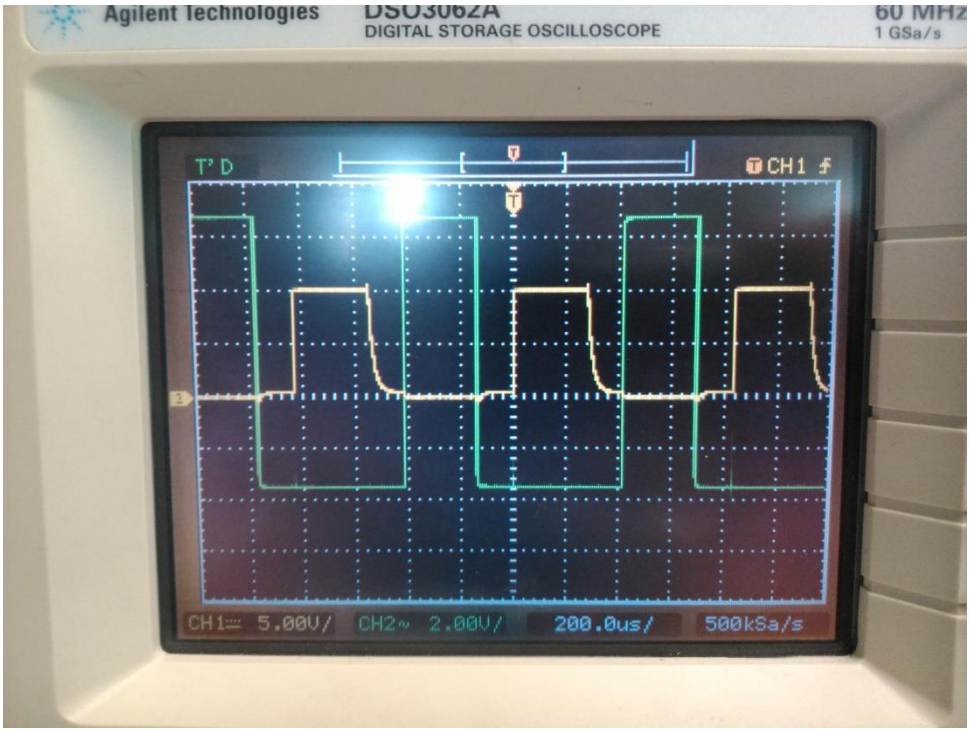

Figura 27. Tensión de una fase con respecto a la tensión de puerta de un transistor MOSFET canal N.

Para evitar sobretensiones y sobrecorrientes que se puedan originar se han colocado condensadores de desacoplo de 100nF en los drivers y condensadores de 220uF en las entradas de las fuentes de alimentación. Estas sobrecorrientes y sobretensiones se originan cuando se producen conmutaciones a alta frecuencia ya que pueden crear en las pistas del circuito impreso inductancias que provoquen picos de corriente y estos puedan dañar los integrados.

Como el diseño ha arrojado resultados favorables para poder poner en funcionamiento el motor sin dañar el circuito o el motor, se ha alimentado el motor en lazo abierto y se han obtenido las siguientes imágenes que representan las formas de onda del motor en el puente en H. La Figura 28 muestra una fase del motor y la Figura 29 dos fases consecutivas.

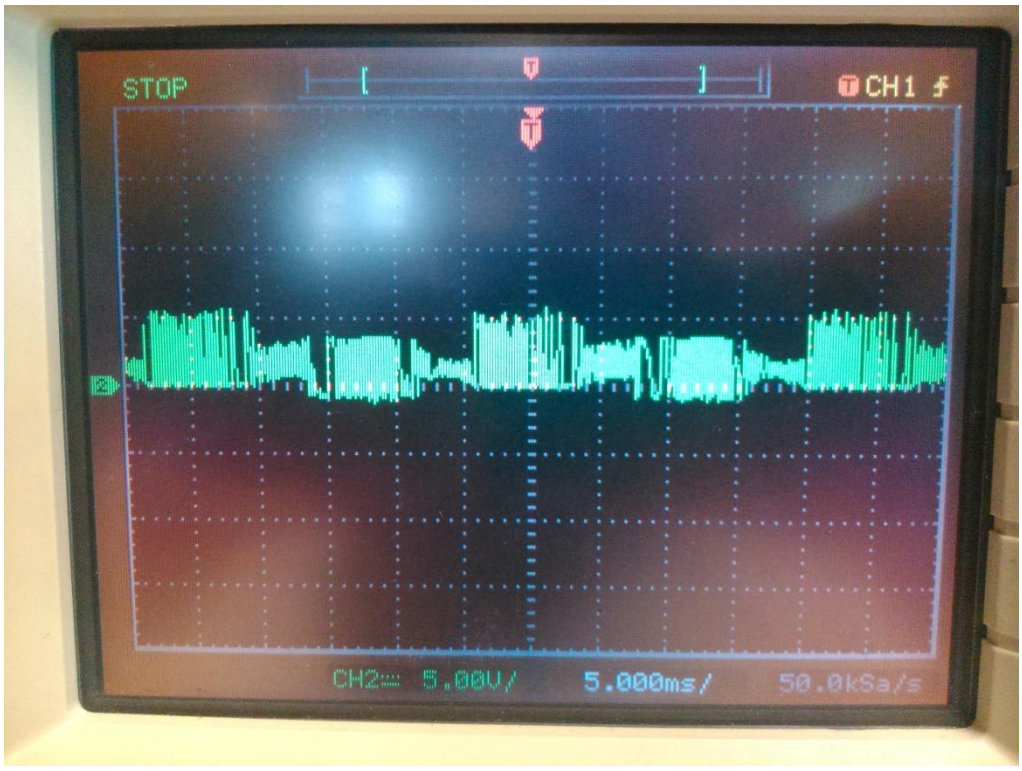

Figura 28. Fase A del motor en el arranque.

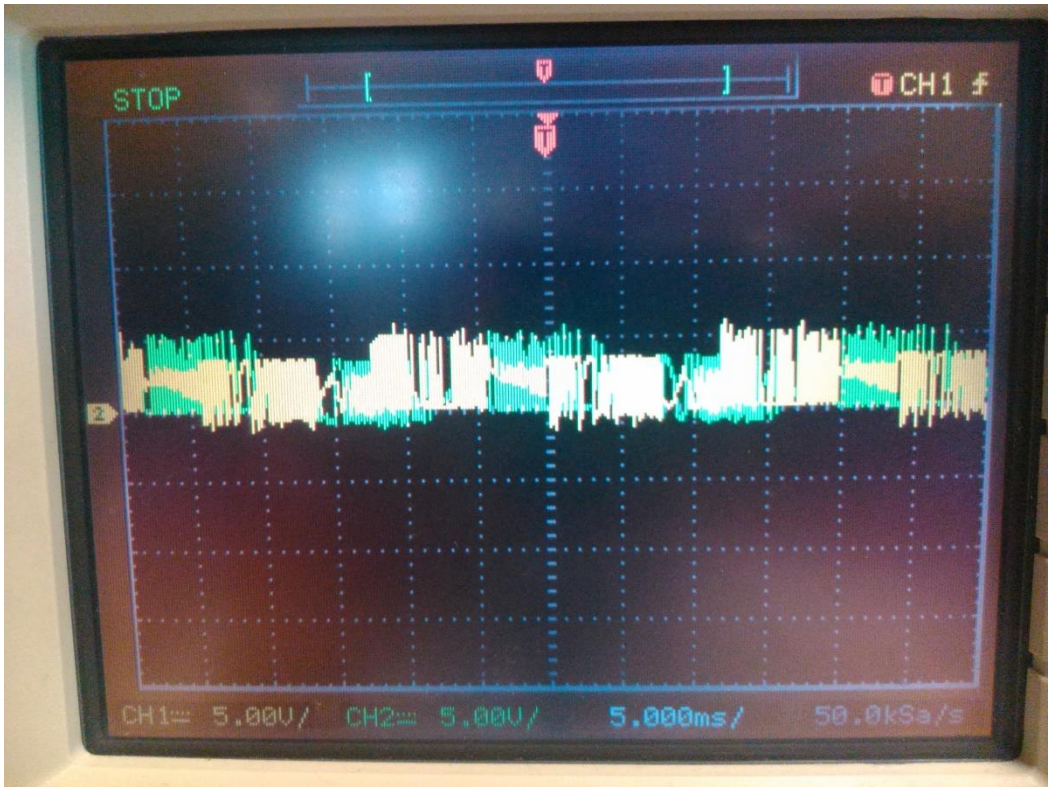

Figura 29. Dos fases del motor en el arranque.

Como se ha limitado la corriente la tensión a la que se alimenta el motor es inferior a los 10V iniciales, por ello las formas de onda de las imágenes tienen una amplitud de unos 5V. La corriente que absorbía el motor era de 1.5A.

El rizado que se muestra corresponde al paso de los imanes permanentes por la fase variando su tensión y se puede observar claramente cuándo la bobina que le corresponde a la fase está alimentada en alto, en bajo o flotante. Cuando se encuentra flotante se observa una rampa, esta rampa tiene una dimensión de 30º mecánicos y a mitad de ella es donde se produce la conversión analógico-digital para controlar la velocidad del motor en lazo cerrado.

6.4.Conclusiones y mejoras

En este diseño se ha requerido de un radiador para los drivers a parte del radiador para los transistores ya que las corrientes con las que trabaja el motor son elevadas y los componentes que forman el puente en H son muy pequeños como para disipar ellos mismos el calor generado por las pérdidas.

Se ha logrado un diseño más eficiente y mejor preparado para el control del motor con respecto al primer diseño.

Como mejora se puede incluir un pequeño circuito para obtener el valor de la tensión de la fuente de alimentación de cara a controlar la tensión de la batería por seguridad.

7. Control motor brushless sensorless

Los motores brushless sensorless se les puede controlar su velocidad y par ejercido en función de la frecuencia de los pulsos con los que se alimentan las fases y la corriente que absorben.

Este proyecto se basa en el control de un motor brushless específico para aeronaves no tripuladas el cual no precisa de algoritmos complejos ya que no se busca controlar con precisión el motor. Por ello la principal variable a controlar va a ser la velocidad y con menor precisión la corriente.

Se distinguen dos tipos de control para el motor, el lazo abierto donde el micro controlador envía las señales apropiadas para que el motor funcione y el lazo cerrado donde implementa la misma operación que en lazo abierto pero se le añade un algoritmo de realimentación para controlar la velocidad de una forma más precisa.

7.1.Lazo abierto

El control en lazo abierto radica en generar las formas de ondas correctas para que el motor arranque y alcance la velocidad fijada por software. Es un control donde el mico controlador envía las acciones que desea que el motor realice pero no hay forma de conocer si verdaderamente el motor se encuentra girando y a la velocidad prevista. La forma de onda teórica de cada fase es la de la Figura 30 y en la Figura 31 aparecen las formas de onda reales de dos fases.

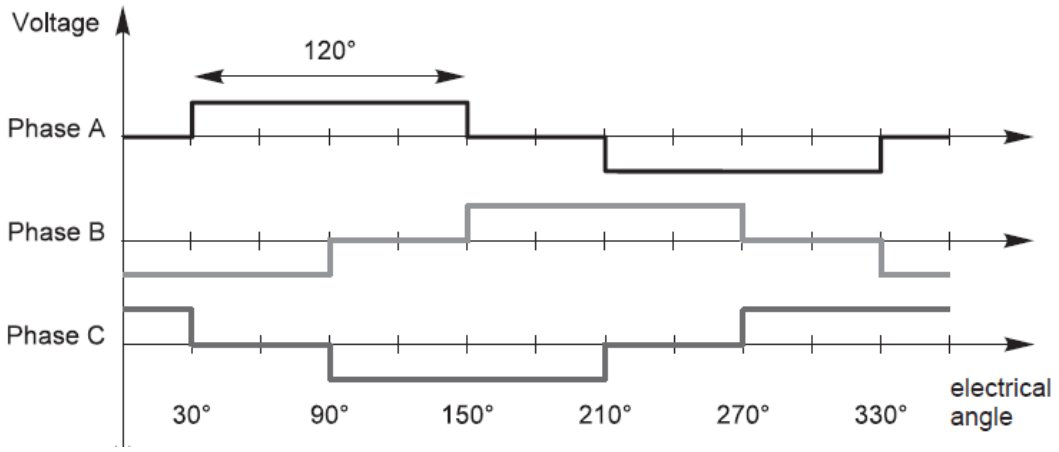

Figura 30. Pulsos eléctricos en cada fase durante una vuelta.

Figura 31. Pulsos reales obtenidos entre dos fases del puente en H.

Para que el motor alcance la velocidad prevista, partiendo del reposo, el micro controlador comienza a enviar los pulsos a una baja frecuencia eléctrica de tal forma que se cree un campo magnético en el estator que permita al rotor girar y que el campo magnético del rotor siga al del estator. Gradualmente se aumenta la frecuencia para que el rotor gire a mayor velocidad hasta que alcance la velocidad fijada. Si se desea variar la velocidad con el motor ya girando, se deberá hacer también de forma gradual ya que si se produce un salto brusco en la frecuencia que alimenta al motor, puede originar una pérdida de sincronismo en los campos magnéticos y que se produzcan pares contrarios que originen vibraciones o la parada del motor.

Para lograrlo se programa el micro controlador de tal forma que con unos vectores que contienen los datos necesarios para generar las activaciones correspondientes de los transistores MOSFET generen las formas de onda de la Figura 30. Para regular qué elemento del vector debe transmitir el micro controlador a una frecuencia determinada, se emplea el Timer 1. Para acelerar el motor, ir aumentando gradualmente la frecuencia de los pulsos, se emplea el Timer 0 donde incrementa el valor de la variable “frec” que se cargará en el Timer 1 controlando así la frecuencia de los pulsos que se transmiten y por ello la velocidad del motor.

Una vez que se controla la velocidad del motor, se puede controlar el par que ejerce el motor por medio de un control de corriente.

Para controlar la corriente se introduce una PWM, modulación por anchura de pulso, cuando la fase correspondiente se conecta a la alimentación. De esta forma, controlando la anchura de pulso se controla la tensión media que se alimenta el motor y por ello la corriente que absorbe. Como es un control en lazo abierto, no se puede conocer la corriente real que el motor absorbe.

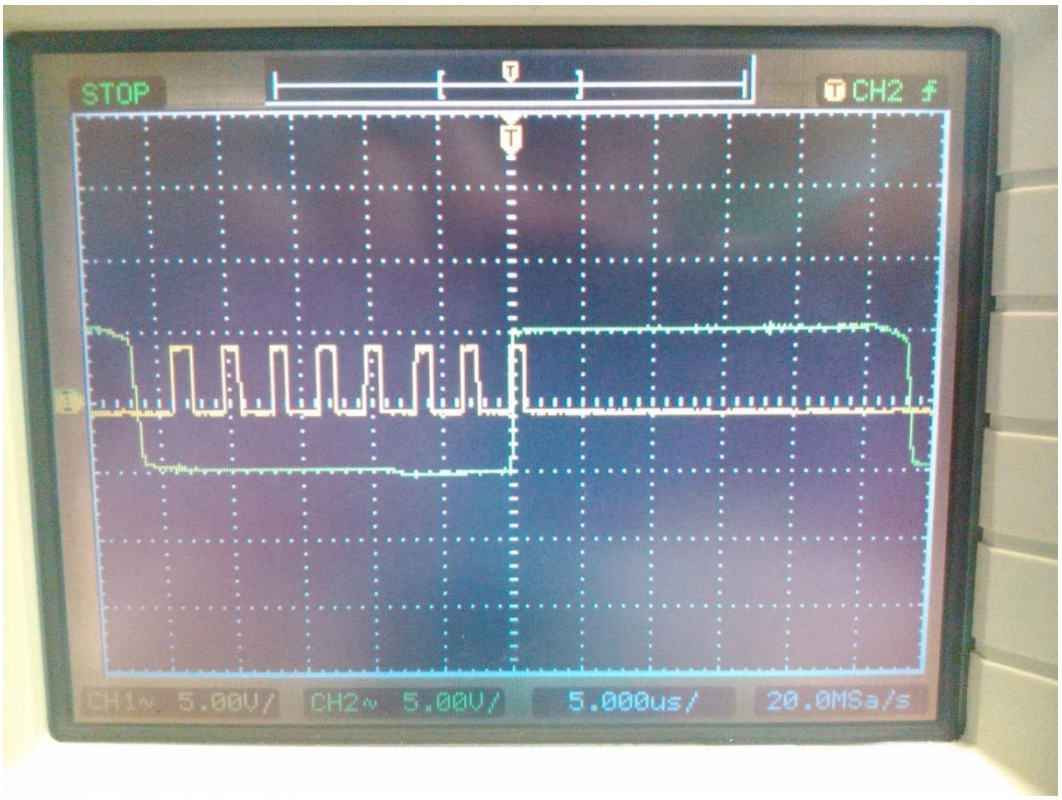

Figura 32. Forma de onda con PWM integrado con respecto a una forma de onda sin PWM.

En la Figura 32 aparecen las formas de onda de dos fases consecutivas. A la primera fase que se muestra en amarillo se le ha integrado la modulación por anchura de pulso, PWM, para controlar la tensión media que se alimenta el motor. En la segunda fase de color verde no se le ha aplicado ninguna técnica y como se puede apreciar, en todo el intervalo que se conecta la fase a la alimentación, la fase recibe de media el valor máximo de la alimentación.

Como el ciclo de trabajo de la PWM recoge un valor desde el 0% donde no hay tensión hasta el 100% donde la tensión media es la máxima y la tensión media con la que se alimenta el motor es directamente proporcional a la corriente, para variar la corriente que absorbe el motor se variará el ciclo de trabajo con un valor entre el 0 y 100% siendo el 100% la máxima corriente que se entrega al motor y 0% parada del motor.

Al micro controlador se le añaden unas líneas de código donde en función de si se encuentra en la parte alta de la forma de onda que se transmite o no, active la PWM correspondiente con el ciclo de trabajo que se desee.

7.2.Lazo cerrado

El control en lazo cerrado del motor reside en enviar los pulsos al igual que en el lazo abierto pero esta vez se realiza un proceso para determinar si el motor gira a la velocidad deseada o hay una desviación para corregirla.

Esta técnica se llama detección de la fuerza contra electromotriz, en inglés “Back Electromotive Force Detection”. Consiste en recoger en un punto determinado llamado punto de cruce por cero, en inglés “zero cross point”, la tensión de la fase que se encuentra flotante para compararla con la tensión en el neutro ficticio del motor y así determinar si se encuentran alineadas la frecuencia del pulso que se transmite con la velocidad del motor.

Para realizar esta detección, se emplea un divisor de tensión para disminuir la tensión que recoge el micro controlador a un nivel que pueda leer y mediante un conversor analógico digital recoger el valor para realizar la comparación por software. En ocasiones debido al ruido producido por la conmutación de los transistores es conveniente colocar un filtro para obtener una señal más limpia. En la Figura 33 se muestra un esquema visual del hardware empleado para la detección.

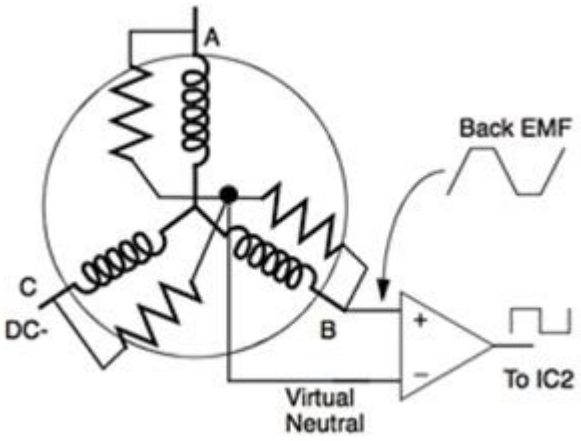

Figura 33. Obtención de la fuerza contra electromotriz.

La comparación se realiza con un neutro ficticio ya que no se precisa de un acceso al neutro real del motor. Se unen mediante tres resistencias las tres fases a un punto y se recoge de ese punto la tensión de la misma forma que si fuera una fase, con un divisor de tensión, un filtro y un conversor analógico digital. La tensión de este neutro ficticio suele ser de la mitad de la tensión a la que se alimenta el motor. Por ello en ocasiones por simplificar el hardware se compara la tensión de la fase flotante directamente con este valor pero puede provocar un mayor error.

El momento para realizar la conversión es cuando la fase que se desea medir se encuentra flotante y a 30º antes de que se activen los transistores de canal P que suban la tensión de la fase a la de alimentación. En ese momento se debe cumplir que la tensión de la fase es aproximadamente la tensión del neutro, la tensión de alimentación entre dos. Si el resultado de la comparación es mayor implica que la fase se encuentra atrasada y si es menor que se encuentra adelantada. En la Figura 34 se muestran con triángulos morados el momento donde se mide el paso por cero de la fuerza contra electromotriz de cada fase que se representa con una línea contínua. También con líneas discontínuas la corriente de cada fase que es positiva cuando la parte superior del puente

en H se encuentra activa y negativa cuando la parte inferior del puente se encuentra activa.

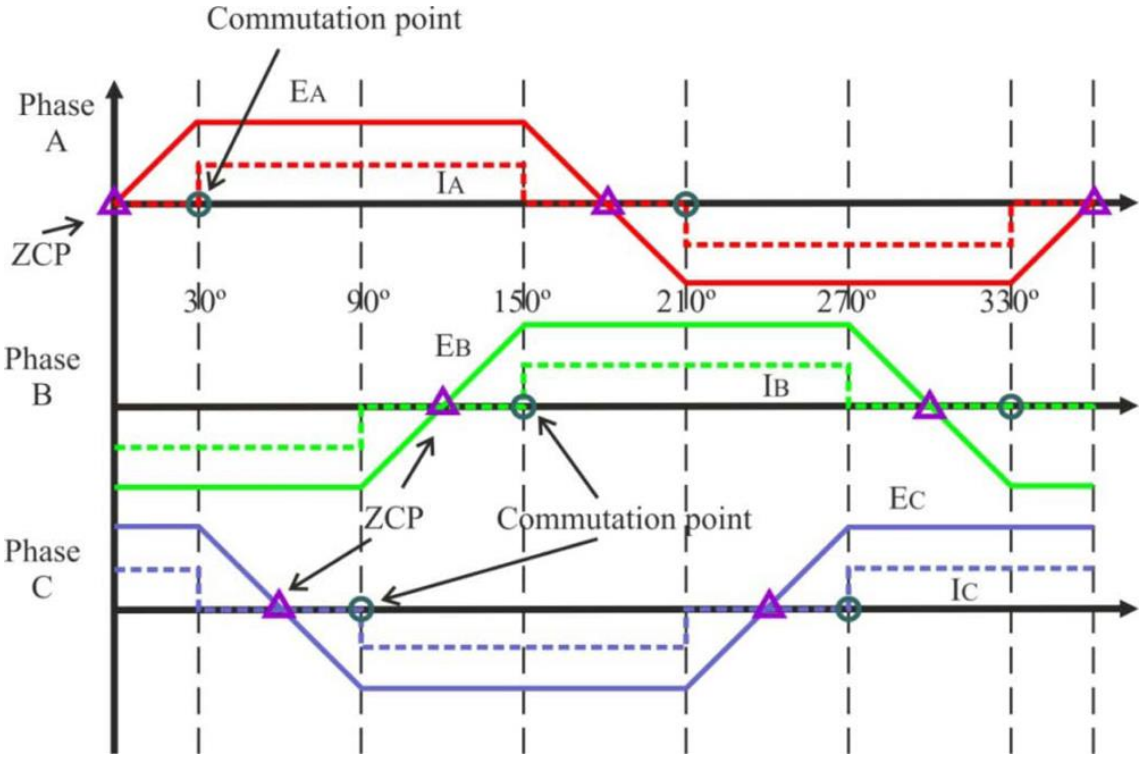

Figura 34. Gráfico donde aparece la fuerza contra electromotriz de cada fase y la corriente.

La mayor desventaja de este método es que a bajas velocidades el valor de la fuerza contra electromotriz es muy pequeño como para poder realizar el control en lazo cerrado. La implementación del lazo cerrado en este tipo de motores se produce una vez que han arrancado en lazo abierto y han alcanzado cierta velocidad para asegurarse de que el valor de la fuerza contra electromotriz es suficientemente grande como para medirla correctamente.

En la siguiente imagen, Figura 35, se puede apreciar la forma de onda de una fase del motor en color verde y en amarillo la forma de onda del puente en H. Se trata de la representación real del gráfico anterior.

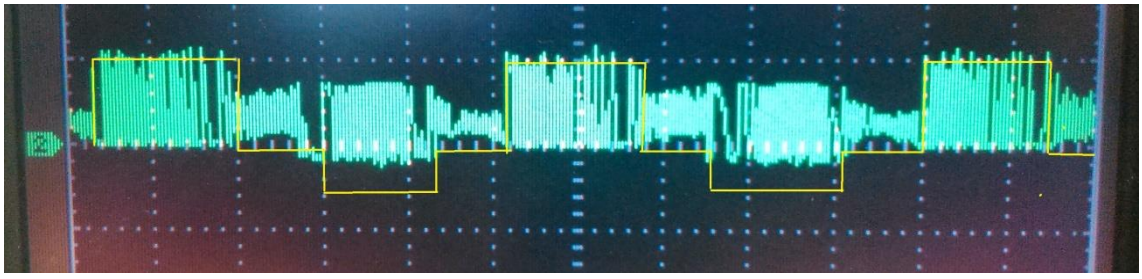

Figura 35. Fuerza contra electromotriz del motor con respecto a la forma de onda del puente en H.

7.3.Control de corriente

Tanto en este diseño como el diseño comercial no se realiza un control en lazo cerrado de corriente. Sí que se puede emplear la PWM en los pulsos para variar la corriente que absorbe el motor pero no se encuentra implementado en la tarjeta un circuito capaz de obtener la corriente y poder controlar en lazo cerrado la misma.

Debido al ámbito de uso del motor, no se implementa ese control de corriente ya que interesa un circuito de control simple para poder variar la velocidad del motor únicamente.

La falta del control de corriente implica el desconocimiento tanto de la potencia real que el motor absorbe como de los valores que toma en el tiempo.

El hardware que se necesitaría complicaría el circuito ya que la resistencia de los bobinados es muy pequeña y si se coloca una resistencia entre la fase y tierra para observar la caída de tensión de la misma y obtener por la Ley de Ohm el valor de la corriente podría alterar el valor real que puede consumir el motor. Por ello se precisa de un hardware más complejo para interferir lo menos posible en el funcionamiento del motor y obtener el valor real de la corriente.

Una forma de abordar el problema es con circuitos integrados específicos que sean un puente en H y que proporcionen un sensado de corriente y de la fuerza contra electromotriz inducida en cada fase.

8. Comunicación con el ordenador mediante RS-232

En el segundo diseño de la tarjeta se ha introducido el hardware necesario para la comunicación con un ordenador. Introduciendo unas líneas de código en el micro controlador se puede conseguir una visualización y control en tiempo real de variables del programa en el ordenador facilitando al usuario una visión del comportamiento del control del motor brushless y una rápida modificación de la velocidad o par deseados.

El software MPLAB contiene una herramienta llamada DMCI-Data Monitor Control Interface, con la que se podrán observar en tiempo real las variables del programa, la apariencia de la herramienta se muestra en la Figura 36.

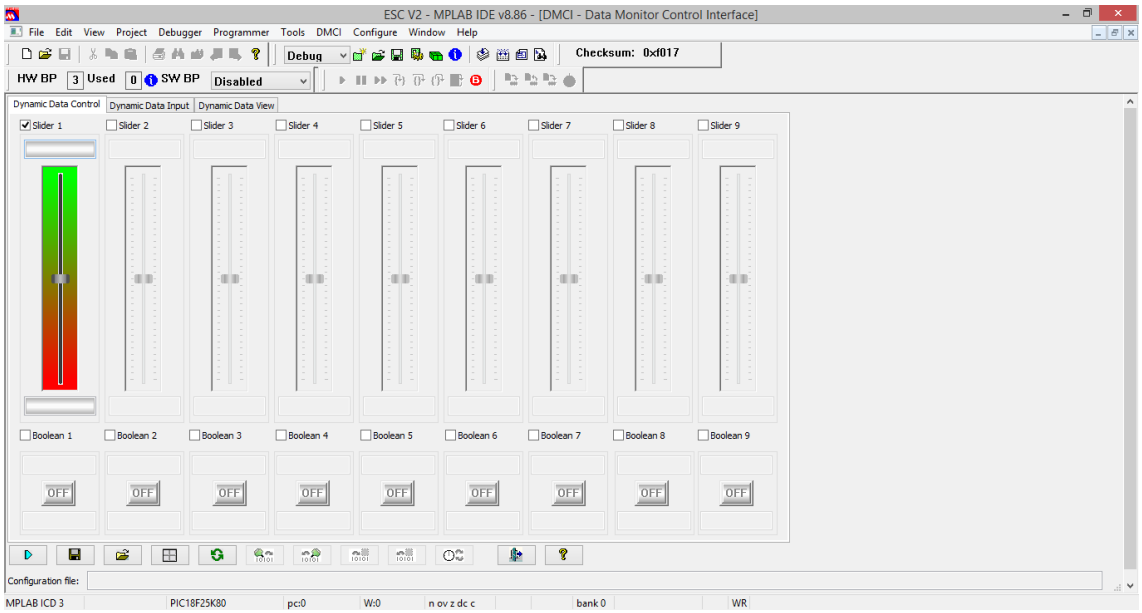

Figura 36. Herramienta DMCI del software MPLAB para el control y visualización en tiempo real.

El miro controlador emplea la comunicación USART para transmitir y recibir los datos en tiempo real. Un circuito integrado, el MAX232, eleva la tensión de los datos transmitidos de 5V a 12V para obtener la señal RS-232 que se comunique con el ordenador.

Para que se pueda transmitir la información en tiempo real, se ha modificado un código proporcionado por la empresa Microchip específico para este tipo de transmisiones con el ordenador y originalmente para micro controladores de gama más alta que el empleado en el proyecto. En el Anexo I se encuentra el código modificado para el micro controlador que se emplea junto con el código de control en lazo abierto del motor.

Este código se basa en el llamado RTDM- Real Time Data Monitor, donde se pueden visualizar datos del micro controlador específicos en tiempo real y modificarlos. Para ello implementa un protocolo de comunicación teniendo como base la comunicación RS232 que lee y escribe la memoria del micro controlador que se ha seleccionado en la herramienta DMCI previamente.

El micro controlador transmite los datos en formato de 8 bits con un bit de arranque a 0V y uno de parada a 5V cuando realiza una comunicación simple por USART. Como se precisa comunicarse con el ordenador se modifica la trama de datos que transmite en función de si se desea leer o escribir en el micro controlador.

Cada trama de datos se inicia con un Byte (8bits) que representa el valor de ‘$’ en el caso de solicitar comunicación, si es respuesta se inicia con dos Bytes de valor ‘+’ y ‘$’. Se cierra la trama con un Byte que representa el valor de ‘#’ para que posteriormente se añadan los dos Bytes del campo CRC- Cyclic Redundancy Check.

8.1.Lectura del micro controlador

El ordenador inicia la petición de leer los datos y la trama que envía se corresponde con la indicada en la Tabla 1.

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Start Code	$	1	0x24
Function Code	m	1	0x6D
Starting Address in little-endian format	[XXXXXXXX]	4	0x00000000-0xFFFFFFFF
Number Of Bytes in little-endian format (N)	[XXXX]	2	0x0000-0xFFFF
End Of Message Code	#	1	0x23
CRC16L	[XX]	1	0x00-0xFF
CRC16H	[XX]	1	0x00-0xFF

Tabla 1.Petición de lectura del ordenador.

Un ejemplo de la trama que envía el ordenador es: $m432100002000#[CRC16]

El micro controlador responde a la petición del ordenador de la forma que muestra la Tabla 2.

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Reply Code	+	1	0x2B
Start Code	$	1	0x24
Memory Values in little-endian format	[XX]…. [XX]	Nx2	0x00-0xFF
End Of Message Code	#	1	0x23
CRC16L	[XX]	1	0x00-0xFF
CRC16H	[XX]	1	0x00-0xFF

Tabla 2. Respuesta del micro controlador a la lectura.

Un ejemplo de la trama que enviaría el micro controlador es: +$EFBE#[CRC16]

8.2.Escritura del micro controlador

Para realizar la escritura de un dato en el micro controlador el ordenador inicia la petición con una trama de la forma que muestra la Tabla 3.

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Start Code	$	1	0x24
Function Code	M	1	0x4D
Starting Address in little-endian format	[XXXXXXXX]	4	0x00000000-0xFFFFFFFF
Number Of Bytes in little-endian format (N)	[XXXX]	2	0x0000-0xFFFF
Values to be written in little-endian format	[XX]	Nx2	0x00-0xFF
End Of Message Code	#	1	0x23
CRC16L	[XX]	1	0x00-0xFF
CRC16H	[XX]	1	0x00-0xFF

Tabla 3. Petición de escritura del ordenador.

Con respecto a la trama de lectura se puede observar que lo que se modifica es la función que el ordenador solicita. Un ejemplo: $M43210000EFBE#[CRC16]

La respuesta del micro controlador una vez recibida la petición de escritura será de la forma que muestra la Tabla 4.

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Reply Code	+	1	0x2B
Start Code	$	1	0x24
Command Acknowledge	O	1	0x4F
Command Acknowledge	K	1	0x4B
End Of Message Code	#	1	0x23
CRC16L	[XX]	1	0x00-0xFF
CRC16H	[XX]	1	0x00-0xFF

Tabla 4. Respuesta del micro controlador a la escritura.

Esta trama de respuesta es siempre la misma: +$OK#[CRC16]

8.3.Campo CRC

El campo CRC- Cyclic Redundancy Check tiene una dimensión de dos Bytes (16 bits) y contiene la comprobación de errores de la trama que se envía.

En el código se calcula este campo tal y como se muestra en la Figura 37.

```
unsigned int RTDM_CumulativeCrc16( unsigned char *buf, unsigned int u16Length, unsigned int u16CRC )
{
    unsigned int    u16Poly16 = 0xA001; // Bits 15, 13 and 0
    unsigned int    DATA_BITS = 8;      // Number of data bits
    unsigned int    u16BitIdx;
    unsigned int    u16MsgIdx;
    unsigned int    u16MsgByte;

    for( u16MsgIdx = 0; u16MsgIdx < u16Length; u16MsgIdx++ )
    {
//      asm( "mov.w w14,_wcopy" );
        u16MsgByte = 0x00FF &*buf++;
        for( u16BitIdx = 0; u16BitIdx < DATA_BITS; u16BitIdx++ )
        {
            if( (u16CRC ^ u16MsgByte) & 0x0001 )
            {
                u16CRC = ( u16CRC >> 1 ) ^ u16Poly16;
            }
            else
            {
                u16CRC = u16CRC >> 1;
            }

            u16MsgByte >>= 1;           // Right shift one to get to next bit
        }
    }

    return ( u16CRC );
}
```

Tabla 5. Cálculo CRC.

Para calcular el campo CRC se realizan estos pasos:

1) Se carga un valor de 16 bits: 0xFFFF y se le llama Registro CRC.
2) Se realiza una OR exclusiva a los 8 primeros bits de la trama con los 8 bits menos significativos del Registro CRC y se introduce el resultado en el Registro CRC.
3) Se desplaza un bit a la derecha del Registro CRC (hacia el bit menos significativo) y se llena el bit más significativo con un “0”. Se extrae y examina el bit menos significativo.
4) Si el bit menos significativo es “0” se repite el paso 3, sino se realiza una OR exclusiva al Registro CRC con el valor 0xA001.
5) Se repiten los pasos 3 y 4 hasta realizar 8 desplazamientos. Se habrán completado 8 bits.
6) Se repiten los pasos 2 a 5 para los siguientes 8 bits de la trama, así hasta procesar toda la trama.
7) El contenido final del Registro CRC será el valor del CRC.
8) Finalmente al colocar el valor del CRC en la trama se debe colocar primero los 8 bits menos significativos y después los más significativos.

Líneas futuras

Como línea futura se propone minimizar al máximo los componentes de tal forma que se obtenga una tarjeta de circuito impreso eficiente segura y con unas dimensiones mínimas permitiendo así la producción en cadena y poder acoplar el diseño a una aeronave no tripulada para que controle los motores.

También se puede mejorar el algoritmo de control introduciendo un control de la corriente, un freno del motor vía software o que emita pitidos para indicar que se encuentra listo para arrancar.

Tabla de Ilustraciones

Bibliografía y referencias

La gran mayoría de bibliografía ha sido obtenida de Internet:

- www.rcgroups.com
- www.microchip.com/forums
- http://www.digikey.com/es/articles/techzone/2013/jun/controlling-sensorless-bldc-motors-via-back-emf
- http://www.controlengeurope.com/article/33031/Sensorless-Control-of-a-Brushless-DC-Motor--Part-2-.aspx
- http://www.todopic.com.ar/foros/
- http://ww1.microchip.com/downloads/en/AppNotes/00857B.pdf

Referencias:

[1]http://www.hobbyking.com/hobbyking/store/__9484__Mystery_20A_Brushless_Speed_Controller_Blue_Series_.html

Anexo I

En este anexo se encuentra el código en lenguaje C con el que se ha programado el micro controlador PIC18F25K80 para controlar en lazo abierto el motor y comunicarse con el DMCI de MPLAB.

MainProyecto.c

```
#include "p18f25k80.h"
#include "rtdmProyecto.h"
#include <timers.h>                    //Librería de los temporizadores
#include <pwm.h>                       //Librería de las funciones de la PWM

//declaración variables y funciones

void main(void);                       //Declaración de la función principal
void ISRTimer (void);                  //Declaración de la interrupción de alta prioridad
void ISRTimer0 (void);                 //Declaración de la función de baja prioridad
void ISRRC (void);
void MotorStart (void);
void MotorStop (void);
// ******************************************************************************
// Section: File Scope or Global Constants
// ******************************************************************************
#define FCY                    64000000    //Instruction Cycle frequency
#define AMOUNT_OF_DATA_TO_BE_PLOTTED    120        //number of sanpshot samples expressed in 16-bit words since the Buffer is 16bit wide
unsigned int myVariable, frequency, amplitude;     //Varaible to be recored and its paramters to be modified using the DMCI sliders
unsigned int snapShotBufferPlottedInDMCI[AMOUNT_OF_DATA_TO_BE_PLOTTED]; //buffer where the data is recorded
unsigned int *pointerToSnapShotBuffer = &snapShotBufferPlottedInDMCI[0]; //Tail pointer required to plot circular buffers in DMCI
unsigned int *pointerToSnapShotBufferUpperLimit = &snapShotBufferPlottedInDMCI[0] + AMOUNT_OF_DATA_TO_BE_PLOTTED - 1;
unsigned char fasesC[6]={0x32,0x36,0x36,0x32,0x3A,0x3A}; //Vector con los valores de la forma de onda para el puerto C
unsigned char fasesB[6]={0x14,0x04,0x08,0x28,0x20,0x10}; //Vector con los valores de la forma de onda para el puerto B
unsigned int pos_fases=0,dutycycle=0,periodo=0,frec=0,inicio=0;                 //Declaración de las variables empleadas
unsigned int start=0,vel=0;
//Buffer Upper limit
struct
{
   unsigned    start : 1;   //variable to start
   unsigned    acel : 1;    //variable to acel or decel
  unsigned TrigggerSnapshot: 1;
   unsigned    unused : 5;
} RTDM;


void main( void )
{
TRISC=0;            //Puerto C salidas binarias
TRISB=0;            //Puerto B salidas binarias
TRISA=0;            //Puerto A salidas binarias
```

```
        PORTA=0b00001000;       //Se pone en alto RA4 para activar el driver de la fase A

        //Prescaler 1:4; TMR1 Preload = 25536; Actual Interrupt Time : 10 ms
          T1CON   = 0x01;
          TMR1H   = 0xC1;
          TMR1L   = 0x80;
        //Prescaler 1:64; TMR0 Preload = 24036; Actual Interrupt Time : 166 ms
          T0CON   = 0x85;
          TMR0H   = 0x5D;
          TMR0L   = 0xE4;
        PIR1bits.TMR1IF= 0;             //Se deshabilita el flag de las interrupciones del Timer 1
        PIE1bits.TMR1IE= 1;             //Se habilidan las interrupciones del Timer 1
        INTCON= 0xE0;                   //Habilitadas las interrupciones globales, de periféricos y del
Timer 0
        PIE1bits.RC1IE=1;               //Habilitado el flag de interrupción de RC1
        RCONbits.IPEN=1;                //Habilitadas las prioridades de las interrupciones
        IPR1bits.RC1IP=0;               //Interrupción de baja prioridad para RC1
        INTCON2bits.TMR0IP=1;  //alta prioridad timer 0
        IPR1bits.TMR1IP=0;              //baja prioridad timer 1

        frec=23869;                     //Valor inicial de la variable frec
        inicio=5;

        //PWM Configuration of Timer 2
        //T2CON = 0b00000111 ;
        //periodo=0x31;
        //dutycycle=15;

           // NOTE: DMCI is configured to auto-adjust the plot min and max values,
           //To notice amplitude changes on the DMCI plot, right click on the DMCI plot
           // goto to Extended Functions Menu > Customization > Axis, set the desired
           //min an max values for the Y AXIS
        RTDM_Start();                   //RTDM start function

           // Overview:
           // Here is where the RTDM code initilizes the UART to be used to
           // exchange data wiht the host PC
           // Note:
           // Some processors may have 2 UART modules, that is why it is required to
           // specify wich UART module is going to be used by RTDM
        ////      RTDM_ProcessMsgs();             // Overview:
              // Here is where the RTDM code process the message received and then
              // executes the required task. These tasks are reading an specified memory
              // location, writing an specified memory location, receive a communication
              // link sanity check command, or being asked for the size of the bufffers.


        //while(RTDM.start==0)
        //{
        //MotorStop();
        //}
        //WriteTimer0(frec);
        //T1CONbits.TMR1ON=1;  //El Timer 1 se activa
        //T0CONbits.TMR0ON=1;  //El Timer 0 se sactiva
        while(1)
        {
        RTDM_ProcessMsgs();
        if(RTDM.start==0)
          {
                    MotorStop();      //Parada del motor
```

```
                inicio=5;       //Si se desea volver a arrancar se tiene que partir desde la Fase 1
        }
        else if(inicio>0)
        {
                MotorStart();   //Arranque del motor
        }


        //Ajuste PWM
                // FASE A
                //      if(pos_fases==1|pos_fases==2)   //Cuando la posición del vector es la adecuada para activar los mosfet canal P de la fase
                //      {
                //              CloseEPWM1();           //Se desactiva la PWM que se encuentra activa
                //              OpenPWM2 ( periodo, CCP_2_SEL_TMR12 );  //Se ajusta y activa la nueva PWM
                //              SetDCPWM2(dutycycle); //Se ajusta el valor del ciclo de trabajo
                //      }
                // FASE B
                //      else if(pos_fases==3|pos_fases==4)      //Cuando la posición del vector es la adecuada para activar los mosfet canal P de la fase
                //      {
                //              ClosePWM2();            //Se desactiva la PWM que se encuentra activa
                //              OpenPWM5 (periodo, CCP_5_SEL_TMR12);    //Se ajusta y activa la nueva PWM
                //              SetDCPWM5(dutycycle); //Se ajusta el valor del ciclo de trabajo
                //      }
                // FASE C
                //      else if(pos_fases==5|pos_fases==0)      //Cuando la posición del vector es la adecuada para activar los mosfet canal P de la fase
                //      {
                //              ClosePWM5();            //Se desactiva la PWM que se encuentra activa
                //              OpenEPWM1 (periodo, CCP_4_SEL_TMR12);   //Se ajusta y activa la nueva PWM
                //              SetDCEPWM1(dutycycle);          //Se ajusta el valor del ciclo de trabajo
                //      }

        }

        //      if(frec>vel)
        //      {
        //              RTDM.acel=0;            //Freno motor
        //              T1CONbits.TMR1ON=1; //El Timer 1 se activa
        //      }
        //      else if(frec<vel)
        //      {
        //              RTDM.acel=1;            //Aceleración motor
        //              T1CONbits.TMR1ON=1; //El Timer 1 se activa
        //      }
        //      else if(frec==vel)
        //      {
        //              T1CONbits.TMR1ON=0; //Se desactiva el Timer 1 para que no varíe la variable frec
        //      }
        //    //If the TriggerSnapShot is ON then the values of the variable are recorded on the SnapShot Buffer,
```

```
//      // if the TriggerSnapShot is OFF the values are not recorded  this is a lock condition in order to prevent that the
//          // data recorded on the snapshot buffer are being updated while the target device is sending/receiving information
//      //to DMCI. If the data is  corrupted while a TX is in progress, DMCI will flush the received data and it
//      //will display an error message. The values won't be displayed on the graph
//      if( RTDM.TrigggerSnapshot )
//      {
//         *pointerToSnapShotBuffer++ = vel;    //Recordinf values
//         if( pointerToSnapShotBuffer > pointerToSnapShotBufferUpperLimit )
//         {      //SnapShot Buffer is a circular buffer
//            pointerToSnapShotBuffer = snapShotBufferPlottedInDMCI;
//
//             // RTDM.TrigggerSnapshot = 0; //Turning OFF the snapshot mode indicating that new data is available to be sent
//            //DMCI user has to push the RECEIVE button in order to retrive data from the target
//            //device
//         }
//      }
//

      }


      //Interrupciones

      //Interrupción baja prioridad

      //-----------------------------------------------------------------------
      #pragma code VectorInterrupcion = 0x18                  //Sección de código a partir de la dirección 0x18
      void VectorInterrupcion (void)
      {
        _asm
         goto ISRTimer
         _endasm
      }

      #pragma code                                                                  //Se cierra la sección
      //-----------------------------------------------------------------------

      #pragma interrupt ISRTimer

      void ISRTimer (void)                  //Función de la interrupción de baja prioridad
      {
            if(PIR1bits.TMR1IF==1)  //Selección de la interrupción del Timer 1
            {
                  PIR1bits.TMR1IF= 0;      //Se deshabilita el flag de interrupción del Timer 1
                   TMR1H          =0xC1; //T1H;//0x63;0xC1;
                   TMR1L          =0x80; //T1L;//0xC0;0x80;

                  if(RTDM.acel==1)          //En función de si se desea acelerar o frenar
                  {
                        frec=frec+1;        //aumenta el valor de la variable frec +1
                  }
                  else if(RTDM.acel==0)
                  {
                        frec=frec-1;        //disminuye el valor de la variable frec -1
```

```
                }
        }
        if(PIR1bits.RC1IF == 1)
        {
                ISRRC();                //Función de interrupción recepción de datos
        }
}
//---------------------------------------------------------------------------

//Interrupción alta prioridad

//---------------------------------------------------------------------------
#pragma code VectorInterrupcion0 = 0x08                 //Sección de código a partir de la dirección 0x08
void VectorInterrupcion0 (void)
{
  _asm
   goto ISRTimer0
   _endasm
}

#pragma code                                                                    //Se cierra la sección
//---------------------------------------------------------------------------

#pragma interrupt ISRTimer0

void ISRTimer0 (void)           // Función de la interrupción de alta prioridad
{
   if(INTCONbits.TMR0IF==1)     //Selección de la interrupción del Timer 0
        {
                INTCONbits.TMR0IF=0; //Se deshabilita el flag de interrupción del Timer 0
                WriteTimer0(frec);
                PORTC=fasesC[pos_fases];        //Se carga la posición del vector correspondiente en el puerto C
                PORTB=fasesB[pos_fases];        //Se carga la posición del vector correspondiente en el puerto B
                if(pos_fases==5) //En caso de que la posición del vector supere el valor 5
                        {
                                pos_fases=0;    //Se resetea
                        }
                else
                        {
                                pos_fases++;    //Sino aumenta
                        }
        }
}

//  RTDM_Close();   // Overview:

   // Here is where the RTDM code closes the UART used to
   // exchange data wiht the host PC


void MotorStart (void)          //Arranque inicial del motor
{
        RTDM.acel=1;            //Aceleración del motor
                if(inicio==5)           //Fase 0, ajuste inicial del prescaler del Timer 0 y cuenta inicial
                {
```

```
                                        T0CON  = 0x85; //Prescaler 1:64
                                    frec=23869;                 //Valor inicial de la variable frec
                                        WriteTimer0(frec);      //Cargar el valor de frec en el
Timer 0
                                        T1CONbits.TMR1ON=1; //El Timer 1 se activa
                                        inicio=inicio-1;
                                }
                                if((inicio==4)&&(frec>55000))   //Fase 1 completada, se modifica el
prescaler del Timer 0 y se parte de la misma frecuencia que se encontraba el motor
                                {
                                        frec=23392;
                                    T0CON           = 0x83; //Prescaler 1:16
                                        inicio=inicio-1;
                                }
                                else if((inicio==3)&&(frec>55000))      //Fase 2 completada, se modifica el
prescaler del Timer 0 y se parte de la misma frecuencia que se encontraba el motor
                                {
                                                frec=23392;
                                      T0CON           = 0x81; //Prescaler 1:4
                                        inicio=inicio-1;
                                }
                                else if((inicio==2)&&(frec>55000))      //Fase 3 completada, se modifica el
prescaler del Timer 0 y se parte de la misma frecuencia que se encontraba el motor
                                {
                                                frec=23392;
                                      T0CON           = 0x88; //Prescaler 1:1
                                                inicio=inicio-1;
                                }
                                else if((inicio==1)&&(frec>50000))      //Fase 4 completada, se desactiva
el Timer 1 para que no aumente el valor de frec. Finalizada la aceleración inicial
                                {
                                        T1CONbits.TMR1ON=0; //El Timer 1 se desactiva
                                        frec=50000;
                                        frec=vel;
                                        inicio=inicio-1;
                                }
        }

        void MotorStop (void)           //Parada del motor
        {
                T1CONbits.TMR1ON=0; //El Timer 1 se desactiva
                T0CONbits.TMR0ON=0; //El Timer 0 se desactiva
                PORTC=0;                        //Todos los puertos C y B a 0
                PORTB=0;
            RTDM_ProcessMsgs();
        }
        /*****************************************************************************
*
         End of File
        */
```

RtdmProyecto.c

```
        #include "rtdmProyecto.h"

        // *****************************************************************************
        // *****************************************************************************
        // Section: File Scope or Global Constants
        // *****************************************************************************
        // *****************************************************************************
```

```
/* Received data is stored in array rtdmRxBuffer  */
unsigned char      rtdmRxBuffer[RTDM_RXBUFFERSIZE];
unsigned char      *rtdmRxBufferLoLimit = rtdmRxBuffer;
unsigned char      *rtdmRxBufferHiLimit = rtdmRxBuffer + RTDM_RXBUFFERSIZE - 1;
unsigned char      *rtdmRxBufferIndex = rtdmRxBuffer;
unsigned char      *rtdmRxBufferStartMsgPointer;
unsigned char      *rtdmRxBufferEndMsgPointer;

/* Data to be transmitted using UART communication module */
const unsigned char rtdmTxdata[] = { 'R', 'T', 'D', 'M', '\0' };
const unsigned char rtdmSanityCheckOK[] = { '+', '$', 'R', 'T', 'D', 'M', '#', 0x1B, 0x86, '\0' };
const unsigned char rtdmWriteMemoryOK[] = { '+', '$', 'O', 'K', '#', 0x4C, 0x08, '\0' };
const unsigned char rtdmErrorIllegalFunction[] = { '-', '$', 'E', 0x01, '#', 0xD3, 0x6A, '\0' };
unsigned char      rtdmErrorFrame[] = { '-', '$', 'E', 0, '#', 0, 0, '\0' };

/* Temp variables used to calculate the CRC16*/
unsigned int       rtdmcrcTemp, rtdmcrcTempH, rtdmcrcTempL;

//unsigned char      rtdmPacketBuf[16];

/*Structure enclosing the rtdm flags*/
struct
{
   unsigned    MessageReceived : 1;
   unsigned    TransmitNow : 1;
   unsigned    unused : 6;
} rtdmFlags;

unsigned char   rtdmPacketBuf[16];

/* UART Configuration data */
/* Holds the value of uart config reg */
unsigned int    rtdm_Uart_Mode_Value;

/* Holds the information regarding uart TX & RX interrupt modes */
unsigned int    rtdm_Uart_Sta_Value;

#if ( RTDM_UART == 1 )

/*****************************************************************************
* Function:     RTDM_Start()
*
* Output:  return 0 if no errors
*
* Overview: Here is where the RTDM code initilizes the UART to be used to
*   exchange data wiht the host PC
*
* Note:  Some processors may have more UART modules, that is why it is required to
*   specify wich UART module is going to be used by RTDM
*****************************************************************************
/

int RTDM_Start( void )
{
   /********************** UART CONFIGURATIION ***************************/
   /* Turn off UART1 module */
   Close1USART();

//    /* Configure UART1 module to transmit 8 bit data with one stopbit.  */
baud1USART(BAUD_IDLE_RX_PIN_STATE_HIGH &
```

```
                BAUD_IDLE_TX_PIN_STATE_HIGH &
                BAUD_16_BIT_RATE &
                BAUD_WAKEUP_OFF &
                BAUD_AUTO_OFF);

    Open1USART( USART_TX_INT_OFF & USART_RX_INT_ON &
            USART_ASYNCH_MODE & USART_EIGHT_BIT &
            USART_CONT_RX &
            USART_BRGH_HIGH & USART_ADDEN_OFF,
            138); //1666 PARA 9600

    /************** RTDM Flags Configuration & Initial Values ****************/
    rtdmFlags.MessageReceived = 0;
    rtdmFlags.MessageReceived = 0;
    rtdmRxBufferIndex = rtdmRxBufferLoLimit;
    rtdmRxBufferStartMsgPointer = rtdmRxBufferLoLimit;
    rtdmRxBufferEndMsgPointer = rtdmRxBufferLoLimit;

    return ( 0 );
}
#endif

/*****************************************************************************
* Function:     CloseRTDM()
*
* PreCondition:  None
*
* Input:        None
*
* Output:       return 0 if no errors
*
* Overview:     Here is where the RTDM code closes the UART used to
*               exchange data wiht the host PC
*
* Note:         Some processors may have more UART modules, that is why it is
*               required to specify wich UART module is going to be used by RTDM
*****************************************************************************
/
int CloseRTDM( void )
{
    int nRet = 0;
    Close1USART();
    return ( nRet );
}

/*****************************************************************************
* Function:     RTDM_ProcessMsgs()
*
* PreCondition:  None
*
* Input:        None
*
* Output:       return 0 if no errors
*
* Overview:     Here is where the RTDM code process the message received and
*               then executes the required task. These tasks are reading an
*               specified memory location, writing an specified memory location,
*               receive a communication link sanity check command, or being
*               asked for the size of the bufffers.
*
```

```
        * Note:        Some processors may have more UART modules, that is why it is
        *              required to specify wich UART module is going to be used by RTDM
        ********************************************************************************
/
        int RTDM_ProcessMsgs( void )
        {
           //Local pointer management variables
           unsigned long int   *rtdmpu32AddressTemp;
           unsigned char       *rtdmpucWrData;
           unsigned char       *rtdmpucRdData;
           unsigned char       *rtdmpucWrAddr;
           unsigned short      rtdmNumBytes;

           unsigned int        rtdmProcessMsgsTemp1;
           unsigned int        rtdmProcessMsgsTemp2;
           unsigned int        N;

           if( !rtdmFlags.MessageReceived )
           {
              return ( -1 );
           }

          rtdmcrcTemp = RTDM_CumulativeCrc16( rtdmRxBufferStartMsgPointer,( unsigned int )
(rtdmRxBufferEndMsgPointer - rtdmRxBufferStartMsgPointer) + 1,0xFFFF );

           rtdmcrcTempH = ( rtdmcrcTemp & 0xFF00 ) >> 8;
           rtdmcrcTempL = rtdmcrcTemp & 0x00FF;
           rtdmProcessMsgsTemp1 = ( unsigned int ) *( (rtdmRxBufferEndMsgPointer) + 2 );
           rtdmProcessMsgsTemp2 = ( unsigned int ) *( (rtdmRxBufferEndMsgPointer) + 1 );

           rtdmRxBufferStartMsgPointer += 2;
         if( (rtdmProcessMsgsTemp1 == ( unsigned ) rtdmcrcTempH) && (rtdmProcessMsgsTemp2
==(unsigned) rtdmcrcTempL) )

           {
              switch( *((rtdmRxBufferLoLimit) + 1) )
              {
                 case 'm':
                    /*************** Extract Address **************/
                    //Capture address as 32 bit quantity to match protocol definition.
                    rtdmpu32AddressTemp = ( ( unsigned long * ) rtdmRxBufferStartMsgPointer );

                    //Increment receive buffer pointer to length field.
                    rtdmRxBufferStartMsgPointer += sizeof( unsigned long );

                    //Init a byte oriented data pointer
                    rtdmpucRdData = ( unsigned char * ) ( ( unsigned int ) *rtdmpu32AddressTemp );

                    /********** Extract Number of Bytes ************/
                    //Capture address as 16 bit quantity to match protocol definition.
                    rtdmNumBytes = *( ( unsigned short * ) rtdmRxBufferStartMsgPointer );

                    //Increment receive buffer pointer to start of data payload.
                    rtdmRxBufferStartMsgPointer += sizeof( unsigned short );

                    //Init the CRC seed for the cumulative checksum calculation.
                    rtdmcrcTemp = 0xffff;

                    //Add packet header prefix
                    rtdmPacketBuf[0] = '+';
```

```
                rtdmPacketBuf[1] = '$';

                //Add null terminator for putsUARTx function...
                rtdmPacketBuf[2] = 0;

                //Calc header prefix checksum piece
                rtdmcrcTemp = RTDM_CumulativeCrc16( rtdmPacketBuf, 2, rtdmcrcTemp );

                //Calc data payload checksum
                rtdmcrcTemp = RTDM_CumulativeCrc16( rtdmpucRdData, rtdmNumBytes,
rtdmcrcTemp );

                //Send packet header. Use string function to save code space...
                puts1USART( ( unsigned int * ) rtdmPacketBuf );
                while( Busy1USART() );

                //Send data portion of message...
                while( rtdmNumBytes-- )
                {
                   Write1USART( *rtdmpucRdData++ );
                   while( Busy1USART() );
                }

                //Add packet trailer
                rtdmPacketBuf[0] = '#';
                rtdmcrcTemp = RTDM_CumulativeCrc16( rtdmPacketBuf, 1, rtdmcrcTemp );

                //Add checksum bytes to packet
                rtdmPacketBuf[1] = rtdmcrcTemp & 0x00FF;
                rtdmPacketBuf[2] = ( rtdmcrcTemp & 0xFF00 ) >> 8;

                //Send packet trailer and checksum.
                for( N = 0; N < 3; N++ )
                {
                   Write1USART( rtdmPacketBuf[N] );
                   while( Busy1USART() );
                }

                break;

             case 'M':
                {
                   /*************** Extract Address **************/
                   //Capture address as 32 bit quantity to match protocol definition.
                   rtdmpu32AddressTemp = ( unsigned long * ) rtdmRxBufferStartMsgPointer;

                   //Increment receive buffer pointer to length field.
                   rtdmRxBufferStartMsgPointer += sizeof( unsigned long );

                   // Init a byte oriented address pointer for use in incrementing
                   //through the address range properly as we write each byte of data
                   //in the range (length) of this write request.
                   rtdmpucWrAddr = ( unsigned char * ) ( ( unsigned int ) *rtdmpu32AddressTemp );

                   /********* Extract Number of Bytes ************/
                   //MEllis Capture length as 16 bit quantity to match protocol definition.
                   rtdmNumBytes = *( ( unsigned short * ) rtdmRxBufferStartMsgPointer );

                   //MEllis Increment receive buffer pointer to start of data payload.
                   rtdmRxBufferStartMsgPointer += sizeof( unsigned short );
```

```
      /********** Extract Data ************/
      //Init a byte oriented data pointer so that we can increment a byte at at
      //time for as many bytes as are in the range for this write.
      rtdmpucWrData = rtdmRxBufferStartMsgPointer;

      //*** Write Data in specified RAM location *****
      //Important to increment through address range using byte oriented address and data
      //pointers. Otherwise, single byte or odd byte ranges do not get written correctly.
      while( rtdmNumBytes-- )
      {
         *rtdmpucWrAddr++ = *rtdmpucWrData++;
      }

      //Transmit OK message
      puts1USART( ( unsigned int * ) rtdmWriteMemoryOK );

      /* Wait for  transmission to complete */
      while( Busy1USART() );
      break;
   }

case 's':
   {
      /* Load transmit buffer and transmit the same till null character is encountered */
      //Transmit OK message
      puts1USART( ( unsigned int * ) rtdmSanityCheckOK );

      /* Wait for  transmission to complete */
      while( Busy1USART() );
      break;
   }

case 'L':
   rtdmcrcTemp = 0xffff;   //Init the CRC seed.
   rtdmPacketBuf[0] = '+';
   rtdmPacketBuf[1] = '$';

   //Size of the rtdm Receive buffer.
   rtdmPacketBuf[2] = ( sizeof(rtdmRxBuffer) & 0x00FF );
   rtdmPacketBuf[3] = ( sizeof(rtdmRxBuffer) & 0xFF00 ) >> 8;

   //Note: We dod not utilize a transmit buffer since any data memory source is
   //essentially already buffered. So the transmit limit is now just a way to
   //limit the total message length that a client make with any single read request.
   rtdmPacketBuf[4] = ( RTDM_MAX_XMIT_LEN & 0x00FF );
   rtdmPacketBuf[5] = ( RTDM_MAX_XMIT_LEN & 0xFF00 ) >> 8;
   rtdmPacketBuf[6] = '#';
   rtdmcrcTemp = RTDM_CumulativeCrc16( rtdmPacketBuf, 7, rtdmcrcTemp );
   rtdmPacketBuf[7] = ( rtdmcrcTemp & 0x00FF );
   rtdmPacketBuf[8] = ( rtdmcrcTemp & 0xFF00 ) >> 8;

   //Send completed message which is 9 bytes in length.
   for( N = 0; N < 9; N++ )
   {
      Write1USART( rtdmPacketBuf[N] );
      while( Busy1USART() );
   }

   break;
```

```
            default:
               // ---> COMMAND SUPPORTED?? IF NOT ERROR HANDLER
               //Transmit ERROR message 1
              puts1USART( ( unsigned int * ) rtdmErrorIllegalFunction );
               /* Wait for  transmission to complete */
               while( Busy1USART() );
               break;
         }
       }

       memset( &rtdmRxBuffer, 0, sizeof(rtdmRxBuffer) );

       rtdmFlags.MessageReceived = 0;
       rtdmRxBufferIndex = rtdmRxBufferLoLimit;
       rtdmRxBufferStartMsgPointer = rtdmRxBufferLoLimit;
       rtdmRxBufferEndMsgPointer = rtdmRxBufferLoLimit;

       return ( 0 );
     }

     /*******************************************************************************
     * Function:     _U1RXInterrupt(void)
     *
     * Output:  void
     *
     * Overview: Here is where the rtdm receives the messages using the UART receiver
     *   interrupt, If polling method is selected in the RTDMUSER.h file then
     *   the user application should call the RTDM_ProcessMsgs routine in order
     *   to precess up comming messages. If polling method is disabled then the
     *   RTDM_ProcessMsgs routine is called in the UART received interrupt
     *   routine.
     *
     * Note:  Some processors may have more UART modules, that is why it is required to
     *   specify wich UART module is going to be used by RTDM
     *******************************************************************************
/
     //Interrupción de baja prioridad
     //------------------------------------------------------------------------
     //#pragma code VectorInterrupcion = 0x18                    //Sección de código a partir de la
dirección 0x18
     //void VectorInterrupcion (void)
     //{
     //  _asm
     // goto ISRRC
     //   _endasm
     //}
     //
     //#pragma code                                                                    //Se
cierra la sección
     ////------------------------------------------------------------------------
     //
     //#pragma interrupt ISRRC
     /* This is UART1 receive ISR Polling RTDM Messages*/
     void ISRRC ( void )
     {
     #if ( RTDM_POLLING == YES )
     {
      //   PIR1bits.RC1IF = 0;      //Clean RC interrupt
```

```
    /* Read the receive buffer until at least one or more character can be read */
    while( DataRdy1USART() )
    {
       *( rtdmRxBufferIndex++ ) = Read1USART();
    }

    rtdmRxBufferEndMsgPointer = rtdmRxBufferIndex - 3;
    if( rtdmRxBufferIndex > (rtdmRxBufferHiLimit - 1) )
    {
       rtdmRxBufferIndex = rtdmRxBufferLoLimit;
       rtdmRxBufferEndMsgPointer = rtdmRxBufferHiLimit - 1;
    }

    if( *(rtdmRxBufferStartMsgPointer) == '$' )
    {
       if( *(rtdmRxBufferEndMsgPointer) == '#' )
       {
          rtdmFlags.MessageReceived = 1;
       }
    }
    else
    {
       rtdmRxBufferIndex = rtdmRxBufferLoLimit;
    }
}
#endif
}

/*******************************************************************************
* Function:     _U1RXInterrupt(void)
*
* Output:  void
*
* Overview: Here is where the RTDM receives the messages using the UART receiver
*   interrupt, If polling method is selected in the RTDMUSER.h file then
*   the user application should call the RTDM_ProcessMsgs routine in order
*   to precess up comming messages. If polling method is disabled then the
*   RTDM_ProcessMsgs routine is called in the UART received interrupt
*   routine.
*
* Note:  Some processors may have more UART modules, that is why it is required to
*   specify wich UART module is going to be used by RTDM
*******************************************************************************
/
//    #else
//{
///* This is UART1 receive ISR without polling RTDM Messages */
////void __attribute__ ( (__interrupt__, no_auto_psv) ) _U1RXInterrupt( void )
////{
//     PIR1bits.RC1IF = 0;     //Clean RC interrupt
//
//    /* Read the receive buffer until at least one or more character can be read */
//    while( DataRdy1USART() )
//    {
//       *( rtdmRxBufferIndex++ ) = Read1USART();
//    }
//
//    rtdmRxBufferEndMsgPointer = rtdmRxBufferIndex - 3;
//    if( rtdmRxBufferIndex > (rtdmRxBufferHiLimit - 1) )
//    {
```

```
//      rtdmRxBufferIndex = rtdmRxBufferLoLimit;
//      rtdmRxBufferEndMsgPointer = rtdmRxBufferHiLimit - 1;
//   }
//
//   if( *(rtdmRxBufferStartMsgPointer) == '$' )
//   {
//      if( *(rtdmRxBufferEndMsgPointer) == '#' )
//      {
//         rtdmFlags.MessageReceived = 1;
//         RTDM_ProcessMsgs();
//      }
//   }
//   else
//   {
//      rtdmRxBufferIndex = rtdmRxBufferLoLimit;
//   }
//}

//#endif
//}
//
//Conditionally compile for speed performance or minimum code size.
#if ( RTDM_MIN_CODE_SIZE == NO )

//When RTDM_MIN_CODE_SIZE is not defined we employ a table driven crc
//calculation with pre-calculated polyniomial values to speed-up
//checksum calculation time.
//const unsigned int  crc_16_tab[] = { 0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280,
0xc241, 0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440, 0xcc01, 0x0cc0, 0x0d80,
0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40, 0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880,
0xc841, 0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40, 0x1e00, 0xdec1, 0xdf81,
0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41, 0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680,
0xd641, 0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040, 0xf001, 0x30c0, 0x3180,
0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240, 0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480,
0xf441, 0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41, 0xfa01, 0x3ac0, 0x3b80,
0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840, 0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80,
0xea41, 0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40, 0xe401, 0x24c0, 0x2580,
0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640, 0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080,
0xe041, 0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240, 0x6600, 0xa6c1, 0xa781,
0x6740, 0xa501, 0x65c0, 0x6480, 0xa441, 0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80,
0xae41, 0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840, 0x7800, 0xb8c1, 0xb981,
0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41, 0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81,
0x7c40, 0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640, 0x7200, 0xb2c1, 0xb381,
0x7340, 0xb101, 0x71c0, 0x7080, 0xb041, 0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x53c0, 0x5280,
0x9241, 0x9601, 0x56c0, 0x5780, 0x9741, 0x5500, 0x95c1, 0x9481, 0x5440, 0x9c01, 0x5cc0, 0x5d80,
0x9d41, 0x5f00, 0x9fc1, 0x9e81, 0x5e40, 0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880,
0x9841, 0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40, 0x4e00, 0x8ec1, 0x8f81,
0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41, 0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680,
0x8641, 0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040 };
const unsigned int crc_16_tab[] = {    0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70,
      0x81, 0x91, 0xA1, 0xB1, 0xC1, 0xD1, 0xE1, 0xF1,0x00, 0x21, 0x42, 0x63, 0x84, 0xA5,
0xC6, 0xE7,
      0x08, 0x29, 0x4A, 0x6B, 0x8C, 0xAD, 0xCE, 0xEF
};
/*****************************************************************************
* Function:  unsigned int  RTDM_CumulativeCrc16 (unsigned char *buf,
*         unsigned int u16Length, unsigned int u16CRC)
*
* PreCondition:None
*
```

```
        * Input:       None
        *
        * Output:      return CRC16
        *
        * Overview:    This routine calculates the polynomial for the checksum byte on
        *              the fly every time. Saves code space because no const table is
        *              required. This approach saves code space but yields slower
        *              throughput performance.
        *
        * Note:        Some processors may have more UART modules, that is why it is
        *              required to specify wich UART module is going to be used by RTDM
        ********************************************************************************
/
        unsigned int RTDM_CumulativeCrc16( unsigned char *buf, unsigned int bsize, unsigned int
crcSeed )
        {
           unsigned char   *pData = buf;

           while( bsize-- )
           {
              crcSeed = ( unsigned int ) ( crcSeed >> 8 ) ^ crc_16_tab[( crcSeed ^ *pData++ ) & 0xff];
           }

           return ( crcSeed );
        }

        #else //This is when the _RTDM_CODE_FOOTPRINT = RTDM_MIN_SIZE

        /********************************************************************************
        * Function:  unsigned int  RTDM_CumulativeCrc16 (unsigned char *buf,
        *            unsigned int u16Length, unsigned int u16CRC)
        *
        * PreCondition:None
        *
        * Input:       None
        *
        * Output:      return CRC16
        *
        * Overview:    This routine calculates the polynomial for the checksum byte on
        *              the fly every time. Saves code space because no const table is
        *              required. This approach saves code space but yields slower
        *              throughput performance.
        *
        * Note:        Some processors may have more UART modules, that is why it is
        *              required to specify wich UART module is going to be used by RTDM
        ********************************************************************************
/
        int wcopy;
        unsigned int RTDM_CumulativeCrc16( unsigned char *buf, unsigned int u16Length, unsigned int
u16CRC )
        {
           unsigned int    u16Poly16 = 0xA001; // Bits 15, 13 and 0
           unsigned int    DATA_BITS = 8;      // Number of data bits
           unsigned int    u16BitIdx;
           unsigned int    u16MsgIdx;
           unsigned int    u16MsgByte;

           for( u16MsgIdx = 0; u16MsgIdx < u16Length; u16MsgIdx++ )
           {
         //    asm( "mov.w w14,_wcopy" );
```

```
            u16MsgByte = 0x00FF &*buf++;
            for( u16BitIdx = 0; u16BitIdx < DATA_BITS; u16BitIdx++ )
            {
               if( (u16CRC ^ u16MsgByte) & 0x0001 )
               {
                  u16CRC = ( u16CRC >> 1 ) ^ u16Poly16;
               }
               else
               {
                  u16CRC = u16CRC >> 1;
               }

               u16MsgByte >>= 1;          // Right shift one to get to next bit
            }
         }

         return ( u16CRC );
      }

      #endif /* End of #ifndef compilation condition. */

      /*****************************************************************************
*
       End of File
      */
```

RtdmProyecto.h

```
      #ifndef RTDM_H
         #define RTDM_H

      // *****************************************************************************
      // *****************************************************************************
      // Section: Included Files
      // *****************************************************************************
      // *****************************************************************************
      //    #include <stdint.h>
      //    #include <string.h>
         #include "usart.h"
         #include "rtdmuser PC.h"

         #ifdef __cplusplus      // Provide C++ Compatability
      extern "C"
      {
            #endif

         //
******************************************************************************
         //
******************************************************************************
         // Section: File Scope or Global Constants
         //
******************************************************************************
         //
******************************************************************************
            #if defined RTDM_FCY
               #if defined RTDM_BAUDRATE
                  #define RTDM_BRG    ( RTDM_FCY / (16 * RTDM_BAUDRATE) ) - 1
               #else
                  #error Cannot calculate BRG value. Please define RTDM_BAUDRATE in
RTDMUSER.h file
               #endif
```

```
            #else
               #error Cannot calculate RTDM_BRG value. Please define RTDM_FCY in RTDMUSER.h
file
            #endif
            #define RTDM_BAUDRATE_ACTUAL    ( RTDM_FCY / (16 * (RTDM_BRG + 1)) )
            #define   RTDM_BAUD_ERROR                    ( (RTDM_BAUDRATE_ACTUAL >
RTDM_BAUDRATE) ? RTDM_BAUDRATE_ACTUAL - \
                           RTDM_BAUDRATE              :              RTDM_BAUDRATE              -
RTDM_BAUDRATE_ACTUAL )
            #define   RTDM_BAUD_ERROR_PERCENT   (   ((RTDM_BAUD_ERROR   *   100)   +
(RTDM_BAUDRATE / 2)) / RTDM_BAUDRATE )
        //      #if ( RTDM_BAUD_ERROR_PERCENT > 2 )
        //         #error The value loaded to the BRG register produces a baud rate error higher than 2%
        //      #endif
        //
           //
*****************************************************************************
           //
*****************************************************************************
           // Section: Interface Routines
           //
*****************************************************************************
           //
*****************************************************************************
           int          RTDM_ProcessMsgs( void );
           int          RTDM_Close( void );
           int          RTDM_Start( void );
           unsigned  int       RTDM_CumulativeCrc16(  unsigned  char  *buf,  unsigned  int  u16Length,
unsigned int u16CRC );
          void ISRRC (void);                    //Declaración de la interrupción de alta prioridad

              #ifdef __cplusplus  // Provide C++ Compatibility
        }

           #endif
        #endif
```

RtdmuserProyecto.h

```
        #ifndef RTDMUSER_H
           #define RTDMUSER_H

           #ifdef __cplusplus                  // Provide C++ Compatability
        extern "C"
        {
             #endif

           //
*****************************************************************************
           //
*****************************************************************************
           // Section: File Scope or Global Constants
           //
*****************************************************************************
           //
*****************************************************************************
              #define YES 1
              #define NO  0
```

```
        /************************************ RTDM DEFINITIONS
*************************************/
            #define RTDM_FCY    64000000        //This define has to be the system operating freq, this

        //value is used to calculate the value of the BRG register
            #define RTDM_BAUDRATE    115200        //This is the desired baudrate for the UART
module to be

        //used by RTDM
            #define RTDM_UART   1               // This is the UART module to be used by RTDM. It has
only

        // two possible values: 1 or 2
        // For dsPIC33E and PIC24E, values 3 and 4 are also supported
            #define RTDM_UART_PRIORITY  1       //This the UART RX interrupt priority assigned to
receive

        // the RTDM messages
            #define RTDM_RXBUFFERSIZE   32      // This is the buffer size used by RTDM to handle
messaages
            #define RTDM_MAX_XMIT_LEN    0x1000   //This the size in bytes of the max num of
bytes allowed in

        //the RTDM protocol Frame
            #define RTDM_POLLING    YES         // This defines the mode that RTDM will be operating
in

        //user's application. If it is YES then the user should place the
        //RTDM_ProcessMsgs() function in the main loop.
        //In order to make sure that the messages are being preoccessed
        // it is recommended that the main loop always polls this
        //function as fast as possible
            #define RTDM_MIN_CODE_SIZE   YES       //When defined causes the RTDM library to
build  without

        //including a pre-calculated polynomial  table for the CRC algorythim.
        //This saves 768  bytes of code space.

/************************************************************************************************
*************/
            #ifdef __cplusplus              // Provide C++ Compatibility
        }

            #endif
        #endif
```