ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"CONTROL DE MOTORES SIN ESCOBILLAS (BLDC) Y SIN SENSOR USANDO EL MICROCONTROLADOR ARM CORTEX3 CON 32 BITS DE LPCXPRESSO"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

INGENIERO EN ELECTRICIDAD

ESPECIALIDAD EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

Jacinto Javier Palma Avellán

Rony Eduardo Chango Murillo

GUAYAQUIL – ECUADOR

AÑO 2013

AGRADECIMIENTO

A mis padres por ser el pilar fundamental en mi vida y apoyarme en todo instante, mostrándome el camino del bien.

A todos mis profesores no solo de la carrera sino de toda la vida, mil gracias porque de alguna manera forman parte de lo que ahora soy.

Especialmente al Ing. Carlos Valdivieso por saber guiarnos en el desarrollo de esta tesina y su apoyo incondicional en la ejecución de nuestro proyecto.

Rony Eduardo Chango Murillo.

A mis padres y mis hermanos por ser las personas que siempre me apoyaron en todo el tiempo que duro mi carrera y ser los pilares fundamentales en mi vida.

A mis compañeros y profesores que en todo instante formaron parte importante en mi formación profesional y aportaron en gran conocimiento y experiencia en la misma.

Jacinto Javier Palma Avellán

DEDICATORIA

La familia es una de las joyas más preciadas que uno puede tener, sin la familia uno no puede conseguir la fuerza necesaria para lograr las metas.

Este documento es un esfuerzo grande que involucra a muchas personas cercanas a mí. Es por eso que dedico esta tesina a mis padres, hermanos, esposa, tíos, primos y principalmente a mis hijas Scarleth y Chiquinquirá, que son el motor que me obliga a funcionar y ser cada día mejor.

Rony Eduardo Chango Murillo.

Primero a Jehová porque sin él nada se puede realizar ya que no pertenece al hombre ni siquiera el paso que da y también a mi familia, porque sin ellos no hubiese podido avanzar ya que cada uno de los integrantes de mi familia pusieron mucho esfuerzo a la distancia para que yo concluya con mi objetivo de ser profesional.

Jacinto Javier Palma Avellán

TRIBUNAL DE SUSTENTACIÓN

Ing. Carlos Valdivieso

PROFESOR DEL SEMINARIO DE GRADUACIÓN

Patricia X. Chávez Burbano M.S.E.E

PROFESOR DELEGADO POR LA UNIDAD ACADÉMICA

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesina, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral".

(Reglamento de Graduación de la ESPOL).

Rony Eduardo Chango Murillo **Jacinto Javier Palma Avellán**

RESUMEN

La finalidad de este proyecto es mostrar el uso de técnicas utilizadas en el control de motores BLDC, con programación a través del puerto USB del computador hacia las tarjetas LPC1114 y LPC1769 las cuales trabajan en conjunto con la tarjeta ‘‘LPCXPRESSO Motor Control’’ que permite el control del Motor y la verificación de los recursos disponibles en la comunicación RS232, SPI, I2C, para poder desarrollar proyectos variados.

Para la realización del proyecto se utiliza el ‘‘Kit Motor Control’’, que permite tener una comunicación con las tarjetas LPC1114 y LPC1769 además de un conjunto de botoneras, las cuales se emplean para la transmisión correspondiente de la instrucción.

Este trabajo nos permite conocer la importancia del control del Motor BLDC y su uso en la industria electrónica, en especial en el aeromodelismo donde permite el óptimo funcionamiento en los prototipos. Además permite el estudio de los microcontroladores avanzados y su implementación en hardware y software necesarios para la ejecución de varios proyectos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

GLOSARIO

ADC Conversor analógico digital o ADC es un circuito electrónico que convierte una señal analógica en digital. Se utiliza en equipos electrónicos como ordenadores o computadoras, grabadores digitales de sonido y de vídeo, equipos de comunicaciones.

BLAC Motor de CA sin escobillas.

BLDC Motor de CC sin escobillas.

CA Corriente Alterna.

CC Corriente Continua.

CODE RED Desarrollador del programa LPCXPRESSO.

DEBUGGER Depurador.

EFECTO HALL Efecto que induce un voltaje sobre dos placas, las cuales atraviesan un campo magnético.

FOC Control de Campo Vectorial.

IDE Entorno de desarrollo integrado.

JTAG "Joint Test Action Group" (JTAG) herramienta necesaria para bajar/depurar las aplicaciones.

LPCXPRESSO Conjunto de herramientas de desarrollo de software para microcontroladores de la familia NXP-LPC.

MHz Mega hertzios.

NXP Fabricante de los módulos LPCXPRESSO.

OTG **“**On The GO” Es una tecnología que permite a equipos que no son una PC, conectarse Directamente a otros dispositivos por USB.

PWM Modulación por Ancho de Pulso.

PCB Tarjeta de circuito impreso o PCB (Printed Circuit Board). Es un sistema de interconexión de componentes electrónicos (resistencias, diodos, transistores, circuitos integrados, bobinas, etc.), mediante pistas de material conductor generalmente cobre.

SPI Comunicación Serial Sincrónica.

SRAM Static Random Access Memory, o Memoria Estática de Acceso Aleatorio es un tipo de memoria basada en semiconductores la cual es capaz de mantener los datos, mientras esté alimentada, sin necesidad de circuito de refresco. Sin embargo, sí son memorias volátiles, es decir que pierden la información si se les interrumpe la alimentación eléctrica.

UART "Universal Asynchronous Receiver-Transmitter".

Transmisor receptor asíncrono universal. Circuito integrado conectado al bus en paralelo de un computador, utilizado para comunicaciones seriales. El UART hace la conversión entre señales seriales y señales paralelas, brinda temporización de transmisiones y realiza el búfering de los datos que se envían hacia y desde el computador.

INTRODUCCIÓN

El presente proyecto tiene como finalidad demostrar el funcionamiento de controles para motores de corriente continua sin escobillas y sin sensores (BLDC), estos motores en los actuales momentos están ganando popularidad rápidamente gracias a los diversos usos que se puede dar en proyectos electrónicos, en el caso de los motores BLDC con sensores la aplicación está dirigida a las implementaciones en automatización industrial y en el caso de los motores BLDC sin sensores el uso específico se lo da en aeromodelismo.

Los motores BLDC se utilizan en industrias tales como electrodomésticos, automoción, aeroespacial, de consumo, equipo médico, automatización industrial y de instrumentación; además, los motores BLDC son de tipo síncrono, es decir, el campo magnético generado por el estator y el campo magnético generado por el rotor giran a la misma frecuencia.

Para la ejecución de este proyecto nos basamos en un ejemplo que la empresa fabricante de las tarjetas había programado, realizamos los cambios pertinentes para adecuarlo a las necesidades de nuestro proyecto. La programación del proyecto fue realizada en lenguaje C, para después ser cargado a la tarjeta respectiva y poder verificar su funcionamiento.

CAPÍTULO 1

DESCRIPCIÓN GENERAL DEL PROYECTO

En el capítulo 1 se dará una descripción general del proyecto, exponiendo los antecedentes, motivación, objetivos y limitaciones.

1.1. Antecedentes

En el siglo XVIII Thomas Newcomen y John Calley [1] diseñaron el motor a vapor, el cual fue fundamental para el desarrollo de aquella época. Con este descubrimiento se dio paso al desarrollo de nuevas fábricas y al desarrollo económico de países como Inglaterra y Alemania, este motor fue basado en una bomba que usaba el vacío creado por el vapor condensado para aspirar el agua de las minas, desde entonces se han desarrollado un sin números de técnicas para mejorar la eficiencia y velocidad de los motores.

Más tarde, en 1832 William Sturgeon [1] diseño un motor eléctrico mucho más eficiente pero con el problema de conmutar la corriente por el magneto, para ese entonces ya se había descubierto que la electricidad podía mover los motores. En 1885 cuando se construye el primer motor a gasolina, el cual era potente aunque poco eficiente, a partir de ese momento comienzan los estudios para hacer motores más potentes y veloces.

Actualmente, el mundo se encuentra con una escasez e inminente extinción del petróleo, y un alto índice de contaminación derivados del incremento de automotores, por lo que se hace indispensable pensar en nuevas formas de alimentar los motores y se retoma el diseño basado en eficiencia más que en la potencia.

Las investigaciones recientes se centran en mejorar la eficiencia de los motores eléctricos, utilizando nuevos tipos de motores y configuraciones de control. Es por esto que entre los motores eléctricos con mayor aceptación para el desarrollo de sistemas eléctricos se encuentran los motores BLDC (motores CC sin escobillas), que ofrecen muchas más ventajas frente a otros motores eléctricos.

El interés en la utilización de motores BLDC es el aumento de la eficiencia operativa en las empresas ya que estos motores no producen rozamientos, aumentan rendimiento, y requieren menos mantenimiento, estos motores realizan el cambio de polaridad en el rotor. Un ejemplo de motor BLDC es el que se usa en pequeños aparatos electrónicos de baja potencia, como los lectores de CD-ROM, ventiladores, etc. Su mecanismo es sencillo y se basa en la conmutación mecánica por un cambio de polaridad electrónica sin contacto; la espira sólo es impulsada cuando el polo es el correcto, y cuando no lo es el sistema electrónico corta el suministro de corriente. Para detectar la posición de la espira del rotor se utiliza la detección de un campo magnético. Este sistema electrónico puede informar de la velocidad de giro, si el motor está parado, o cortar la corriente si se detiene para que no se queme. Estos motores no giran al revés al cambiarles la polaridad, requieren que se crucen dos conductores del sistema electrónico.

1.2. Motivación

Tras una ardua investigación encontramos que los motores BLDC son motores más económicos y más eficientes ya que no existe desgaste por rozamiento, por tal motivo emprendimos en un trabajo y

aprendizaje más a fondo en el desarrollo de motores BLDC para aplicaciones electrónicas.

Sabiendo que cada vez existe más demanda en aplicaciones del ámbito tecnológico y tomando en cuenta que siendo una tecnología barata y eficiente nos disponemos a producir ejemplares a magnitudes mucho mayores en el área de automatización industrial, dado que en los actuales momentos la mano de obra es poco requerida en industrias altamente automatizadas, puesto que sistemas como los desarrollados en este proyecto cubren las expectativas de los empresarios haciendo más eficiente y productiva a sus empresas, es por esto que viendo una gran oportunidad de generar proyectos y negocios nos motivamos a realizar el proyecto explicado a lo largo de este documento.

En el caso de los motores BLDC sin sensores como sus aplicaciones están direccionadas al aeromodelismo, las fuerzas armadas lo utilizan para realizar prototipos de aviones y poder diseñar nuevos modelos. Como es de nuestro interés poder aportar con ideas claras y funcionales a nuestra patria nos involucramos más a fondo en el

estudio de motores BLDC sin sensores, para el desarrollo de estos proyectos.

1.3. Identificación del problema

Los motores BLDC sin sensores son motores que están direccionados para trabajos menos complicados, dado que a estos motores solo se les deja las herramientas netamente necesarias, es decir no poseen sensores, pues esto haría más pesado al motor y lo que se pretende en que este motor sea lo más liviano posible. Pero a pesar de su sencillo hardware, su programación es más compleja en comparación a los motores BLDC con sensores, debido a los varios parámetros que se deben considerar.

Los motores BLDC sin sensores en la industria Ecuatoriana son poco investigados, puesto que el aeromodelismo solo lo efectúan las fuerzas armadas, por lo tanto este es nuestro mayor problema. Sin embargo nuestra intención es poder cubrir las necesidades de nuestros militares, resolviendo o facilitando el trabajo para ellos.

1.4. Objetivos

El objetivo principal de nuestro proyecto es implementar un circuito con la programación respectiva que pueda controlar un motor BLDC para cualquier instrucción que se le indique al mismo.

Para alcanzar este objetivo debemos diferenciar entre el control de un motor BLDC sin sensores y un motor BLDC con sensores, los cuales trabajan sobre la misma plataforma, pero requieren de programación distinta; el objetivo de los motores sin sensores es la investigación y los motores con sensores son para realizar trabajos más eficientes dentro de las empresas, Además, por cuestiones netamente académicas, el primer análisis lo hacemos en motores BLDC con sensores, para comparar su operación con los motores sin sensores.

1.5. Limitaciones

Trabajamos con un microcontrolador poco empleado en el país, como es la tarjeta LPC1769, lo cual implica una ardua investigación y complica el aprendizaje de los bloques, programación, componentes, dispositivos y arquitectura que comprenden cada uno de las tarjetas.

El trabajar con el motor BLDC representa una limitación, ya que requiere del desarrollo de software para su óptimo funcionamiento, mientras que la tarjeta LPC1114 es limitada en el número de pines disponibles para la interacción con la tarjeta controladora “MOTOR CONTROL BOARD”.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

En este capítulo se describe el funcionamiento y características teóricas de cada una de las partes que estarán implementadas en este proyecto como ciertas teorías de necesidad para la programación, mostrando las ventajas y desventajas de sus componentes.

2.1. Requerimientos para la aplicación del Proyecto

El proyecto se lo divide en dos partes: Software y Hardware, ambos indispensables para la elaboración del proyecto. El software para programar el microcontrolador es el LPCXPRESSO, este posee un conjunto de herramientas que son necesarias para desarrollar soluciones de alta calidad, haciendo efectiva en tiempo y costo. Además posee dos compiladores para los lenguajes ensamblador y C; en este proyecto se utilizó el lenguaje C para la creación del código.

2.2. Herramienta de Software

LPCXPRESSO 2.0_2.64 es una herramienta de desarrollo para programar tarjetas creadas por la compañía LPCXPRESSO en la cual la programación se la realiza en el lenguaje C/C++. Este software nos permite verificar línea por línea el desarrollo del programa y se lo puede ejecutar en sistemas operativos Windows y Linux. LPCXPRESSO puede generar aplicaciones que contengan hasta 128 KB de código, este ha sido desarrollado por Code Red junto a NXP. El mismo incluye un entorno de Eclipse específicamente adaptado para interactuar con la tarjeta y utiliza algunos conceptos comunes a otros programas de desarrollo por lo que vamos a ver algunos detalles. [2][3].

Tenemos también un espacio de trabajo donde se encuentran nuestros proyectos y almacena todas las configuraciones del entorno; y un proyecto que puede ser de dos tipos: biblioteca estática o una aplicación ejecutable. Aquí se contiene archivos de código fuente (.c), encabezados (.h) y cualquier otro archivo que sea necesario para el buen funcionamiento del programa a desarrollar [3], ambos se ilustran en la figura 2.1.

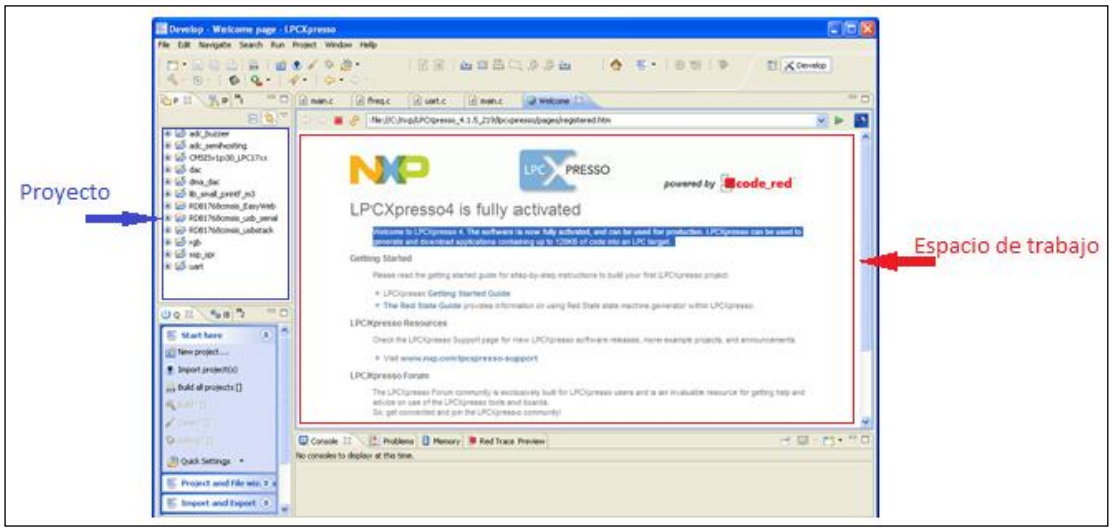

Figura 2.1: Software LPCXPRESSO [4].

2.3. Herramientas de Hardware.

Algunas de las características de esta familia de microcontroladores (LPC11XX) son las interfaces y periféricos que incorporan, y el bajo consumo que el dispositivo presenta. La LPC1769 está compuesta por: el programador JTAG, y un zócalo, que nos permite realizar las diversas funciones, pues en el mismo se encuentra un microcontrolador con un cristal de 2 MHz, con el que podremos trabajar con otros microcontroladores, que serán nuestro complemento en proyectos varios, además poder realizar simulaciones; ambos montados en un pequeño PCB [5].

La tarjeta LPC1769 es un ARM Cortex M3 basado en microcontroladores para sistemas embebidos con un alto nivel de integración, incorpora un procesador de 3 etapas y usa una arquitectura Harvard con instrucción local separada, buses de datos y un bus para periféricos. Alguna de sus características principales es que opera a tasas de trasmisión de hasta 120MHz, la memoria flash tiene una capacidad de 512 Kb, 64Kb de memoria SRAM, posee interfaces de comunicaciones: SPI, RS232, I2C, Ethernet y cuenta con comunicación USB OTG (On the go) [6].

Entre las aplicaciones de la tarjeta LPC1769 tenemos que desempeñan un papel importante en áreas como: iluminación, redes industriales, sistemas de alarma, electrodomésticos de línea blanca, control de motores; los elementos que constituyen la tarjeta LPC1769 se ilustran en la figura 2.2 [7] [8].

La tarjeta LPC1114 es una variante de las tarjetas de la familia de NXP la cual sirve para desarrollar proyectos al igual que la tarjeta LPC1769 con la diferencia es que es basada en ARM Cortex M3, esta se ilustra en la Figura 2.3 [6].

Algunas de sus características principales es que cuenta con un microcontrolador NXP Cortex-M0 (hasta 50 MHz), 32 Kb Flash y 8 Kb SRAM, una I2C de alta velocidad, una UART, dos SSP/SPI, cuatro temporizadores, ADC de 10 bits y 42 puertos de entrada/salida, 50 terminales en conexión con todos los pines del microcontrolador NXP LPC1114 Cortex-M0 (42 del microcontrolador y 8 de alimentación).

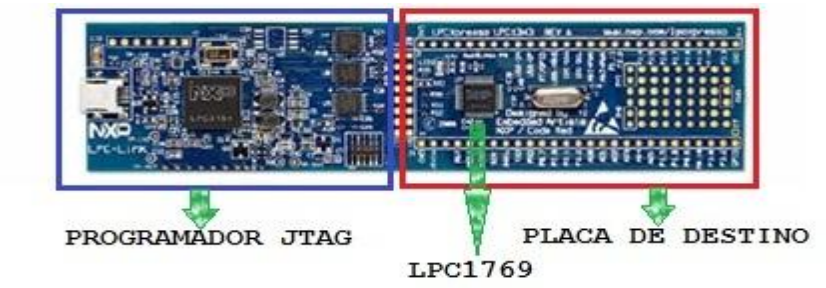

Figura 2.2: Tarjeta de Desarrollo LPC1769 [2].

Figura 2.3: Tarjeta de Desarrollo LPC1114 [9].

2.4. Kit de Control de Motor

Se trata de una tarjeta que incorpora controladores de baja tensión para efectuar trabajos de motores BLDC o BLAC por medio de comandos enviados desde la tarjeta LPCXPRESSO que se esté

utilizando, LPC1114 o LPC1769. El Control de Motor LPCXpresso tiene una estructura tal como se indica en la figura 2.4 [1].

El margen derecho de la tarjeta tiene los elementos necesarios para generar la potencia insertada al motor, conducción de las fases del motor y demás características de electrónica de potencia; y el izquierdo tiene control con zócalos para diferentes tarjetas de LPCXpresso, así como una toma de control del procesador PLCC434. Además tiene incorporado un joystick que permite enviar instrucciones, lo que puede servir como una interfaz de usuario para el sistema, cuando no se tiene dispositivos externos que envíen dicha información, esta información puede ser enviada por diferentes tipos de comunicación [1][2][10].

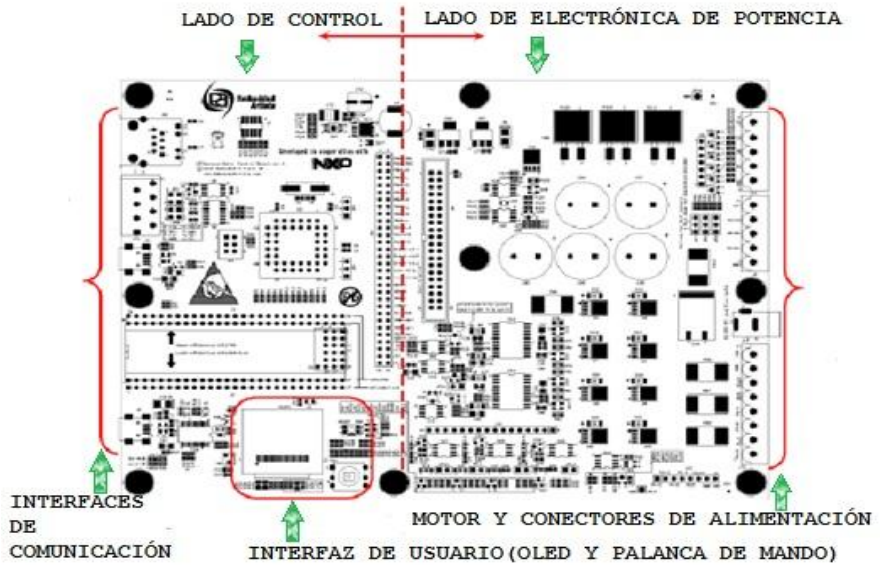

Figura 2.4: Tarjeta Control Motor [1].

2.5 Motores sin escobillas (BLDC)

Estos motores eléctricos carecen de escobillas, o elementos que hacen contacto con el colector. Son clasificados dentro de los motores eléctricos de corriente continua por llevar imanes permanentes [11], un ejemplo de este tipo de motores se ilustra en la figura 2.5.

Normalmente este tipo de motores se usan en pequeños objetos, como ventiladores de ordenadores, motores de reproductores de CDs, debido a su pequeño tamaño y bajo consumo, lo que permite su uso en estos campos; Además, funcionan con una fuente de corriente continua, como pilas o una batería recargable, por lo que son muy usados en campos como el aeromodelismo, o aparatos de radiocontrol [11] [12].

Figura 2.5: Motor sin escobilla (BLDC).

2.5.1 Configuración

Los motores BLDC a diferencia de los motores CC convencionales no tienen sus bobinas en el rotor, sino en el estator y así el conmutador mecánico es sustituido por uno electrónico. Todos los motores BLDC tienen 3 bobinas en conexión estrella, pues esto permite tener el punto principal conectado internamente y tener solo tres cables en el exterior, cada uno perteneciente a una bobina [11][12].

Existen 3 imanes que se encuentran en el rotor y tres sensores llamados “sensores de efecto HALL” que se encuentran en el estator, los cuales permiten medir la posición angular del rotor, esto ocurre cada vez que los polos magnéticos del rotor pasan cerca de los sensores HALL, creando una señal de estado bajo o alto que indica que el polo norte o sur está pasando en ese instante. En la figura 2.6 se muestra la conexión de las bobinas de motor BLDC y como están distribuidas se ilustra en la figura 2.7 [12].

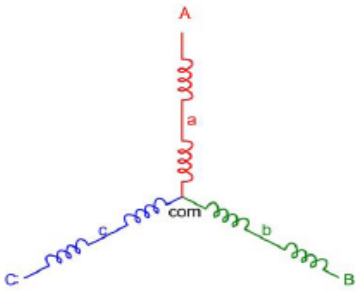

Figura 2.6: Partes de los devanados.

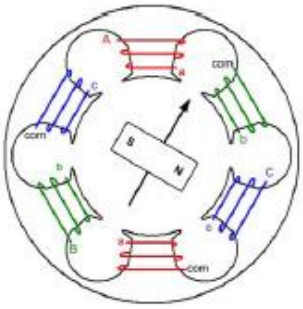

Figura 2.7: Distribución física de estas partes.

2.5.2 Funcionamiento

Los imanes que contiene el motor BLDC son atraídos por la polaridad del campo magnético generado por las bobinas, dicho campo magnético se crea por una frecuencia de pulsos que circulan por las bobinas; siendo la velocidad del motor función de esta frecuencia de pulsos. En cuanto a las ventajas de estos motores, hay que destacar la mayor eficiencia debido a que existe una menor pérdida de energía en forma de calor, esto es así, porque en su funcionamiento no existen escobillas rozando continuamente con los elementos en movimiento, como pasa en otros motores eléctricos. De hecho, el no llevar escobillas, es también el causante de otras ventajas como la mayor potencia, el menor consumo, menor ruido y el menor tamaño de estos motores frente a los demás. Todas estas características son fundamentales en estos motores. Así, son los motores utilizados en radiocontrol, donde el tamaño y peso de los servomotores

empleados juegan un papel esencial, sin embargo al no haber contacto entre ciertas piezas, estos motores tienen más dificultad para disipar calor. De igual manera las inercias creadas serán mayores y por lo tanto los cambios de velocidad del motor son más difíciles [12][13].

Los motores sin escobillas tienen muchas ventajas por sobre los motores con escobillas entre ellas tenemos: mayor rendimiento (mayor duración de las baterías para la misma potencia), mayor eficiencia (menos perdida por calor), menor peso para la misma potencia, conmutación electrónica basada en sensores de posición de efecto Hall, requieren menos mantenimiento al no tener escobillas, relación velocidad/par motor es casi una constante, mayor potencia para el mismo tamaño, mejor disipación de calor, rango de velocidad elevado al no tener limitación mecánica y menor ruido electrónico (menos interferencias en otros circuitos)[13][14][15].

Entre las desventajas de los BLDC tenemos: tiene un mayor costo de construcción, el control es mediante un circuito caro y

complejo, siempre hace falta un control electrónico para que funcione, que a veces duplica el costo [13].

2.5.3 Motores BLDC con Sensores de efecto Hall

Cuando existe la presencia de una corriente que pasa través de un sensor Hall y el mismo se aproxima a un campo magnético perpendicular que es generado por las bobinas, entonces se crea un voltaje saliente que será proporcional al producto de la fuerza del campo magnético y de la corriente. Con esto se censa la posición del rotor. Los sensores se usan para que puedan conmutar las tres bobinas del motor, lo cual sucede de acuerdo a la posición de los polos del imán del rotor [14].

Un motor BLDC de dos polos con 3 sensores que tienen un desfase de 120º, uno para cada bobina del motor, y deberán tener una resolución posicional de 6 pulsos por vuelta, pero se debe considerar que para el caso de los motores BLDC multipolares, esta resolución aumenta. En la figura 2.8 se muestran los elementos para el control de un motor BLDC con sensor de efecto Hall [12][15].

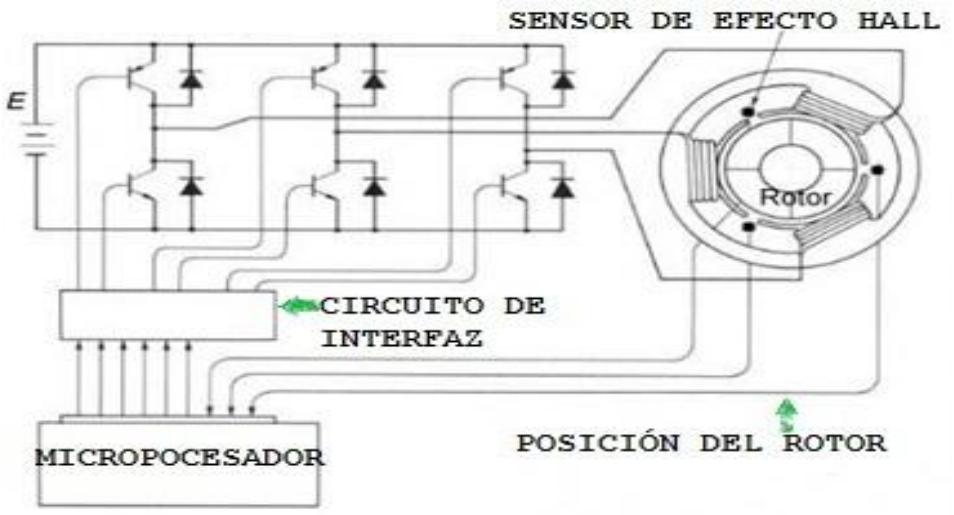

Figura 2.8: Elementos para el control de un Motor BLDC con sensores de efecto Hall [16].

2.5.4 Motores BLDC sin Sensores de efecto Hall.

En este tipo de motores, al carecer de sensores su posición se la determina midiendo la fuerza contra electromotriz (FEM) en las bobinas del rotor. Al igual que en un motor de C.A., el voltaje en las bobinas es sinusoidal, pero sobre una conmutación entera la salida aparece trapezoidal debido a la salida de la C.C. del regulador. Estos reguladores emplean comparadores para determinarse cuando la fase de salida debe ser de avanzada, mientras que reguladores más avanzados emplean microntroladores para manejar la aceleración, la velocidad del control y eficiencia [12][15].

Los reguladores que detectan la posición del rotor mediante la medición de la fuerza contra electromotriz sobre las bobinas tienen desafíos en iniciar el movimiento porque no se produce ningún FEM de vuelta cuando el rotor es inmóvil. Esto es lograda generalmente comenzando la rotación a partir de una fase arbitraria, y después cambiando a la fase correcta. Esto puede hacer al motor funcionar brevemente al revés, agregando aún más complejidad a la secuencia de lanzamiento. En la figura 2.9 se muestra la configuración para el control de un motor BLDC sin sensor [16].

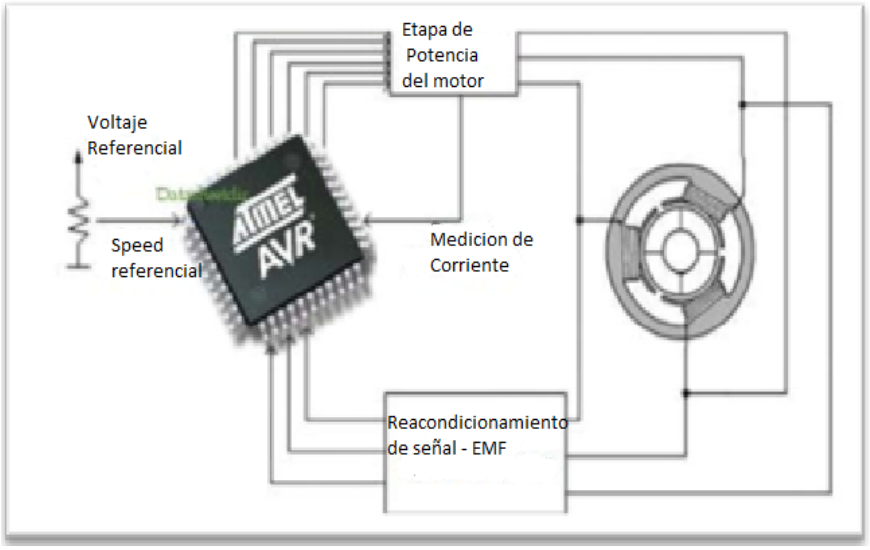

Figura 2.9: Motor BLDC sin sensores [16].

2.5.5 Métodos de conmutación de motores sin escobillas

Para el control de los motores con sensores tenemos la clasificación según el algoritmo de control utilizado. Los más usados son los siguientes, en orden creciente de eficiencia y complejidad [16]:

- Conmutación trapezoidal o "six step mode"
- Conmutación sinusoidal
- Control vectorial

Conmutación trapezoidal

Para el control de motores sin escobillas este método es el más simple, ya que se controla la corriente que circula por los terminales del motor, esto quiere decir que puede determinar cuáles son los dos terminales habilitados simultáneamente y manteniendo el tercer terminal desconectado. Teniendo como resultado que sucesivamente se va alternando el par de terminales, hasta completar las seis combinaciones posibles. Las 6 posibles direcciones de las corrientes se muestran en la figura 2.10 [15].

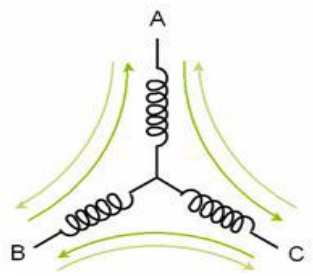

Figura 2.10: Esquema de los seis posibles caminos de circulación de corriente en el control trapezoidal [15].

Este esquema es el mismo tanto para motores con sensores de efecto Hall, como para motores sin sensores, donde se usa para conocer la posición del rotor. Por su facilidad de implementación es el método más usado en motores pequeños. Pero a pesar de su sencillez y facilidad de implementación, se tiene un problema inherente a la conmutación del vector de corrientes que es un rizado en el torque de salida [15]. Esto acurre en aplicaciones donde se requieren fuerzas uniformes o bajas velocidades. En la figura 2.11 se muestran las corrientes por cada una de las fases, la secuencia de conmutación y el torque.

Conmutación sinusoidal

En esta conmutación sinusoidal se controla la posición del rotor de forma continua, es por esto que es vista como un control

más avanzado y exacto que el trapezoidal. Para conseguir este control es necesario inyectar una corriente sinusoidal a las tres bobinas simultáneamente con un desfase de 120º. La fase de estas corrientes se escoge de forma que el vector de corrientes resultante siempre este en cuadratura con la orientación del rotor y tenga un valor constante [18].

En este procedimiento se obtiene un control más preciso que en la conmutación trapezoidal. Pero se debe de considerar que para poder generar la modulación sinusoidal es necesaria una medida precisa de la posición del rotor, esto significa se requiere de un encoder de alta resolución, puesto que difícilmente se logra con sensores de efecto Hall [18].

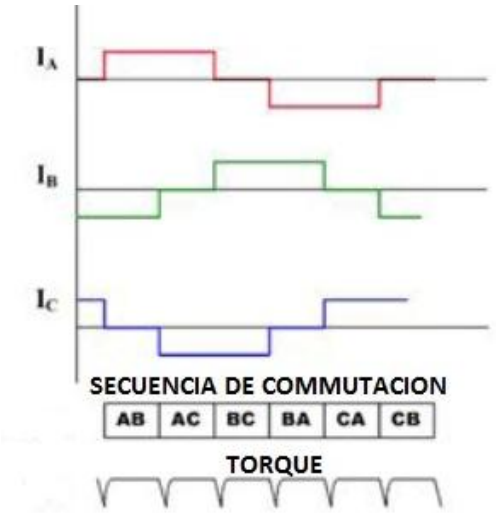

Figura 2.11: Corrientes en las bobinas y torque del motor [5].

Cuando se tiene velocidades bajas este método de control presenta el mejor desempeño en eficiencia y suavidad del torque, y con altas frecuencias no responde de la mejor manera, ya que tiene que procesar señales sinusoidales de frecuencias altas y los controladores PI usados para generar estas señales tienen una respuesta limitada en ganancia y frecuencia. Por lo tanto cuando la frecuencia es suficientemente alta, la eficiencia decrece y el error aumenta, tendiendo a un punto de cero torque [15].

Control vectorial

Para entender un poco más el control vectorial es necesario analizar que en la conmutación sinusoidal el problema principal radica en que intenta controlar directamente las corrientes que circulan por el motor, las cuales son intrínsecamente variantes en el tiempo. Con esto se tiene que al aumentar la velocidad del motor la frecuencia de las corrientes circulantes son mayores, por lo que empiezan a aparecer los problemas.

Para solucionar estos problemas el Control vectorial analiza y determina el vector de corrientes directamente, esto lo hace en

un espacio de referencia ortogonal y rotacional, a este espacio se lo conoce como “espacio d-q” (directa - cuadratura). Este espacio de referencia se encuentra normalmente alineado con en el rotor, permitiendo el control del flujo y del par del motor permitiendo se realice de forma independiente. Además la componente directa permite controlar el flujo y la componente en cuadratura el par [15].

Ahora, se debe de considerar que para este fin es necesario que no solamente tenga una buena medición de la orientación del rotor, sino un tratamiento matemático previo de las señales para transformarlas del marco trifásico estático de los bobinados en el estator, al marco rotacional d-q del rotor. Por lo tanto con esto conseguimos el mejor control entre todos los analizados anteriormente, teniendo que presenta la mejor respuesta en todos los rangos de velocidad, pero como consecuencia resulta también ser el más costoso de implementar, lo cual lo hace inadecuado para toda aplicación en la que no sea estrictamente necesario [15].

CAPÍTULO 3

DESARROLLO DEL PROYECTO

A continuación se describen detalladamente cada uno de los ejercicios los cuales fueron parte importante en la realización del proyecto final, para así obtener el control preciso sobre el movimiento del motor BLDC, utilizando las tarjetas LPC1769 y LPC1114.

Podemos controlar al motor de dos maneras diferentes; la primera cambiando la programación de la tarjeta LPC1114 para acoplarla a la tarjeta LPC1769 y trabajar directamente con el control; en el otro caso tenemos que se trabaja con las dos tarjetas al mismo tiempo y se acopla a la tarjeta controladora con ciertos ajustes en software y hardware para un correcto funcionamiento. En nuestro caso se utilizó el segundo método, teniendo un funcionamiento excelente.

3.1 Encendido secuenciado de leds

La primera interacción con el software (LPCXPRESSO) y con la tarjeta LPC1769 fue la elaboración de un código para familiarizarnos con la manipulación y configuración de entradas y salidas de la tarjeta LPC1769. Nuestro elemento controlador es la tarjeta LPC1769 a la cual se le conectará una botonera la misma que enviará señales de voltaje que serán procesadas por la tarjeta LPC1769, esta enviará las diferentes señales de voltaje a través del puerto 2 para encender cada uno de los LEDs en el sentido indicado dependiendo de la señal enviada por la botonera, esto se ilustra en la figura 3.1 la cual muestra el diagrama de bloques del sistema.

Figura 3.1: Diagrama de bloques del encendido secuenciado de LEDs.

El programa completo en lenguaje C se encuentra en el Anexo 1, en la figura 3.2 se presenta el diagrama de flujo del sistema, el cual resume las principales acciones que se realizan durante la ejecución de este ejercicio. El primer paso es básicamente una inicialización de variables y definición de puertos a utilizar, declarándolos como entrada o salida para luego inicializar los valores en cada uno de sus pines. Esto se

ejecuta sólo una vez al momento de energizar el sistema, luego se ingresa a un lazo infinito en el cual se pregunta si el pin 8 del puerto 2 recibe 0 voltios como consecuencia de tener presionada la botonera, de ocurrir esto los LEDs se encenderán en sentido horario, caso contrario los LEDs se encenderán en sentido anti horario, el código ha sido realizado para seguir esta secuencia indefinidamente. Este lazo infinito nos permite simular el cambio de giro del motor y a su vez que el mismo nunca se apague, dado que teóricamente el motor tiene como mínima velocidad 600 RPM sin apagarse por completo. Esto nos permite comprobar el óptimo funcionamiento del sistema para apreciar y corregir errores en la programación, en síntesis es un ejercicio de introducción para la familiarización del hardware y software.

Aquí se demostró que a través de la comunicación UART para la LPCXpresso 1769 se configuraron las salidas de la misma teniendo como objetivo tener una secuencia de encendidos de los LEDs que en operación normal tiene el sentido horario, mientras que cuando se presiona la botonera invertimos el sentido de encendido de los LEDs comprobando así que el dato enviado por la botonera es receptado por la LPC y se refleja en la salida con el encendido y apagado de los LEDs.

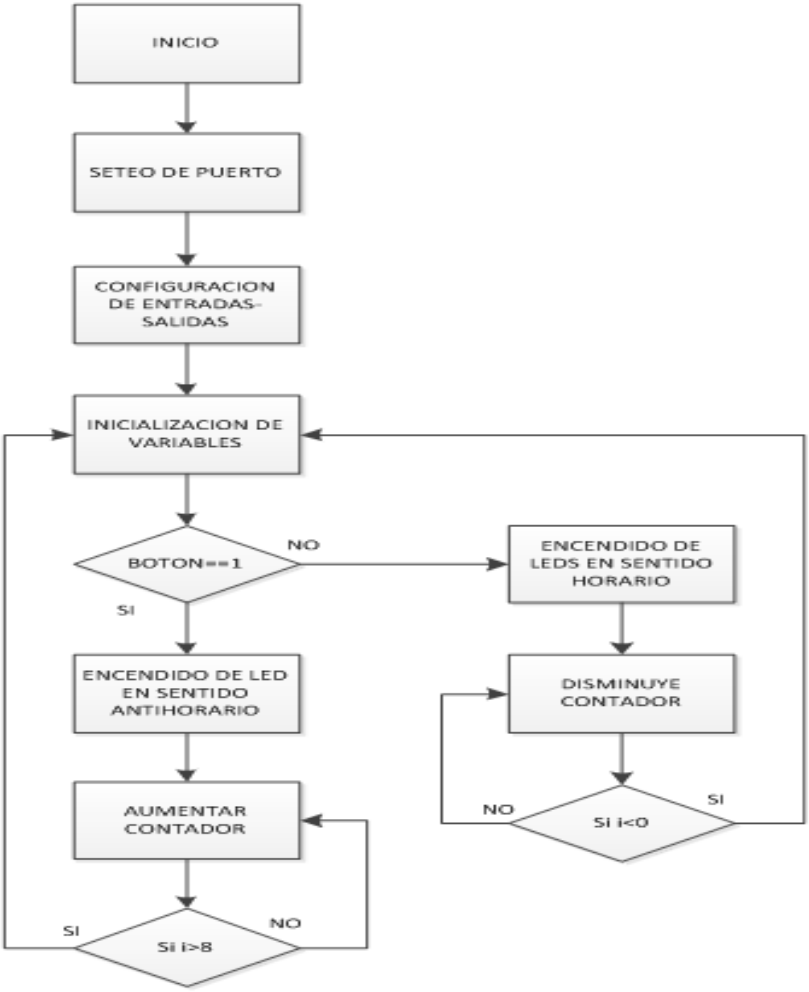

Figura 3.2: Diagrama de flujo del encendido secuenciado de LEDs.

3.2 Control de motor BLDC mediante la tarjeta LPC1114

En el siguiente ejercicio la tarjeta LPC1114 es el controlador para el motor BLDC, ya que controlaremos la velocidad, sentido de giro, podremos parar o dar nueva marcha al motor, este control se lo logra mediante las señales enviadas desde un conjunto de botoneras. El “Motor Control Board” trabaja en conjunto con la tarjeta LPC1114 para

mantener un control constante sobre el motor. Esto se ilustra en la figura 3.3 la cual muestra el diagrama de bloques del sistema.

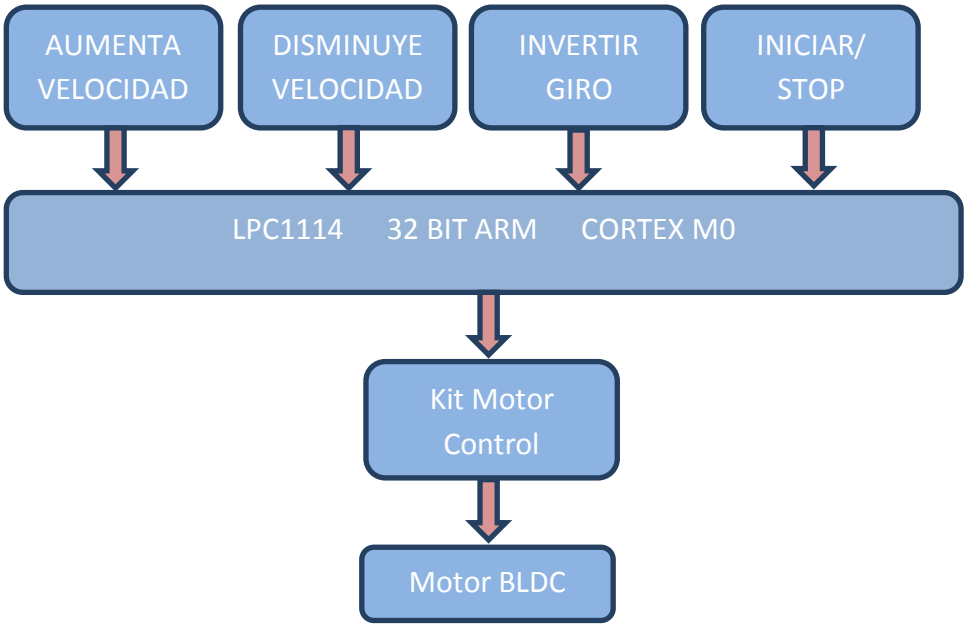

Figura 3.3: Diagrama de bloques del control del motor BLDC mediante la tarjeta LPC1114.

El programa completo en lenguaje C se encuentra en el Anexo 2,en la figura 3.4 se presenta el diagrama de flujo del programa principal main.c del sistema, el cual resume las principales acciones que se realizan durante la ejecución de este ejercicio, primero tenemos la definición de variables y puertos a utilizar, dichos puertos tenemos que declararlos como entrada o salida, luego el programa dependerá de la variable systick_cntr ya que al cumplirse la primera condición de que la variable systick_cntr sea mayor a la variable cnt_5ms, el programa consulta si el motor se encuentra habilitado y sin movimiento para

posteriormente realizar el proceso PID. En el caso de que systick_cntr es mayor a la variable cnt_25ms, el programa nos permite mostrar un pixel en la pequeña pantalla denominada Oled, esto de acuerdo a la velocidad que posee el motor, conforme se va formando la gráfica. Si systick_cntr es mayor a cnt_100ms y el motor se encuentra habilitado y en movimiento, el programa permite aumentar gradualmente la velocidad del motor hasta el límite dado por el setpoint. En el caso de que systick_cntr sea mayor a cnt_1s, se envía la estructura tipo motor. Esto se repite indefinidamente ya que se encuentran en un lazo infinito.

Así podemos concluir que la primera limitación en este ejercicio fue deshabilitar el joystick del “Motor Control Board” y establecer nuevas entradas para la tarjeta LPC1114 y así poder controlar el motor BLDC, esto se logró ya que la LPC1114 tenía entradas disponibles las cuales eran: PIO2.4,PIO3.1, PIO3.2, PIO3.3, en las cuales se conectaron botoneras para poder controlar el motor BLDC, este ejercicio fue muy útil ya que nos permitió tener una idea más clara para la realización de nuestro proyecto, ya que el último paso es comunicar las tarjetas LPC1769,LPC1114, “Motor Control Board”, para poder controlar el motor.

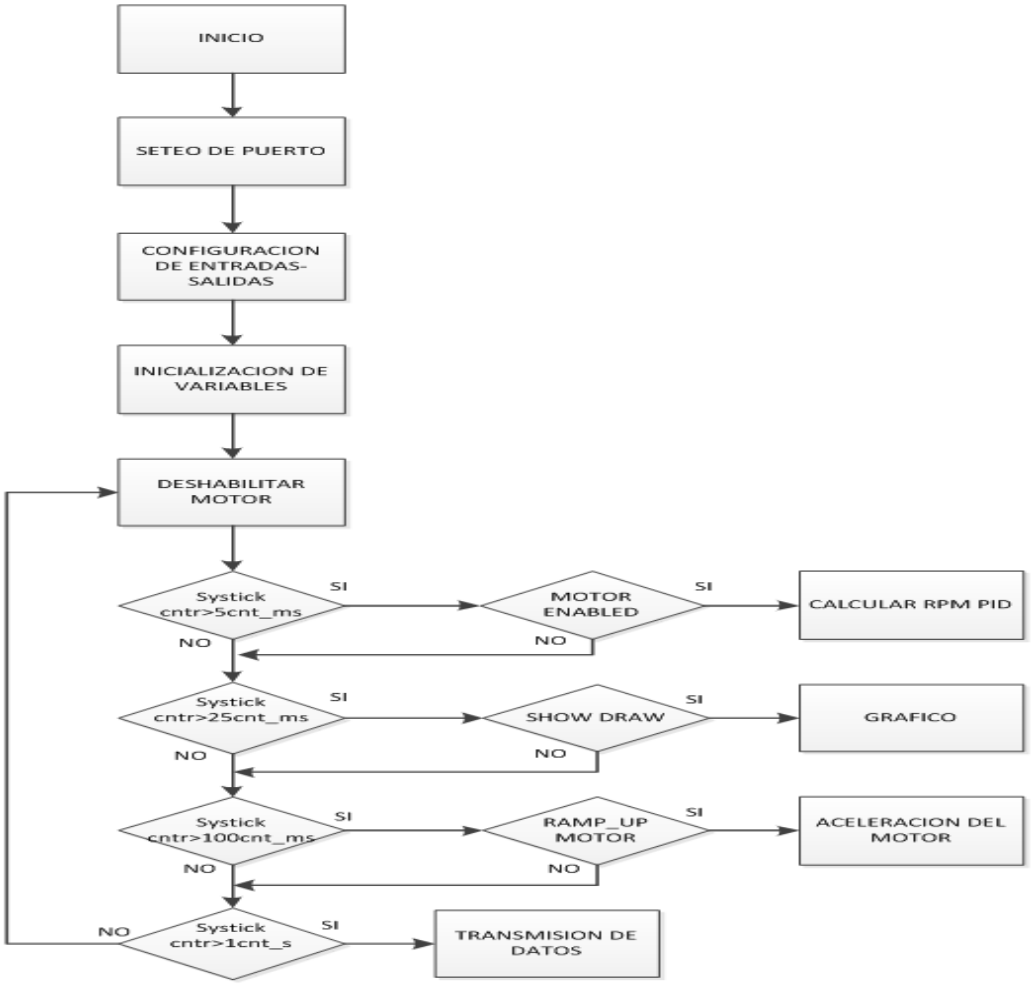

Figura 3.4: Diagrama de flujo del control del motor BLDC mediante la tarjeta LPC1114.

En la figura 3.5 se presenta el diagrama de flujo del programa controlador de movimiento del motor BLDC handlers.c, el cual resume las principales acciones que se realizan durante la ejecución de este ejercicio como son: definición de variables a utilizar, inicialización, declaración de puertos como entradas y salidas, para esto utilizamos el puerto 2 y 3. El programa completo en lenguaje C se encuentra en el Anexo 3. El programa se ejecuta de la siguiente manera, cuando el

usuario presiona el botón correspondiente al centro, la LPC efectúa la acción correspondiente y se ve reflejado en el estado del motor, es decir, el motor comenzara a funcionar. En el caso de que el usuario presione el botón arriba, el motor incrementa su velocidad, en el caso de alcanzar su máxima velocidad (4100 rpm) el motor se estabiliza. Si el usuario presiona el botón abajo, el motor disminuye su velocidad hasta alcanzar el mínimo (600 rpm) en cuyo caso se estabiliza .El aumento o disminución es de 50 rpm cada vez que se presiona la botonera ya sea arriba o abajo, el motor se mueve mediante un controlador PID y llega hasta una velocidad máxima de 4100 rpm y mínima de 600 rpm (ósea q nunca se detiene). Cuando el usuario presiona el botón de la derecha, el motor invierte el sentido de giro, esta acción se realiza sin importar la velocidad que tenga el motor. Todo se repite cada vez que se genere una interrupción la cual activa la función void PIOINT2_IRQHandler (void) que contiene todo el procedimiento detallado.

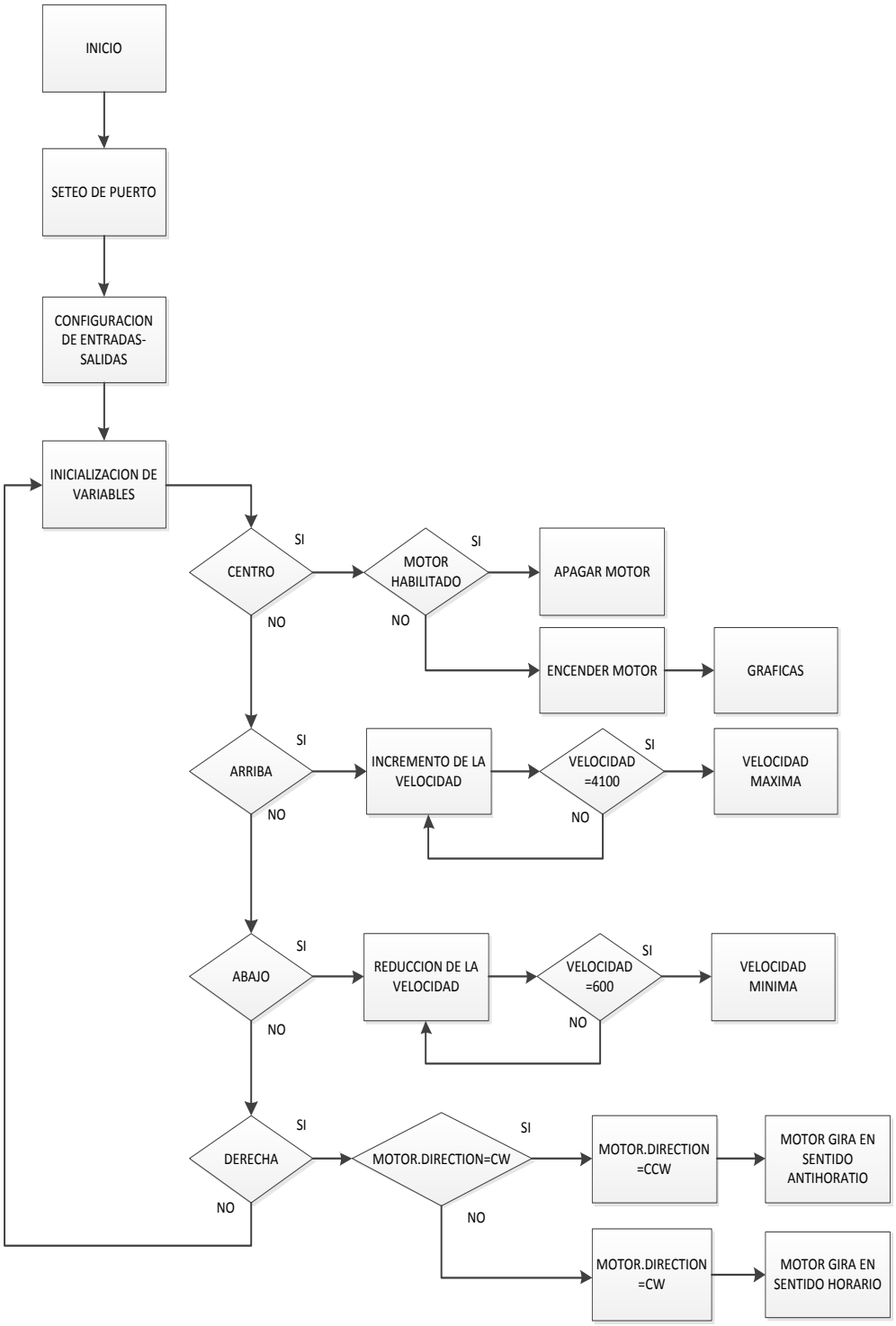

Figura 3.5: Diagrama de Flujo Programa Controlador de Movimiento del Motor BLDC handlers.c.

3.3 Control del motor BLDC mediante las tarjetas LPC1769 Y LPC1114.

En este caso la tarjeta LPC1114 es el dispositivo que se encarga de controlar el motor BLDC, ya sea para controlar su velocidad, sentido de giro, para iniciar o detener su movimiento. La tarjeta LPC1769 es aquella que se encarga de procesar las señales que llegan desde el exterior ya sea de un joystick o de un conjunto de botoneras, dicha tarjeta envía las señales de voltaje para de esta manera lograr alguna respuesta inmediata en el motor BLDC. El “Motor Control Board” trabaja conjuntamente con la tarjeta LPC1114 para mantener un control constante sobre el motor, controlando parámetros básicos como la velocidad máxima, velocidad mínima, temperatura. Esto se ilustra en la figura 3.6 la cual muestra el diagrama de bloques del sistema.

El programa completo en lenguaje C se encuentra en el Anexo 4, en la figura 3.7 se presenta el diagrama de flujo del programa del control del motor BLDC mediante las tarjetas LPC1769 y LPC1114, en el cual se detallan las instrucciones a seguir como son: la inicialización de variables, definición de puertos a utilizar como entradas y salidas, utilizaremos el puerto 2, del cual los pines 0, 1, 2, 3 serán activados como salida y los pines 4, 5, 6, 7 como entrada, una vez hecho esto, si

el usuario presiona el botón asignado al pin P2.4, se activa con un ‘0’ lógico el pin P2.0 lo que activa el motor o detiene su movimiento, si el usuario presiona el botón asignado al pin P2.5, se activa con un ‘0’ lógico el pin P2.1 lo que ocasiona la inversión del sentido de giro, si el usuario presiona el botón asignado al pin P2.6, se activa con un ‘0’ lógico el pin P2.2 lo que ocasiona que se incremente la velocidad de giro del motor, si el usuario presiona el botón asignado al pin P2.7, se activa con un ‘0’ lógico el pin P2.3 lo que ocasiona que disminuya la velocidad de giro del motor, estos pasos se repiten indefinidamente ya que este procedimiento se encuentra en un lazo infinito.

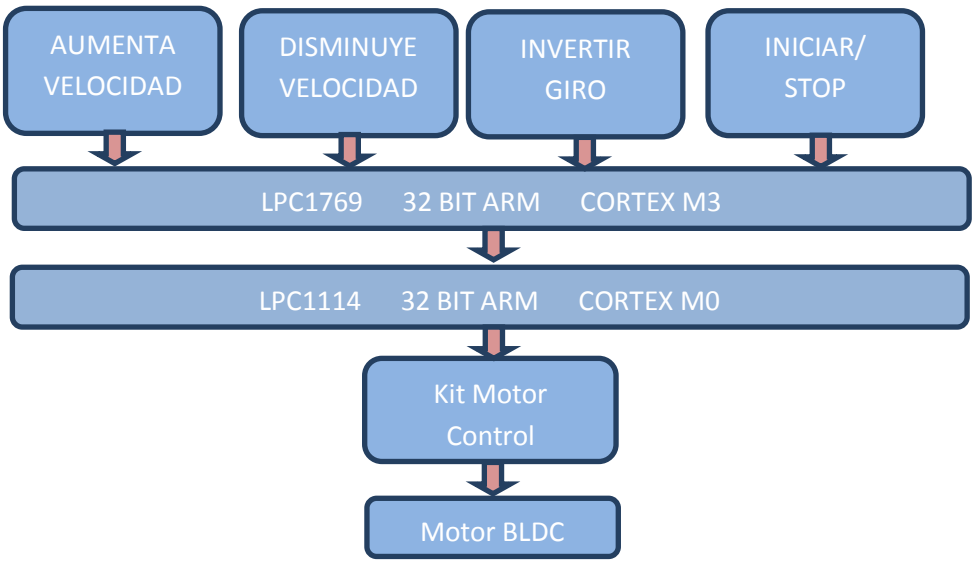

Figura 3.6: Diagrama de bloques del control del motor BLDC mediante las tarjetas LPC1769 y LPC1114.

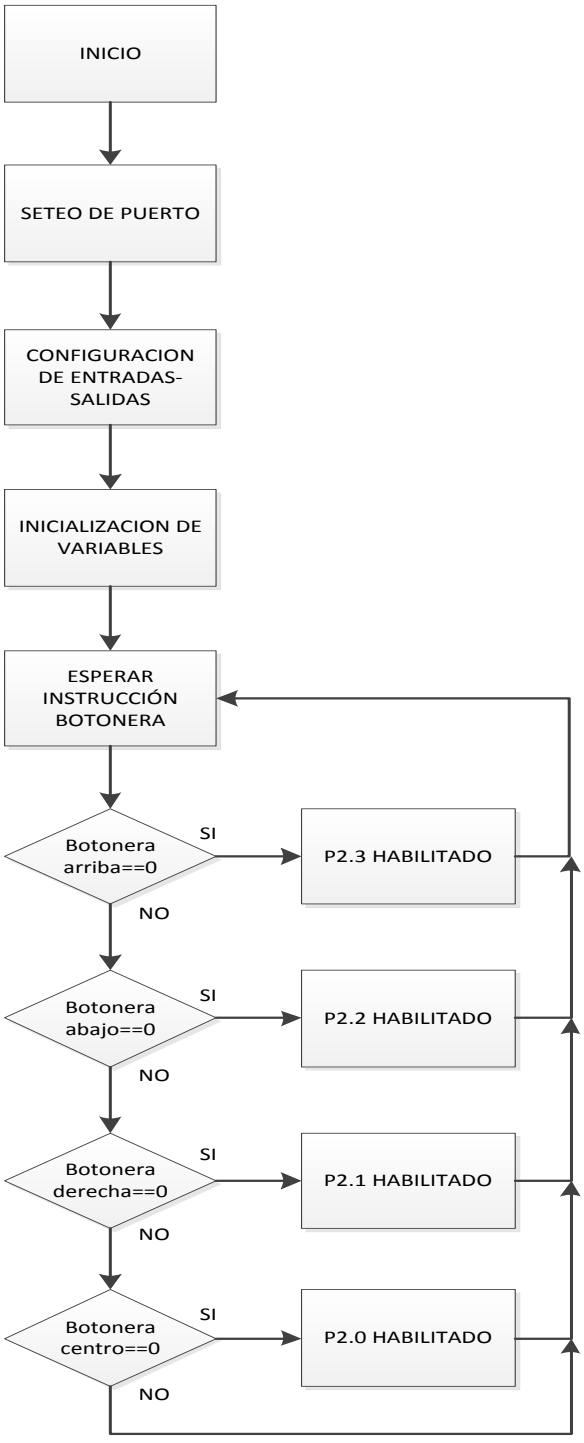

Figura 3.7: Diagrama de Flujo del Programa Controlador de las botoneras en la LPC1769 Interfaz.c.

CAPÍTULO 4

PRUEBAS Y SIMULACIONES

En este capítulo detallaremos las actividades desarrolladas para la realización de los diferentes ejercicios, dando especificaciones de materiales y dispositivos que hemos utilizado y que son necesarios para dicha implementación.

4.1 Elementos de Hardware

Para el desempeño óptimo del proyecto es necesario el uso de herramientas y materiales electrónicos que permitirán la interacción con el usuario. A continuación se detallaran los elementos más importantes, tomando en cuenta que en este proyecto existen elementos básicos pero a su vez indispensables tales como resistencias, botoneras, cable utp y LEDs.

4.1.1 Protoboard

El Protoboard es una placa de uso genérico reutilizable o semipermanente, usado para construir prototipos de circuitos electrónicos con o sin soldadura. Normalmente se utilizan para la realización de pruebas experimentales, como en nuestro caso. Está compuesto por bloques de plástico perforados y numerosas láminas delgadas, de una aleación de cobre, estaño y fósforo, que unen dichas perforaciones, creando una serie de líneas de conducción paralelas. Las líneas se cortan en la parte central del bloque de plástico para garantizar que dispositivos en circuitos integrados tipo DIP (Dual Inline Packages) puedan ser insertados perpendicularmente a las líneas de conductores.

Debido a las características de capacitancia (de 2 a 30 pF por punto de contacto) y resistencia que suelen tener los protoboard están confinados a trabajar a relativamente baja frecuencia (inferior a 10 o 20 MHz, dependiendo del tipo y calidad de los componentes electrónicos utilizados), los demás componentes electrónicos pueden ser montados sobre perforaciones adyacentes que no compartan la tira o línea

conductora, e interconectados a otros dispositivos usando cables, usualmente unifilares [19].

4.1.2 Kit Control de Motor

El kit contiene el “Motor Control Board” y la tarjeta LPCXPRESSO LPC1114 Cortex M0 las cuales se muestran en la figura 4.1 para el control del motor BLDC sin escobillas. El “Motor Control Kit” se trata de una plataforma universal para el control en baja tensión del motor, todo esto basado en microcontroladores de NXP.

Figura 4.1: LPCXpresso Motor Control Kit [2].

A continuación detallamos las principales características del Motor Control Board: zócalo para LPCXpresso LPC1114, LPC11C24 and LPC1343, zócalo para LPCXpresso LPC176x, 12-30V voltaje de entrada, 17A máxima corriente (max. 300W

salida), 15W fuente de alimentación (+11V, +5V, +3.3V), joystick de 5 posiciones, dimensiones de 200 x 150 mm, pantalla OLED de 96x64 pixeles.

4.2 Encendido secuenciado de leds.

Como controlador tendremos a la tarjeta LPC1769 quien recibirá información desde el exterior para generar señales de voltaje a través del puerto 2 y así poder apreciar en los diodos leds dos secuencias las cuales van a depender de la botonera conectada a la tarjeta LPC1769.

Para realizar el ejercicio referente al encendido rotatorio de leds el cual se muestra en la figura 4.2 se requiere de los siguientes materiales:

- 1 Tarjeta LPC1769 NXP CORTEX M3.
- 1 Botonera.
- 8 resistencias 330 Ω.
- 8 diodos leds

La figura 4.3 muestra la Conexión de los elementos a la tarjeta LPC1769.

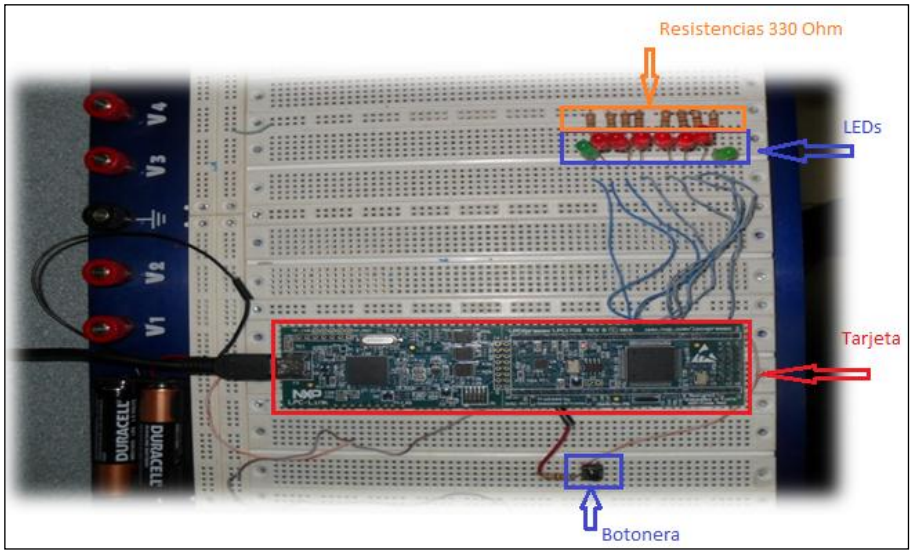

Figura 4.2: Desplazamiento Rotatorio de Leds.

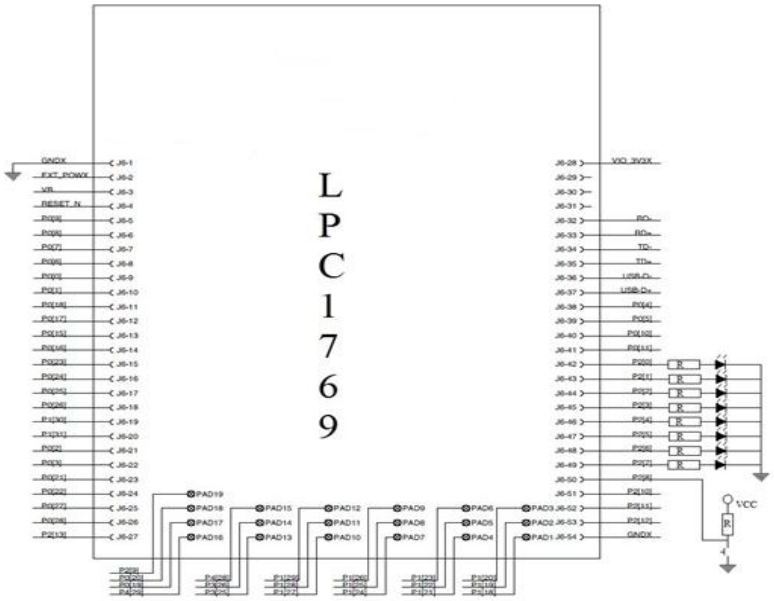

Figura 4.3: Conexión de los elementos a la tarjeta LPC1769.

4.3 Control del motor BLDC mediante la tarjeta LPC1114

En este caso como controlador tendremos la tarjeta LPC1114 la cual se encargará de controlar el motor BLDC, dependiendo de las señales enviadas de las 4 botoneras externas, con estas podemos controlar su velocidad, sentido de giro, iniciar o detener su movimiento.

Para realizar el ejercicio referente al control de motor BLDC, mediante la tarjeta LPC1114 el cual se muestra en la figura 4.4 se requiere de los siguientes materiales:

- 1 Tarjeta LPC1114 NXP CORTEX M0.
- 1 “Motor Control Board” NXP.
- 4 Botoneras.
- 4 Resistencias 270 Ω.
- 4 Resistencias 10 kΩ.
- 4 capacitores de 100 nf.

La figura 4.5 muestra la Conexión de los elementos a la tarjeta LPC1114.

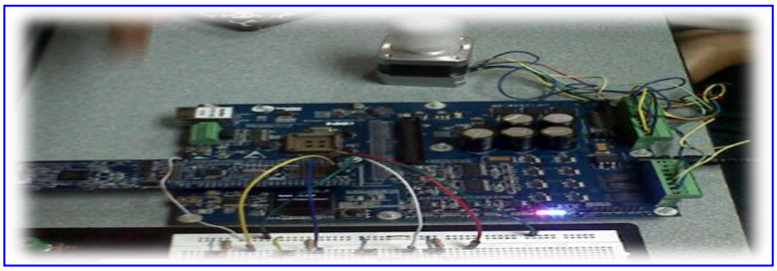

Figura 4.4: Control de motor BLDC con la tarjeta LPC1114.

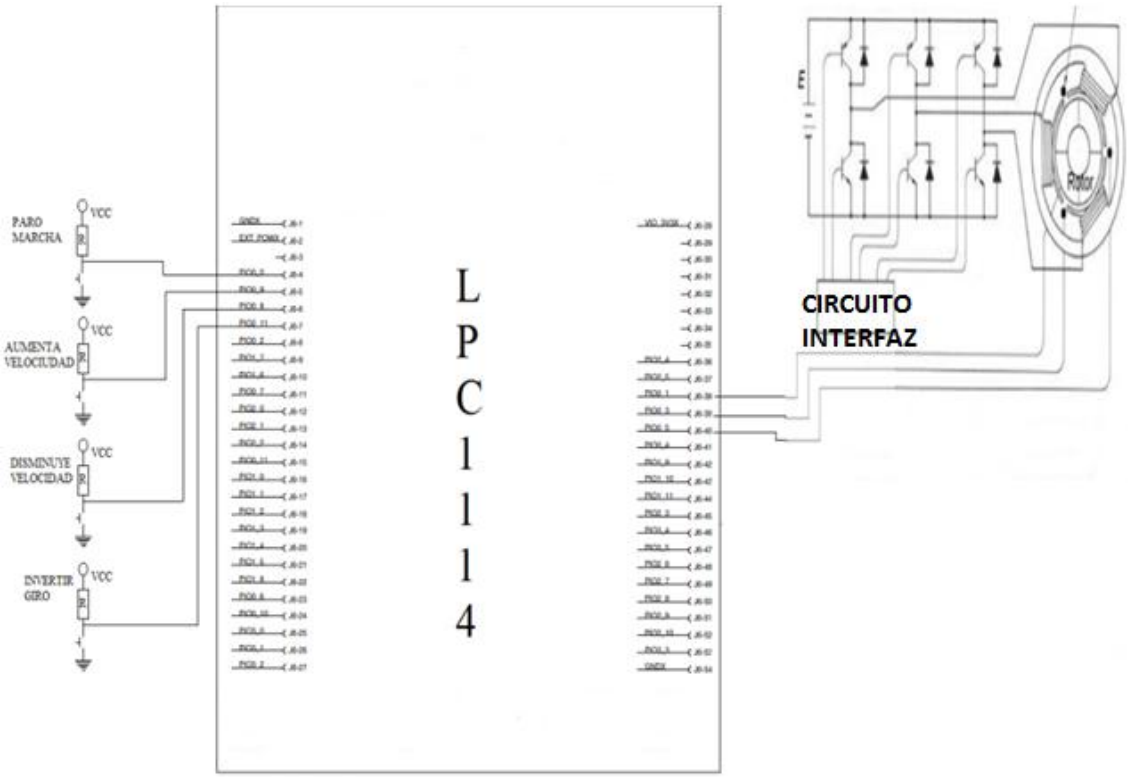

Figura 4.5: Conexión de los elementos a la tarjeta LPC1114.

4.4 Control del motor BLDC mediante las tarjetas LPC1769 Y LPC1114

En este caso como controlador tendremos la tarjeta LPC1114 la cual se encargará de controlar el motor BLDC, La tarjeta LPC1769 es la

aquella que se encarga de procesar las señales que llegan desde el exterior ya sea de un joystick o de un conjunto de botoneras como lo es para nuestro caso, dicha tarjeta envía las señales de voltaje para de esta manera lograr alguna respuesta inmediata en el motor BLDC.

Para realizar el ejercicio referente al control de motor BLDC mediante las tarjetas LPC1114 y LPC1769 el cual se muestra en la figura 4.6 se requiere de los siguientes materiales:

- 1 Tarjeta LPC1114 NXP CORTEX M0.
- 1 Tarjeta LPC1769 NXP CORTEX M3.
- 1 “Motor Control Board” NXP.
- 4 Botoneras.
- 4 Resistencias 270 Ω.
- 4 Resistencias 10 kΩ.
- 4 Capacitores de 100 nf.

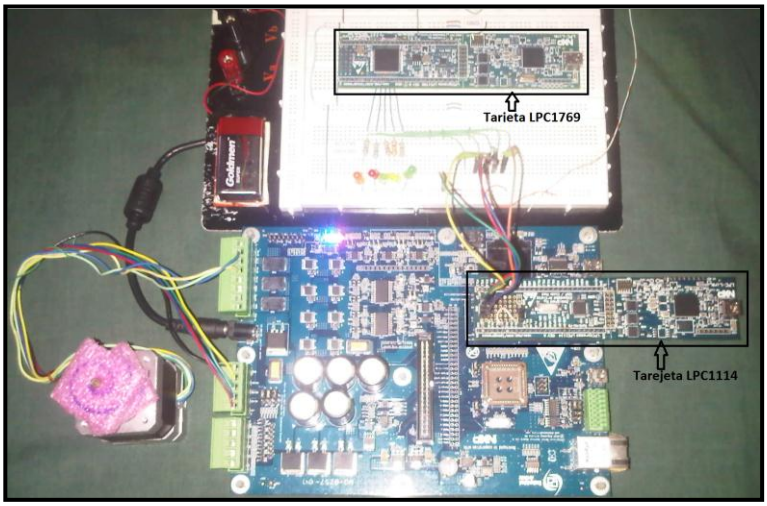

Figura.4.6: Control de motor BLDC usando tarjetas LPC1114 y LPC1769.

La figura 4.7 muestra la Conexión de los elementos a la tarjeta LPC1114 y LPC1769.

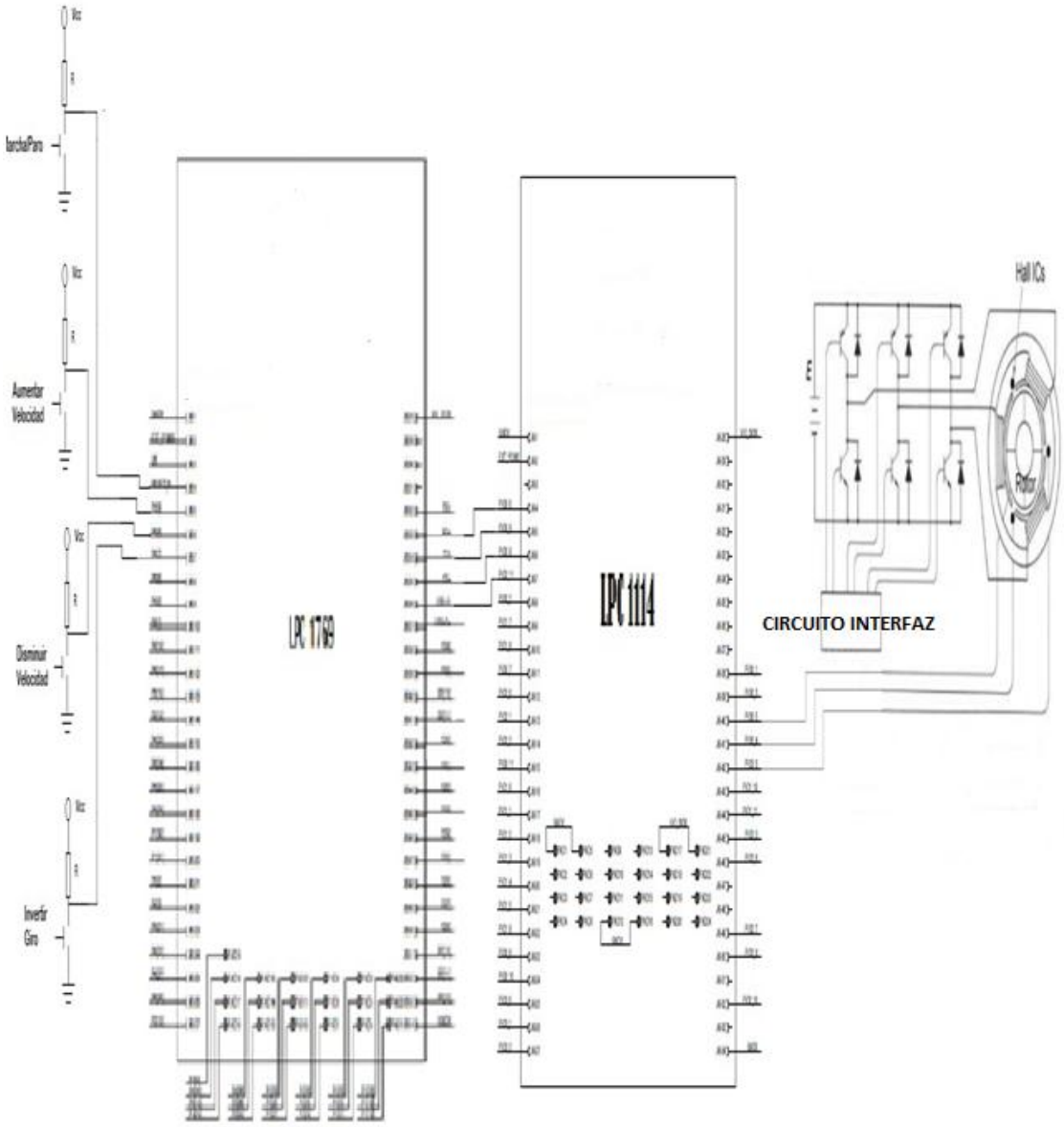

Figura 4.7: Conexión de los elementos a las tarjetas LPC1114 y LPC1769.

4.5 Simulación y comparación de sistemas de motores BLDC con sensores y sin sensores.

Tras el estudio de estos sistemas es necesario demostrar que las dos programaciones, aunque difieren en su estructura pueden tener el mismo fin en un proyecto, debido a que los dos sistemas trabajan de la misma manera, es por eso que a continuación ilustraremos las simulaciones de estos dos sistemas.

Para las simulaciones se utilizaron motores BLDC bajo el simulador Proteus. La programación completa se la detalla en el anexo 5, en lenguaje Asembler. Lo que vamos a demostrar es que para ambos casos el control, ya sea para motores BLDC sin sensores o con sensores sus señales de salidas serán las mismas, la diferencia radica en que el control de un motor con sensores es mucho más fácil debido a que en todo momento sabemos la posición del rotor cuando este está girando, caso contrario en motores sin sensores no existe dicha técnica.

Para el caso de la figura 4.8 y figura 4.10 donde se detalla la simulación de un sistema de control de motor BLDC con sensores y

sin sensores la conmutación de las señales de entradas al motor son las mismas para ambos casos.

En la figura 4.8 se detalla la circuitería de la simulación de motores con sensores, y en la fig. 4.9 las señales de salidas donde se muestra el tren de pulso generado por los sensores de efecto hall que están sincronizados en la armadura del rotor, para poder demostrar que las señales de salidas son idénticas en los dos casos se realizó una división de 20 milisegundos por cuadro.

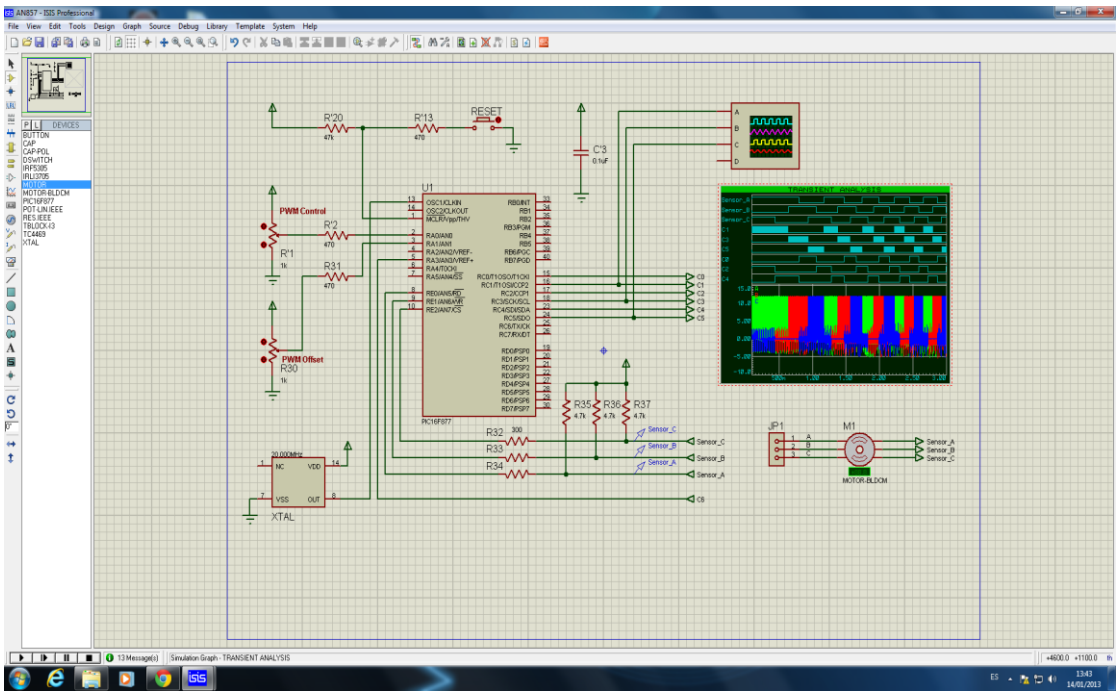

Figura 4.8: Simulación de Motor BLDC con sensores.

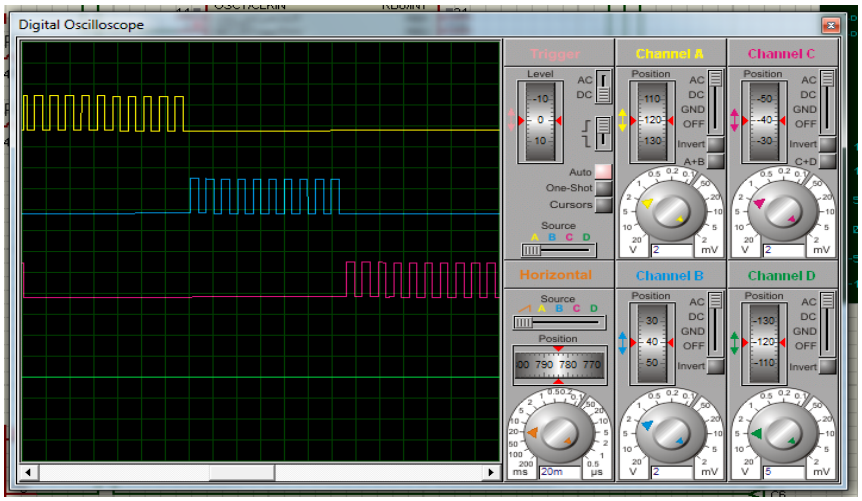

Figura 4.9: Simulación de las salidas del Motor BLDC con sensores.

En la figura 4.10 se detalla el esquemático para la simulación de control de motor BLDC sin sensores, donde las señales de salidas de motores BLDC sin sensores que a pesar de que podemos comprobar que son iguales a las señales de salidas del ejemplo de motores con sensores figura 4.11, su conmutación no es sincronizada debido a que no existe una técnica para controlar el giro del rotor, es decir no existe sensor alguno que detecte la posición del rotor. Por lo tanto el control del giro en motores BLDC sin sensores es el más básico, esto quiere decir que con un cambio de polaridad podemos realizar el cambio de giro.

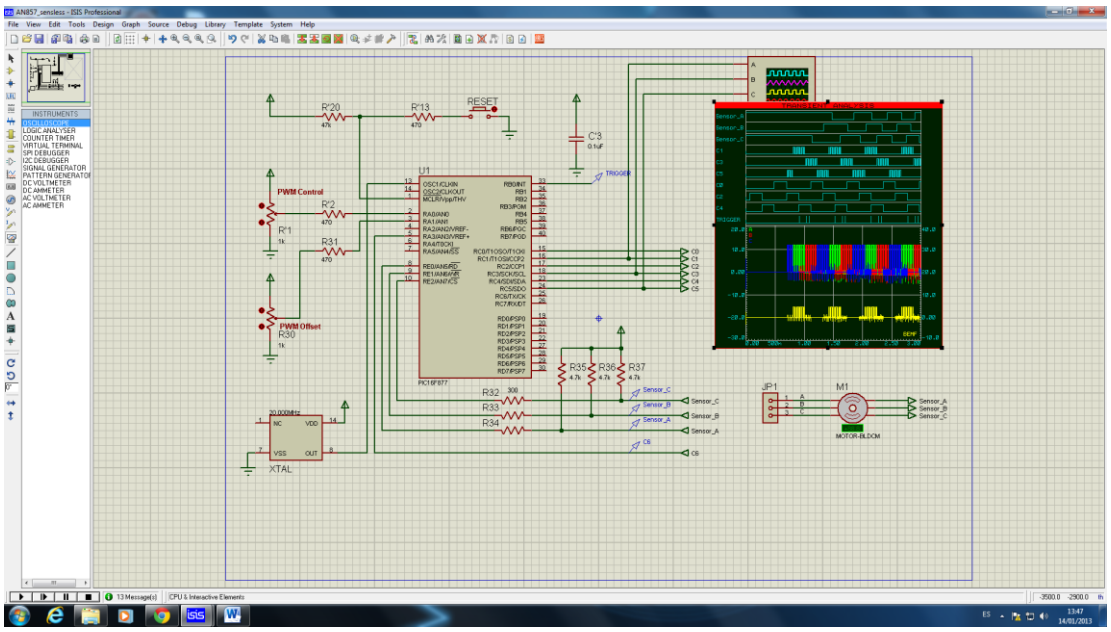

Figura 4.10: Simulación de Motor BLDC sin sensores.

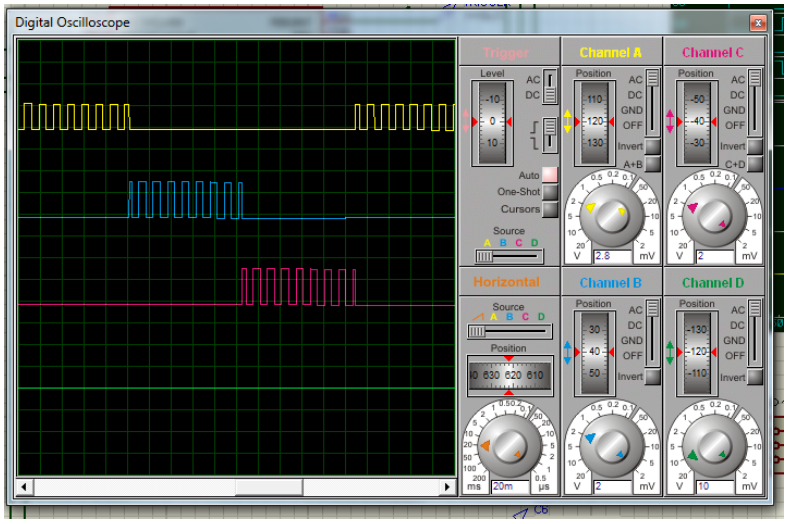

Figura 4.11: Simulación de las salidas del Motor BLDC sin sensores.

Por lo tanto se demuestra que para ambos controles de motores BLDC las salidas son iguales (figura 4.9 y figura 4.11) ya sean para un motor

BLDC con sensores o sin sensores. Por otra parte realizamos un ejemplo práctico para demostrar el uso del motor BLDC sin sensores en proyectos de aeromodelismo, el cual trae una programación precargada en uno de los dispositivos utilizados en dicho ejemplo llamado “Turnigy”. Para este ejemplo se utilizara un kit que bien con un motor BLDC sin sensores; el cual consiste en una pila de 3 celdas de 11.1 voltios, un control remoto que trabaja a 2,4 GHz, un receptor de 6 canales.

Lo que se quiere es demostrar que este tipo de motor son potentes pero que la aplicación es dirigida en su gran porcentaje al área de aeromodelismo, debido a que los cambios de sentido en el giro se lo hace de forma manual, es decir que invertimos la polaridad de las señales entrantes para poder tener un giro en sentido contrario al establecido, la figura 4.12 muestra el ejemplo práctico con todos los dispositivos.

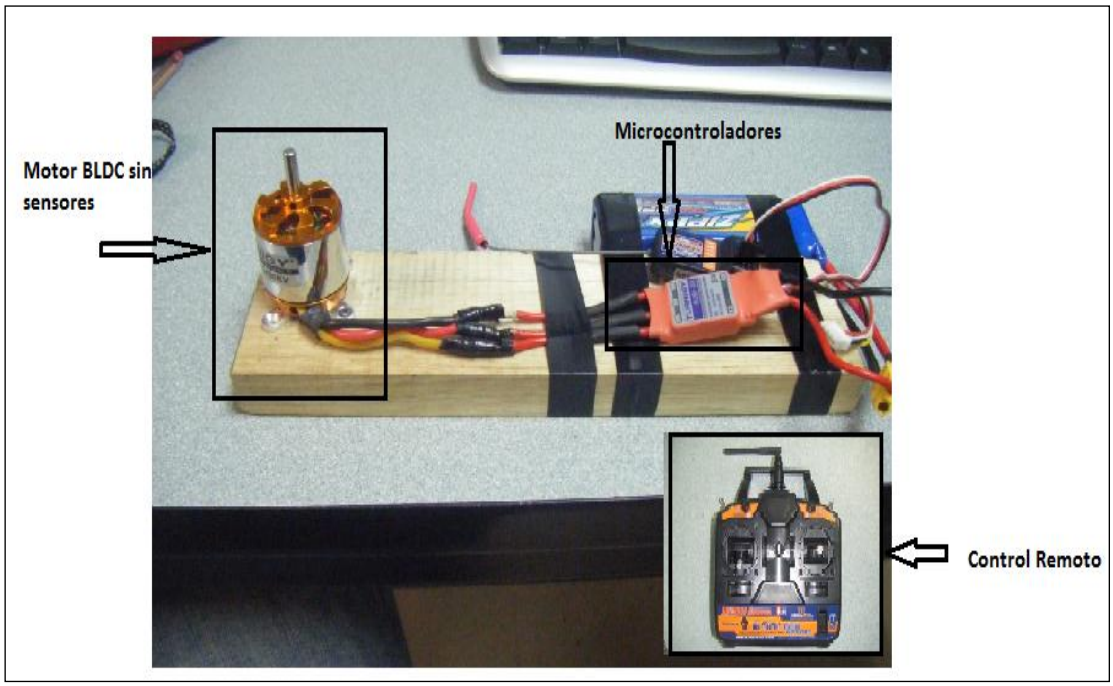

Figura 4.12: Motor BLDC sin sensores y demás equipos.

CONCLUSIONES

1. Al analizar el funcionamiento de los motores BLDC tanto con sensores como sin sensores, es válido decir que funcionan de la misma manera solo que la estructura en la programación difiere, siendo la de motores BLDC sin sensores mucho más compleja, pero teniendo resultados funcionales. Las señales de salidas son las mismas, la diferencia radica en que tienen distintos campos de desarrollo.

2. Los motores BLDC al utilizar sensores, permiten tener un conocimiento exacto de la posición en la que se encuentra el rotor y por ende poder interpretar el movimiento, además que el cambio de giro lo realiza sin ninguna dificultad dado al mismo principio de los sensores. A pesar que de los dos motores BLDC con y sin sensores tienen como resultado la misma eficiencia y señales de salida, se concluye que los motores BLDC con sensores son más funcionales y por ende más utilizados en el campo laboral,.

3. Los motores BLDC obtienen mejor tiempo de respuesta ya que en las pruebas realizadas en laboratorio de microcontroladores de la ESPOL, al invertir la dirección del movimiento del motor este respondía inmediatamente sin importar el tipo de comunicación que se esté usando. Además al realizar nuestro proyecto pudimos darnos cuenta que para el caso particular del ejercicio en que se utiliza la tarjeta controladora del Motor, se tuvo que usar una tarjeta diferente a la esperada la cual fue la LPC1114 para así acoplarla con el Motor, con esto se verifico el buen funcionamiento de nuestra programación, por lo tanto se puede concluir que las instrucciones dadas por las botoneras influyen en la transmisión de datos hacia el motor por lo que el tiempo de respuesta estará limitado por estos elementos lo que ocasionará pequeños retardos en las instrucciones del motor.

4. Una aplicación de los motores BLDC sin sensores, es el área de aeromodelismo. Estas aplicaciones buscan es que se mantenga una velocidad constaste en el motor; como sabemos los motores sin sensores funcionan con tres bobinas de las cuales solo dos son utilizadas mientras que la tercera es la encargada de censar. Por lo tanto se puede decir que los motores sin sensores sirven en aplicaciones con menos requerimiento y los motores con sensores en aplicaciones industriales donde las aplicaciones son más exigentes y necesitan saber en cada instante la posición del rotor y así poder introducir la siguiente instrucción sin que afecte el rendimiento.

RECOMENDACIONES

1. Es importante que en la configuración de los diferentes tipos de comunicación, siempre se empleen los mismos parámetros, dados por quienes programaron el control del motor, tanto en el AvrButterfly como en la tarjeta LPC1769. Si no se tiene esta concordancia no se podrá establecer la comunicación entre ambos dispositivos.

2. Es importante considerar que los proyectos pueden ser utilizados por otras personas, las cuales necesitan tener la información pertinente de forma comprensible y clara para facilitar la implementación de nuevos proyectos.

3. Antes de realizar un proyecto es preferible que las personas lean todo lo relacionado con el microcontrolador Atmega169, AvrButterfly y el kit Motor Control, así como también la documentación de esta tesina, con lo cual podrán sacarle el mayor provecho a esta información minimizando los

errores y el tiempo de ejecución llegando así a cumplir con las expectativas tanto del tutor como del desarrollador.

4. Para realizar la comunicación entre la PC y la tarjeta de desarrollo LPC1114 o LPC1769, es necesario que se cuente con un cable de comunicación serial RS232 USB. Se recomienda que estos adaptadores sean de buena calidad, para tener una comunicación óptima.

5. Es importante, antes de manipular este tipo de hardware, leer las normas de seguridad ya que la mayoría de sus integrados son de tecnología cmos y por lo tanto son extremadamente delicados pudiendo sufrir un cortocircuito si se los manipula de una manera errada.

6. Los cambios de giro del motor son bruscos, por lo tanto es importante que se reduzca prudencialmente la velocidad para poder apreciar el cambio, esto se experimentó en el laboratorio.

7. Se debe tomar en cuenta también conectar el GND de las tarjetas y el kit del motor control en un punto común para las tres referencias del GND.

BIBLIOGRAFÍA

[1] Bridgman Roger, Illustrated Guide To Great Inventions, EL TIEMPO, 2002.Fecha de consulta Febrero de 2012

[2] Kharsansky Alan, Introduccion al LPCXpresso y repaso del lenguaje C. Seminario de Sistemas Embebidos. http://laboratorios.fi.uba.ar/lse/seminario/material-2011/Sistemas_Embebidos-2011_2doC-Intro_a_LPCXpresso_y_repaso_lenguaje_C-Kharsansky.pdf; Fecha de consulta Marzo de 2012.

[3] Kharsansky Alan, Tutorial LPCXpresso IDE CodeRed. Seminario de Sistemas Embebidos. www.laboratorios.fi.uba.ar/lse/seminario/material-2011/Sistemas_Embebidos-2011_2doCMini__Tutorial_CodeRed_LPCXpresso IDE Kharsansky.pdf; Fecha de consulta Febrero del 2012.

[4] NXP, LPCXpresso Getting started with NXP **LPCXpresso** http://ics.nxp.com/support/documents/microcontrollers/pdf/lpcxpresso.getting.started.pdf Fecha de consulta Febrero del 2012.

[5] Jay Hafling, wordPress, http://www.micro32b.wordpress.com/

Fecha de consulta Marzo del 2012

[6] NXP, LPC17xx User manual.

http://www.nxp.com/documents/user_manual/UM10360.pdf

Fecha de consulta Febrero del 2012.

[7] Jorge Cotte y Andres Moreno, Universidad Nacional de Colombia, Diseño de Control Robusto de Velocidad de Motores Brushless para Robotica Aerea.

http://www.bdigital.unal.edu.co/1896/1/jorgemariocottecorredor.2010.pdf

Fecha de consulta Febrero del 2012.

[8] Juanpere Tolrá, Roger, Técnicas de control para motores Brushless, Universidad de Cataluña, Motion Control Department, Barcelona, España.

Roger Juanpere Tolrà, Ingenia-cat, http://www.ingeniamc.com/Es/-Control-techniques-for-brushless-motors.pdf

Fecha de consulta Abril del 2012

[9] Embedded Artists,

http://www.embeddedartists.com/products/lpcxpresso/lpc1114_xpr.php

Fecha de consulta Febrero del 2012.

[10]Embedded Artist, LPCXpresso Motor Control kit. User Guide.

http://ics.nxp.com/support/microcontrollers/motor.control/;

Fecha de consulta Marzo 2012.

[11] Johnson Electric, http://www.johnsonelectric.com/en/resources-for-engineers/automotive-applications/motion-technology/ec-motor-brushless.html

Fecha de consulta junio del 2012

[12] Gerardo Man, Caracteristicas de los Motores BLDC

http://www.brushlessmotor.com.ar/Caracteristicas.html

Fecha de consulta Febrero del 2012.

[13] 2 Microchip Technology Inc, Brushless DC Motor Control **Made Easy - Microchip** AN857

http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf

Fecha de consulta Febrero del 2012.

[14] Stevens Aero Model, http://www.stevensaero.com/Hacker-Brushless-Motor-200W-A20-20L-Evo-V2-HBM-A20-20L-p-16820.html

Fecha de consulta Febrero del 2012.

[15] Juanpere Tolrá, Roger, Técnicas de control para motores Brushless, Universidad de Cataluña, Motion Control Department,

Barcelona, España.

Fecha de consulta julio del 2012

[16] LPCWare, http://www.lpcware.com/content/project/nxpusblib/get-it

Fecha de consulta Febrero del 2012

[17] DataSheet Directory, http://www.datasheetdir.com/Sensorless-Control-Of-Bldc-Motors-Using-Attiny261-461-861+Application-Notes

Fecha de consulta Marzo de 2012.

[18] Xavier Lopez, Maniobras de Máquinas Eléctricas mediante el RELE programable Zelio, ESCOLA TECNICA SUPERIOR ENGINYERIA Universitat Rovira I Virgili

Accionamiento con motor brushless

http://deeea.urv.cat/DEEEA/lguasch/XavierLopez%20PFC.pdf

Fecha de consulta Febrero del 2012.

[19] Diana Mercedes Bohorquez, Janifer Durley y Aleida Urrea,

http://es.slideshare.net/fapilla/el-protoboard-9542384

Fecha de consulta Febrero del 2012.

ANEXOS

ANEXO I
PROGRAMA EN LENGUAJE C DEL ENCENDIDO SECUENCIADO DE LEDS

```
/*****************************************************************************
 *   Autores: Rony Chango / Jacinto Palma
 *   Nombre: Rotación de LEDS
*****************************************************************************
Descripción
*EL CODIGO EJECUTA UNA SECUENCIA DE LUCES 8 LEDS
*CONECTADOS EN EL PUERTO 2 GPIO2.0 HASTA GPIO2.7
*LA SECUENCIA DEPENDERA DEL ESTADO DE LA BOTONERA.
*****************************************************************************/
#include<cr_section_macros.h>
#include <NXP/crp.h>
// Variable to store CRP value in. Will be placed automatically
// by the linker when "Enable Code Read Protect" selected.
// See crp.h header for more information
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
#include "lpc17xx.h"
#include "type.h"
int main (void)
{
uint32_ti, j; /*variables a utilizar*/
/* SystemClockUpdate() updates the SystemFrequency variable */
SystemClockUpdate();
  LPC_GPIO2->FIODIR = 0xFFFFFEFF; /* P2.xx definido como salida */
LPC_GPIO2->FIOMASK= 0x00000000;
  LPC_GPIO2->FIOCLR = 0xFFFFFFFF;  /* todos los leds off */

while(1) /* lazo infinito */
  {
            /*lee y compara*/
          if (LPC_GPIO2->FIOPIN==0x00000000)
          {
            for(i = 0; i< 8; i++)
            {
              LPC_GPIO2->FIOSET = 1 <<i;
            for(j = 1000000; j > 0; j--);/*retardo*/
            }
            LPC_GPIO2->FIOCLR = 0xFFFFFFFF; /* todos los leds off */
            for(j = 1000000; j > 0; j--);/*retardo*/
          }
          /*lee y compara*/
          if (LPC_GPIO2->FIOPIN==0x00000100)
```

```
        {
          for(i = 8; i>0; i--)
          {
          LPC_GPIO2->FIOSET = 1 << (i-1);
          for(j = 1000000; j > 0; j--);/*retardo*/
          }
          LPC_GPIO2->FIOCLR = 0xFFFFFFFF; /* todos los leds off */
          for(j = 1000000; j > 0; j--);/*retardo*/
        }
}
}
```

ANEXO II
PROGRAMA EN LENGUAJE C DEL CONTROL DEL MOTOR BLDC MEDIANTE LA TARJETA LPC1114 (main.c)

```
/*****************************************************************************
* $Id::BLDC.c 3871 2010-07-16 11:50:22Z gerritdewaard              $
*
 * Nombre: Main
 * Autor: Code Red Technologies
* Description:
 * Programa principal que nos permite manejar en un lazo infinito el
 * motor BLDC utilizando determinados procedimientos con la ayuda
 * del motor control board, además nos permite mostrar en pantalla
 * la gráfica de la velocidad variable del motor.
*****************************************************************************/
#include "application.h"
#define GRAPH_LIMIT     64-8
/*****************************************************************************
 * External global variables
 ****************************************************************************/
externvolatile uint32_t SysTick_cntr;
externvolatileMOTOR_TypeDef Motor;
//extern volatile RX_PACKET RxPackets[RX_MAX_PACKETS];
/*****************************************************************************
 * Local variables
 ****************************************************************************/
volatile SYNC_STRUCT sync;
volatile BLDC_STRUCT bldc;
volatile uint8_t blink = 0;
volatile uint8_t TXBUF[0xFF];
```

```
volatile uint8_t RXBUF[0xFF];
uint16_tconnTimer=0;                    // Timer for connection timeout
volatile uint8_t oled_X = 0;
volatile uint8_t oled_Y = 0;
volatile uint8_t oled_Y_PixBUF[96];
volatile uint8_t show_graph = FALSE;
volatile uint8_t show_temperature = FALSE;
volatile uint8_t clear_screen = TRUE;
volatile uint8_t sec5 = 5;
/*****************************************************************************
 * Local Functions
 *****************************************************************************/
voidMemCopyBYTE(uint8_t *src, uint8_t *dst, uint32_t count);
voidMemCopyDWORD(uint32_t *src, uint32_t *dst, uint32_t count);
voidInitSyncPacket (void);
voidProcessRxPacket(uint8_t index);

/*****************************************************************************
** Functionname: Call_5ms
** Descriptions: procedimiento que realiza el cálculo PID
** Parámetros: ninguno
** Valor a Retornar: ninguno
*****************************************************************************/
voidCall_5ms(void)
{
        if (Motor.Enable&& (Motor.RampingUp == FALSE))
        {
                /* Do the PID calculations */
                vPID_RPM(&Motor);
        }
}
/*****************************************************************************
** Functionname: Call_25ms
** Descriptions: procedimiento que muestra un punto de la grafica de    **
velocidad del motor
** Parámetros: ninguno
** Valor a Retornar: ninguno
*****************************************************************************/
voidCall_25ms(void)
{
#if MC_BOARD_ENABLE_OLED == 1
        if (show_graph == TRUE)
        {
                /* Down scale the actual RPM value to screen size */
```

```
                oled_Y = (Motor.RPM/80);

                /* Clip the output */
                if (oled_Y>= GRAPH_LIMIT) oled_Y = GRAPH_LIMIT;

                /* Clear the previous pixel */
                oled_putPixel(oled_X, oled_Y_PixBUF[oled_X] ,
OLED_COLOR_BLACK);
                oled_Y = 64 - oled_Y;
                oled_putPixel(oled_X, oled_Y , OLED_COLOR_WHITE);

                oled_Y_PixBUF[oled_X] = oled_Y;
                oled_X++;

                if (oled_X == 96) oled_X = 0;
        }
#endif
}
/*****************************************************************************
** Functionname: Call_100ms
** Descriptions: procedimiento que aumenta o disminuya la velocidad del
** motor.
** Parámetros: ninguno
** Valor a Retornar: ninguno
*****************************************************************************
voidCall_100ms(void)
{
        if (Motor.Enable&&Motor.RampingUp)
        {
                if (clear_screen == TRUE)
                {
                        /* Clear the OLED screen */
                        oled_clearScreen(OLED_COLOR_BLACK);
                        clear_screen = FALSE;
                }
                /* Ramp-up the motor.. */
                vBLDC_RampUp (&Motor, Motor.sp);
        }

}
/*****************************************************************************
** Functionname: Call_1s
** Descriptions: procedimiento que nos muestra la temperature en
** pantalla.
```

```
** Parámetros: ninguno
** Valor a Retornar: ninguno
*****************************************************************************
voidCall_1s(void)
{
        /* */
        vCOMMS_send(&Motor);

#if (MC_BOARD_ENABLE_OLED == 1)
#if (MC_BOARD_ENABLE_LM75 == 1)
static uint8_t firstTime = 1;
        uint8_t  buffer[13] = "Temp: xx.x C";
        uint16_t temperature;

        if (show_temperature == TRUE)
        {
                if (firstTime)
                {
                        /* Clear the OLED screen */
                        oled_clearScreen(OLED_COLOR_BLACK);
                        oled_putString(1, 1, (char *)buffer,
                        OLED_COLOR_WHITE, OLED_COLOR_BLACK);
                        firstTime = 0;
                }
                temperature = (uint16_t)lm75a_readTemp();
                temperature = (temperature + 5) / 10; //round to 0.1 degree
                buffer[6] = ((temperature / 100) % 10) + '0';
                buffer[7] = ((temperature / 10) % 10) + '0';
                buffer[8] = '\0';
                oled_putString((6*6), 1, (char *)&buffer[6],
                OLED_COLOR_WHITE, OLED_COLOR_BLACK);
                buffer[9] = (temperature % 10) + '0';
                buffer[10] = '\0';
                oled_putString((9*6), 1, (char *)&buffer[9],
                OLED_COLOR_WHITE, OLED_COLOR_BLACK);
        }
        elsefirstTime = 1;
#endif //MC_BOARD_ENABLE_OLED
#endif //MC_BOARD_ENABLE_LM75
}
/*****************************************************************************
** Nombre de la funcion: Appl_Init
** Descripción: Función que permite la inicialización de diversas
**aplicaciones.
```

```
** Parametros:Ninguno
** Valor a retornar:Nínguno
**
*****************************************************************************
voidAppl_Init (void)
{
        /*************************************************/
        /* SYSTICK
        */
        /*************************************************/
        SysTick_Config (SysTick_VALUE);
        /*************************************************/
        /* FREERUNNING COUNTER
                */
        /*************************************************/
        /* Setup Timer16 1 as free running counter for e.g. RPM calc..*/
        /* Enable the clock to Timer16 1 */
        LPC_SYSCON->SYSAHBCLKCTRL |= (1<<8);
        /* Set the prescaler to get never get an overflow */
        LPC_TMR16B1->PR = 55;
        /* RESET timer16 1  */
        LPC_TMR16B1->TCR = 1<<1;
        /* ENABLE timer16 1 */
        LPC_TMR16B1->TCR = 1;
        /*************************************************/
        /* BLDC MOTOR INIT
                */
        /*************************************************/
        vBLDC_Init(&Motor);
        /*************************************************/
        /* HALL sensors      INIT
        */
        /*************************************************/
        GPIOInit();
        GPIOSetDir( LED_PORT, LED_BIT, 1 );
        GPIOSetValue( LED_PORT, LED_BIT, 0 );
        /* Set the direction to input */
        GPIOSetDir(HALL_A_PORT,HALL_A_PIN,0); // HALL A
        GPIOSetDir(HALL_B_PORT,HALL_B_PIN,0); // HALL B
        GPIOSetDir(HALL_C_PORT,HALL_C_PIN,0); // HALL C
        /* Setup the interrupt, seq: portNum, bitPosi, sense = edge, single =
        both, event*/
        GPIOSetInterrupt(HALL_A_PORT,HALL_A_PIN,0,1,1);
        GPIOSetInterrupt(HALL_B_PORT,HALL_B_PIN,0,1,1);
```

```
        GPIOSetInterrupt(HALL_C_PORT,HALL_C_PIN,0,1,1);
        /* Enable the interrupts */
        GPIOIntEnable(HALL_A_PORT,HALL_A_PIN);        // HALL A
        GPIOIntEnable(HALL_B_PORT,HALL_B_PIN);        // HALL B
        GPIOIntEnable(HALL_C_PORT,HALL_C_PIN);        // HALL C
        NVIC_EnableIRQ(EINT2_IRQn);
        /*************************************************/
        /* I2C INIT
        */
        /*************************************************/
I2CInit( (uint32_t)I2CMASTER);
        /*************************************************/
        /* OLED INIT
        */
        /*************************************************/
#if MC_BOARD_ENABLE_OLED == 1
   /* Initialize the OLED */
oled_init();
   /* Clear the OLED screen */
oled_clearScreen(OLED_COLOR_BLACK);
#endif //MC_BOARD_ENABLE_OLED
#if USE_STARTUP_GUI == 1
        /*************************************************/
        /* Startup message on OLED
        */
        /*************************************************/
vGUI_StartupMessage();
#endif //USE_STARTUP_GUI
        /*************************************************/
        /* JOYSTICK INIT
        */
        /*************************************************/
#if MC_BOARD_ENABLE_JOYSTICK == 1
joystick_init();
#endif
        /*************************************************/
        /* CAN ROM DRIVER INIT
        */
        /*************************************************/
#if (USE_CAN == 1)
        /* Initialize the CAN ROM driver handlers */
        vCAN_initRomHandlers();
#endif
        /*************************************************/
```

```
        /* UART INIT
        */
        /*************************************************/
        /* Initialize the UART */
#if (USE_UART == 1)
        /* Push UART_RS485 high for UART -> USB mode */
        GPIOSetDir(3,4,1);
        GPIOSetValue(3,4,1);
        /* Initialize UART */
        UARTInit(115200);
        /* Add a 5 sec delay before sending startup message to UART. */
        /* Delay added to allow for USB enumeration on PC after reset.*/
        SysTick_cntr = 0;
        while (SysTick_cntr> 5000);
        SysTick_cntr = 0;
        UARTSend((uint8_t *)"\n\r/*********************************************/",
50);
        UARTSend((uint8_t *)"\n\r/*                                           */", 50);
        UARTSend((uint8_t *)"\n\r/* BLDC Motor Control Demo Application on
*/", 50);
        UARTSend((uint8_t *)"\n\r/* LPCXpresso Motor Control Board
*/", 50);
        UARTSend((uint8_t *)"\n\r/* Date: 2011-02-14, rev A                 */",
50);
        UARTSend((uint8_t *)"\n\r/*                                           */", 50);
        UARTSend((uint8_t *)"\n\r/*********************************************/",
50);
        UARTSend((uint8_t *)"\n\rRPM\tMV\tActive\tDirection\n\r", 27);
#if (USE_NXP_GUI == 1)
        /*************************************************/
        /* PACKET INITIALIZATION
        */
        /*************************************************/
        InitSyncPacket();
#endif
#endif // (USE_UART == 1)
}
voidMemCopyBYTE(uint8_t *src, uint8_t *dst, uint32_t count)
{
        while(count--)*dst++ = *src++;
}
voidMemCopyDWORD(uint32_t *src, uint32_t *dst, uint32_t count)
{
        while(count--)*dst++ = *src++;
```

```
}
/*********************************************************
 * MAIN
/*********************************************************
intmain (void) {
        uint32_t cnt_5s = 0, cnt_1s=0, cnt_5ms=0, cnt_25ms=0, cnt_100ms=0;
//      uint8_t packet_cnt, index;
        /* Initialize the application */
        Appl_Init ();
        /* Disable the Motor */
        Motor.Enable = 0;
        /* Loop forever */
        while (1)
        {
                if ( SysTick_cntr>= cnt_5ms )
                {
                        cnt_5ms = SysTick_cntr+10;
                        Call_5ms();
                }
                if ( SysTick_cntr>= cnt_25ms )
                {
                        cnt_25ms = SysTick_cntr+25;
                        Call_25ms();
                }
                if ( SysTick_cntr>= cnt_100ms )
                {
                        cnt_100ms =SysTick_cntr +100;
                        Call_100ms();
                }
                if ( SysTick_cntr>= cnt_1s )
                {
                        cnt_1s = SysTick_cntr+1000;
                        Call_1s();
                }
                if ( SysTick_cntr>= cnt_5s )
                {
                        cnt_5s = SysTick_cntr+5000;
                        Call_5s();
                }
#if (USE_NXP_GUI == 1)
                /* Process Rx Packets */
                packet_cnt = GetRxPacketCnt();
                if(packet_cnt> 0)
                {
```

```
                index = GetNextRxPacketIndex();
                ProcessRxPacket(index);
            }
#endif
        }
}
```

ANEXO III
PROGRAMA CONTROLADOR DEL MOVIMIENTO DEL MOTOR BLDC HANDLERS.C

```
/*****************************************************************
* Nombre: handlers.c
 * Autor: Rony Chango / Jacinto Palma
 * Descripción:
 * Programa que nos permite el control de la velocidad del motor BLDC
 * Por medio de los puertos 2 y 3, además nos permite cambiar el
sentido
 * De giro del eje del motor, detener y activar su funcionamiento.
*****************************************************************/
#include"application.h"
externvolatileuint32_tbldc_Tick;
externvolatileuint32_tbldc_SySTickold;
externvolatileuint32_tbldc_SySTicknew;
externvolatileMOTOR_TypeDef Motor;
externvolatileuint8_tshow_graph;
externvolatileuint8_tshow_temperature;
volatileuint32_tSysTick_cntr = 0;
/*****************************************************************************
** Function name:           PIOINT0_IRQHandler
**
** Descriptions:            Use one GPIO pin(port0 pin1) as interrupt
source
**
** parameters:                      None
** Returned value:          None
**
*****************************************************************************/
#ifndef GPIO_GENERIC_INTS
voidPIOINT2_IRQHandler(void)
{
        uint32_tregVal;
        uint32_t regVal2;
        regVal = LPC_GPIO2->MIS;
        regVal2 = LPC_GPIO3->MIS;
```

```
switch (regVal)
{
        case (1<<HALL_A_PIN):
                /* Calculate the actual RPM */
                vBLDC_CalcRPM(&Motor);
                /* Read the HALL sensor  */
                vBLDC_ReadHall();
        break;
        case ((1<<HALL_B_PIN)):
        case ((1<<HALL_C_PIN)):
                vBLDC_ReadHall();
        break;
        case (1<<JOY_C_PIN):
                if (Motor.Enable)
                {
                        /* Disable the motor */
                        Motor.Enable = 0;
                        /* Set the RPM to 0 */
                        Motor.RPM = 0;
                        Motor.RampingUp = FALSE;
                }
                else
                {
                        /* Enable the motor */
                        Motor.Enable = 1;
                        Motor.CMT_CNT = 0;
                        Motor.RampingUp = TRUE;
                        show_graph = TRUE;
                        show_temperature = TRUE;
                }
        break;
        default:
        break;
}
switch (regVal2)
{
        case (1<<JOY_U_PIN):/*incremento de la velocidad*/
                Motor.sp+=50;
                if (Motor.sp>= 4100)/*velocidad máxima*/
                Motor.sp = 4100;
        break;
        case (1<<JOY_R_PIN):/*cambio de giro*/
                if (Motor.Direction == CW)
                        Motor.Direction = CCW;
```

```
                    else
                            Motor.Direction = CW;
                    break;
                    case (1<<JOY_D_PIN):/*decremento de la velocidad*/
                            Motor.sp-=50;
                            if (Motor.sp<= 0)
                            Motor.sp = 0;
                    break;
                        default:

                            break;
                }
                /* Clear all interrupt sources */
                LPC_GPIO2->IC = 0xFFF;
                LPC_GPIO3->IC = 0xFFF;
        return;
        }
        #endif//GPIO_GENERIC_INTS

        voidSysTick_Handler (void)
        {
                SysTick_cntr++;
                bldc_Tick++;
        }
```

ANEXO IV
PROGRAMA CONTROLADOR DE LAS BOTONERAS EN LA LPC1769 INTERFAZ.C

```
/*************************************************************
*   Autores: Rony Chango / Jacinto Palma
*   Nombre: Controlador de botoneras mediante tarjeta LPC1769
*    ************************************************************
#include<cr_section_macros.h>
#include<NXP/crp.h>
__CRPconstunsignedint CRP_WORD = CRP_NO_CRP ;
#include"lpc17xx.h"
#include"type.h"
intmain (void)
{
uint32_ti,j;
/* SystemClockUpdate() updates the SystemFrequency variable */
SystemClockUpdate();
```

```
LPC_GPIO2->FIODIR = 0xFFFFFF0F;/* P2.00 al P2.03 definido como salida,
P2.04 al P2.07 definido como entrada*/
LPC_GPIO2->FIOCLR = 0xFFFFFFFF;          /* turn off all the LEDs */
  LPC_GPIO2->FIOMASK= 0x00000000;
while(1)
 {
        for(i = 0; i< 4; i++)
         {
                LPC_GPIO2->FIOSET = 1 <<i;
         }
        if(LPC_GPIO2->FIOPIN==0x000000EF)
        {
                 LPC_GPIO2->FIOCLR = 0x00000001;
        }
        if(LPC_GPIO2->FIOPIN==0x000000DF)
{
                 LPC_GPIO2->FIOCLR = 0x00000002;
         }
        if(LPC_GPIO2->FIOPIN==0x000000BF)
{
                 LPC_GPIO2->FIOCLR = 0x00000004;
         }
        if(LPC_GPIO2->FIOPIN==0x0000007F)
        {
                LPC_GPIO2->FIOCLR = 0x00000008;
         }
        for(j = 1000000; j > 0; j--);/*retardo*/

        }
        }
```

ANEXO V
PROGRAMA CONTROLADOR DEL MOTOR BLDC SIN SENSORES PARA SIMULACIÓN

```
;*****************************************************************
;                                                                *
;                                                                *
;   Filename:    snsrless.asm                                    *
;   Date:        14 Jan 2002                                     *
;   File Version:  1.0                                           *
;                                                                *
;   Author:      W.R.Brown                                       *
;   Company:     Microchip Technolgy Incorporated                *
;                                                                *
```

```
;                                                       *
;*****************************************************************
;                                                       *
;    Files required: p16f877.inc                                *
;                                                       *
;                                                       *
;                                                       *
;*****************************************************************
;                                                       *
;    Notes: Sensorless brushless motor control                  *
;                                                       *
;        Closed loop 3 phase brushless DC motor control.            *
;        Two potentiometers control operation. One potentiometer (A0)  *
;        controls PWM (voltage) and RPM (from table). The other     *
;        potentiometer (A1) provides a PWM offset to the PWM derived
*
;        from A0. Phase A motor terminal is connected via voltage   *
;        divider to A3. This is read while the drive is on during   *
;        phase 4. The result is the peak applied voltage (Vsupply).  *
;        A3 is also read while the drive is on at two times during  *
;        phase 5. The result is the BEMF voltage. The BEMF voltage is  *
;        read at the quarter (t1) and mid (t2) points of the phase 5  *
;        period. BEMF is compared to VSupply/2. If BEMF is above    *
;        VSupply/2 at t1 and below VSupply/2w at t2 then no speed   *
;        adjustment is made. If BEMF is high at both t1 and t2 then  *
;        the speed is reduced. If BEMF is low at t1 and t2 then the  *
;        speed is increased.                                        *
;                                                       *
;*****************************************************************
;
;
        list P = PIC16F877
        include "p16f877.inc"
        __CONFIG _CP_OFF & _WRT_ENABLE_OFF & _HS_OSC &
_WDT_OFF & _PWRTE_ON & _BODEN_ON

; Acceleration/Deceleration Time = RampRate * 256 * 256 * Timer 0
prescale / Fosc
#define             AccelDelay  D'255'      ; determines full range
acceleration time
#define             DecelDelay  D'200'      ; determines full range
deceleration time
#define             ManThresh   0x3f        ; Manual threshold is
the PWM potentiomenter
                ; reading above which RPM is adjusted automatically
```

```
#define             AutoThresh  0x100-ManThresh
OffMask     equ     B'11010101' ; PWM off kills the high drives
Invalidequ  B'00000000' ; invalid
Phase1      equ     B'00100001' ; phase 1 C high, A low
Phase2      equ     B'00100100' ; phase 2 C high, B low
Phase3      equ     B'00000110' ; phase 3 A high, B low
Phase4      equ     B'00010010' ; phase 4 A high, C low
Phase5      equ     B'00011000' ; phase 5 B high, C low
Phase6      equ     B'00001001' ; phase 6 B high, A low

#define     CARRY       STATUS,C
#define ZERO        STATUS,Z
#define subwl       sublw

;*****************************************************************************
;*
;*      Define I/O Ports
;*

#define     ReadIndicator       PORTB,0             ; diagnostic
scope trigger for BEMF readings
#define DrivePort   PORTC               ; motor drive and lock status

;*****************************************************************************
;*
;*      Define RAM variables
;*

        CBLOCK 0x20
        STATE               ; Machine state
        PWMThresh ; PWM threshold
        PhaseIndx   ; Current motor phase index
        Drive       ; Motor drive word
        RPMIndex    ; RPM Index workspace
        ADCRPM              ; ADC RPM value
        ADCOffset   ; Delta offset to ADC PWM threshold
        PresetHi    ; speed control timer compare MS byte
        PresetLo    ; speed control timer compare LS byte
        Flags       ; general purpose flags
        Vsupply             ; Supply voltage ADC reading
        DeltaV1             ; Difference between expected and actual
BEMF at T/4
```

```
        DeltaV2             ; Difference between expected and actual
BEMF at T/2
        CCPSaveH    ; Storage for phase time when finding DeltaV
        CCPSaveL    ; Storage for phase time when finding DeltaV
        CCPT2H              ; Workspace for determining T/2 and T/4
        CCPT2L              ; Workspace for determining T/2 and T/4
        RampTimer   ; Timer 0 post scaler for accel/decel ramp rate
        xCount              ; general purpose counter workspace
        Status          ; relative speed indicator status

        ENDC

;*****************************************************************************
*
;*
;*      Define Flags
;*

#define DriveOnFlag         Flags,0             ; Flag for invoking drive
disable mask when clear
#define AutoRPM             Flags,1             ; RPM timer is adjusted
automatically
;                   Flags,3             ; Undefined
#define FullOnFlag  Flags,4             ; PWM threshold is set to
maximum drive
#define Tmr0Ovf             Flags,5             ; Timer 0 overflow flag
#define Tmr0Sync    Flags,6             ; Second Timer 0 overflow flag
;                   Flags,7             ; undefined

#define     BEMF1Low    DeltaV1,7   ; BEMF1 is low if DeltaV1 is
negative
#define     BEMF2Low    DeltaV2,7   ; BEMF2 is low if DeltaV2 is
negative

;*****************************************************************************
*
;*
;*      Define State machine states and index numbers
;*

sRPMSetup   equ     D'0'            ; Wait for Phase1, Set ADC GO, RA1-
>ADC
sRPMRead    equ     sRPMSetup+1     ; Wait for ADC nDONE, Read
ADC->RPM
```

```
sOffsetSetup equ    sRPMRead+1      ; Wait for Phase2, Set ADC
GO, RA3->ADC
sOffsetRead equ     sOffsetSetup+1  ; Wait for ADC nDONE, Read
ADC->ADCOffset
sVSetup             equ     sOffsetRead+1   ; Wait for Phase4, Drive
On, wait 9 uSec, Set ADC GO
sVIdle      equ     sVSetup+1   ; Wait for Drive On, wait Tacq, set
ADC GO
sVRead              equ     sVIdle+1    ; Wait for ADC nDONE, Read
ADC->Vsupply
sBEMFSetupequ       sVRead+1    ; Wait for Phase5, set Timer1
compare to half phase time
sBEMFIdle   equ     sBEMFSetup+1    ; Wait for Timer1 compare,
Force Drive on and wait 9 uSec,
                                    ; Set ADC GO, RA0->ADC
sBEMFRead equ       sBEMFIdle+1     ; Wait for ADC nDONE, Read
ADC->Vbemf
sBEMF2Idle  equ     sBEMFRead+1     ; Wait for Timer1 compare,
Force Drive on and wait 9 uSec,
                                    ; Set ADC GO, RA0->ADC
sBEMF2Read          equ     sBEMF2Idle+1    ; Wait for ADC nDONE,
Read ADC->Vbemf

;*************************************************************************
*
;*
;*      The ADC input is changed depending on the STATE
;*      Each STATE assumes a previous input selection and changes
the selection
;*      by XORing the control register with the appropriate ADC input
change mask
;*      defined here:
;*

ADC0to1             equ     B'00001000' ; changes ADCON0<5:3> from
000 to 001
ADC1to3             equ     B'00010000' ; changes ADCON0<5:3> from
001 to 011
ADC3to0             equ     B'00011000' ; changes ADCON0<5:3> from
011 to 000

;*************************************************************************
*
```

```
;*************************** PROGRAM STARTS HERE
*****************************
;*******************************************************************************
*
   org 0x000
        nop
        goto   Initialize

        org    0x004
   goto   SVR_timer0

   org 0x08

SVR_timer0
        bsf    Tmr0Ovf              ; Timer 0 overflow flag used by
accel/decel timer
        bsf    Tmr0Sync    ; Timer 0 overflow flag used to synchronize
code execution
        bcf    INTCON,T0IF
        retfie                      ;

Initialize
        clrf   PORTC                ; all drivers off
        clrf   PORTB

        banksel TRISA
; setup I/O
        clrf   TRISC                ; motor drivers on PORTC
        movlw  B'00001011' ; A/D on RA0 (PWM), RA1 (Speed) and RA3
(BEMF)
        movwf  TRISA        ;
        movlw  B'11111110' ; RB0 is locked indicator
        movwf  TRISB
; setup Timer0
        ;movlw        B'11010000' ; Timer0: Fosc, 1:2
   movlw   B'11010111'    ; Timer0: Fosc, 1:2
        movwf  OPTION_REG
        bsf    INTCON,T0IE        ; enable timer 0 interrupts
; Setup ADC
        movlw  B'00000100' ; ADC left justified, AN0, AN1
        movwf  ADCON1

        banksel        PORTA
        movlw  B'10000001' ; ADC clk = Fosc/32, AN0, ADC on
```

```
        movwf   ADCON0
; setup Timer 1
        movlw   B'00100001' ; 1:4 prescale, internal clock, timer on
        movwf   T1CON
; setup Timer 1 compare
        movlw   0xFF            ; set compare to maximum count
        movwf   CCPR1L                  ; LS compare register
        movwf   CCPR1H                  ; MS compare register
        movlw   B'00001011' ; Timer 1 compare mode, special event -
clears timer1
        movwf   CCP1CON

; initialize RAM

        clrf    PWMThresh
        movlw   D'6'
        movwf   PhaseIndx
        clrf    Flags
        clrf    Status          ;
        clrf    STATE                   ; LoopIdle->STATE
        bcf     INTCON,T0IF             ; ensure timer 0 overflow flag is
cleared
        bsf     INTCON,GIE; enable interrupts

MainLoop
;*****************************************************************
;
;       PWM, Commutation, State machine loop
;
;*****************************************************************

        btfsc   PIR1,CCP1IF             ; time for phase change?
        call    Commutate   ; yes - change motor drive
PWM
        bsf     DriveOnFlag ; pre-set flag
        btfsc   FullOnFlag    ; is PWM level at maximum?
        goto    PWM02                   ; yes - only commutation is necessary

        movf    PWMThresh,w             ; get PWM threshold
        addwf   TMR0,w                  ; compare to timer 0
        btfss   CARRY                   ; drive is on if carry is set
        bcf     DriveOnFlag ; timer has not reached threshold, disable
drive
```

```
        call    DriveMotor    ; output drive word
PWM02
        call    LockTest
        call    StateMachine        ; service state machine
        goto    MainLoop    ; repeat loop

StateMachine
        movlw SMTableEnd-SMTable-1   ; STATE table must have 2^n
entries
        andwf STATE,f               ; limit STATE index to state table
        movlw high SMTable          ; get high byte of table address
        movwf PCLATH                ; prepare for computed goto
        movlw low SMTable; get low byte of table address
        addwf STATE,w               ; add STATE index to table root
        btfsc   CARRY               ; test for page change in table
        incf    PCLATH,f    ; page change adjust
        movwf PCL           ; jump into table

SMTable                             ; number of STATE table entries
MUST be evenly divisible by 2
        goto    RPMSetup    ; Wait for Phase1, Set ADC GO, RA1-
>ADC, clear timer0 overflow
        goto    RPMRead             ; Wait for ADC nDONE, Read ADC-
>RPM
        goto    OffsetSetup  ; Wait for Phase2, Set ADC GO, RA3->ADC
        goto    OffsetRead   ; Wait for ADC nDONE, Read ADC-
>ADCOffset
        goto    VSetup              ; Wait for Phase4
        goto    VIdle           ; Wait for Drive On, wait Tacq, set ADC GO

        goto    VRead               ; Wait for ADC nDONE, Read ADC-
>Vsupply
        goto    BEMFSetup  ; Wait for Phase5, set Timer1 compare to
half phase time
        goto    BEMFIdle     ; When Timer1 compares force Drive on,
Set ADC GO after Tacq, RA0->ADC
        goto    BEMFRead   ; Wait for ADC nDONE, Read ADC->Vbemf
        goto    BEMF2Idle   ; When Timer1 compares force Drive on,
Set ADC GO after Tacq, RA0->ADC
        goto    BEMF2Read ; Wait for ADC nDONE, Read ADC->Vbemf

; fill out table with InvalidStates to make number of table entries evenly
divisible by 2
```

```
        goto    InvalidState  ; invalid state - reset state machine
        goto    InvalidState  ; invalid state - reset state machine
        goto    InvalidState  ; invalid state - reset state machine
        goto    InvalidState  ; invalid state - reset state machine
SMTableEnd

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RPMSetup                ; Wait for Phase1, Set ADC GO, RA1-
>ADC, clear timer0 overflow

        movlw Phase1            ; compare Phase1 word...
        xorwf  Drive,w          ; ...with current drive word
        btfss  ZERO       ; ZERO if equal
        return            ; not Phase1 - remain in current STATE

        bsf     ADCON0,GO       ; start ADC
        movlw ADC0to1           ; prepare to change ADC input
        xorwf  ADCON0,f   ; change from AN0 to AN1
        incf    STATE,f         ; next STATE
        bcf     Tmr0Sync  ; clear timer0 overflow
        return            ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RPMRead                         ; Wait for ADC nDONE, Read ADC-
>RPM

        btfsc   ADCON0,GO       ; is ADC conversion finished?
        return            ; no - remain in current STATE

        movf    ADRESH,w  ; get ADC result
        movwf ADCRPM            ; save in RPM

        incf    STATE,f         ; next STATE
        return            ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OffsetSetup             ; Wait for Phase2, Set ADC GO, RA3->ADC

        movlw Phase2            ; compare Phase2 word...
        xorwf  Drive,w          ; ...with current drive word
        btfss  ZERO       ; ZERO if equal
```

```
        return                  ; not Phase2 - remain in current STATE

        bsf     ADCON0,GO       ; start ADC
        movlw   ADC1to3         ; prepare to change ADC input
        xorwf   ADCON0,f        ; change from AN1 to AN3
        incf    STATE,f         ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OffsetRead              ; Wait for ADC nDONE, Read ADC-
>ADCOffset

        btfsc   ADCON0,GO       ; is ADC conversion finished?
        return                  ; no - remain in current STATE

        movf    ADRESH,w        ; get ADC result
        xorlw   H'80'           ; complement MSB for +/- offset
        movwf   ADCOffset       ; save in offset
        addwf   ADCRPM,w        ; add offset to PWM result
        btfss   ADCOffset,7     ; is offset a negative number?
        goto    OverflowTest    ; no - test for overflow

        btfss   CARRY           ; underflow?
        andlw   H'00'           ; yes - force minimum
        goto    Threshold       ;

OverflowTest
        btfsc   CARRY           ; overflow?
        movlw   H'ff'           ; yes - force maximum

Threshold
        movwf   PWMThresh       ; PWM threshold is RPM result plus offset
        btfsc   ZERO            ; is drive off?
        goto    DriveOff        ; yes - skip voltage measurements

        bcf     FullOnFlag      ; pre-clear flag in preparation of compare
        sublw   0xFD            ; full on threshold
        btfss   CARRY           ; CY = 0 if PWMThresh > FullOn
        bsf     FullOnFlag      ; set full on flag
        incf    STATE,f         ; next STATE
        return                  ; back to Main Loop

DriveOff
```

```
        clrf    Status          ; clear speed indicators
        movlw   B'11000111'     ; reset ADC input to AN0
        andwf   ADCON0,f        ;
        clrf    STATE                   ; reset state machine
        return

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VSetup                                  ; Wait for Phase4

        movlw   Phase4                  ; compare Phase4 word...
        xorwf   Drive,w                 ; ...with current Phase drive word
        btfss   ZERO            ; ZERO if equal
        return                  ; not Phase4 - remain in current STATE

        call    SetTimer        ; set timer value from RPM table
        incf    STATE,f                 ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VIdle                                   ; Wait for Drive On, wait Tacq, set ADC GO


        btfss   DriveOnFlag ; is Drive active?
        return                  ; no - remain in current STATE

        bsf     ReadIndicator           ; Diagnostic
        call    Tacq            ; motor Drive is active - wait ADC Tacq time
        bsf     ADCON0,GO               ; start ADC
        bcf     ReadIndicator           ; Diagnostic
        incf    STATE,f                 ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VRead                                   ; Wait for ADC nDONE, Read ADC-
>Vsupply

        btfsc   ADCON0,GO               ; is ADC conversion finished?
        return                  ; no - remain in current STATE

        movf    ADRESH,w        ; get ADC result
        movwf   Vsupply                 ; save as supply voltage
```

```
        incf    STATE,f             ; next STATE
        bcf     Tmr0Sync    ; clear timer 0 overflow
        return              ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMFSetup               ; Wait for Phase5, set Timer1 compare to
half phase time

        movlw   Phase5              ; compare Phase5 word...
        xorwf   Drive,w             ; ...with current drive word
        btfss   ZERO        ; ZERO if equal
        return              ; not Phase5 - remain in current STATE

        btfss   Tmr0Sync    ; synchronize with timer 0
        return              ;

        btfss   PWMThresh,7         ; if PWMThresh > 0x80 then ON is
longer than OFF
        goto    BEMFS1              ; OFF is longer and motor is currently
off - compute now

        btfss   DriveOnFlag ; ON is longer - wait for drive cycle to start
        return              ; not started - wait

BEMFS1
        bcf     CCP1CON,0 ; disable special event on compare
        movf    CCPR1H,w  ; save current capture compare state
        movwf   CCPSaveH  ;
        movwf   CCPT2H              ; save copy in workspace
        movf    CCPR1L,w  ; low byte
        movwf   CCPSaveL  ; save
        movwf   CCPT2L              ; and save copy
        bcf     CARRY               ; pre-clear carry for rotate
        rrf     CCPT2H,f  ; divide phase time by 2
        rrf     CCPT2L,f  ;
        bcf     CARRY               ; pre-clear carry
        rrf     CCPT2H,w  ; divide phase time by another 2
        movwf   CCPR1H              ; first BEMF reading at phase T/4
        rrf     CCPT2L,w  ;
        movwf   CCPR1L              ;

        incf    STATE,f             ; next STATE
        return              ; back to Main Loop
```

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMFIdle                ; When Timer1 compares force Drive on,
Set ADC GO after Tacq, RA0->ADC

        btfss   PIR1,CCP1IF     ; timer compare?
        return                  ; no - remain in current STATE

        bsf     DriveOnFlag ; force drive on for BEMF reading
        call    DriveMotor  ; activate motor drive
        bsf     ReadIndicator       ; Diagnostic
        call    Tacq        ; wait ADC acqisition time
        bsf     ADCON0,GO       ; start ADC
        bcf     ReadIndicator       ; Diagnostic

; setup to capture BEMF at phase 3/4 T

        movf    CCPT2H,w
        addwf CCPR1H,f  ; next compare at phase 3/4 T
        movf    CCPT2L,w    ;
        addwf CCPR1L,f  ; set T/2 lsb
        btfsc   CARRY               ; test for carry into MSb
        incf    CCPR1H,f    ; perform carry
        bcf     PIR1,CCP1IF     ; clear timer compare interrupt flag
        incf    STATE,f             ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMFRead                ; Wait for ADC nDONE, Read ADC->Vbemf

        btfsc   ADCON0,GO       ; is ADC conversion finished?
        return                  ; no - remain in current STATE

        rrf     Vsupply,w   ; divide supply voltage by 2
        subwf ADRESH,w  ; Vbemf - Vsupply/2

        movwfDeltaV1            ; save error voltage
        incf    STATE,f             ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
BEMF2Idle               ; When Timer1 compares force Drive on,
Set ADC GO after Tacq, RA0->ADC

        btfss   PIR1,CCP1IF     ; timer compare?
        return                  ; no - remain in current STATE

        bsf     DriveOnFlag ; force drive on for BEMF reading
        call    DriveMotor  ; activate motor drive
        bsf     ReadIndicator       ; Diagnostic
        call    Tacq        ; wait ADC acqisition time
        bsf     ADCON0,GO       ; start ADC
        bcf     ReadIndicator       ; Diagnostic
        movlw   ADC3to0             ; prepare to change ADC input
        xorwf   ADCON0,f    ; change from AN3 to AN0

; restore Timer1 phase time and special event compare mode

        movf    CCPSaveH,w
        movwf   CCPR1H              ; next compare at phase T
        movf    CCPSaveL,w      ;
        movwf   CCPR1L              ; set T lsb
        bcf     PIR1,CCP1IF     ; clear timer compare interrupt flag
        bsf     CCP1CON,0 ; enable special event on compare
        incf    STATE,f             ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMF2Read               ; Wait for ADC nDONE, Read ADC->Vbemf

        btfsc   ADCON0,GO       ; is ADC conversion finished?
        return                  ; no - remain in current STATE

        rrf     Vsupply,w   ; divide supply voltage by 2
        subwf   ADRESH,w    ; Vbemf - Vsupply/2

        movwf   DeltaV2             ; save error voltage

        clrf    STATE               ; reset state machine to beginning
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
InvalidState            ; trap for invalid STATE index
```

```
        movlw   B'11000111' ; reset ADC input to AN0
        andwf   ADCON0,f    ;
        clrf    STATE
        return
;_____________________________________________________________
_____________________________________

Tacq
;*****************************************************************
;
;       Software delay for ADC acquisition time
;       Delay time = Tosc*(3+3*xCount)
;
;*****************************************************************

        movlw   D'14'           ; 14 equates to approx 9 uSec delay
        movwf   xCount          ;
        decfsz  xCount,f        ;
        goto    $-1             ; loop here until time complete
        return

LockTest
;*****************************************************************
;
;       T is the commutation phase period. Back EMF is measured on
the
;       floating motor terminal at two times during T to determine
;       the approximate zero crossing of the BEMF. BEMF low means
that
;       the measured BEMF is below (supply voltage)/2.
;       If BEMF is low at 1/4 T then accelerate.
;       If BEMF is high at 1/4 T and low at 3/4 T then speed is OK.
;       If BEMF is high at 1/4 T and 3/4 T then decelerate.
;
;       Lock test computation is synchronized to the PWM clock such
;       that the computation is performed during the PWM ON or OFF
;       time whichever is longer.
;
;*****************************************************************

; synchronize test with start of timer 0

        btfss   Tmr0Ovf             ; has timer 0 wrapped around?
        return                  ; no - skip lock test
```

```
        btfss   PWMThresh,7         ; if PWMThresh > 0x80 then ON is
longer than OFF
        goto    LT05            ; OFF is longer and motor is currently off -
compute now

        btfss   DriveOnFlag ; ON is longer - wait for drive cycle to start
        return                  ; not started - wait

LT05
        bcf     Tmr0Ovf                 ; clear synchronization flag
        decfsz  RampTimer,f ; RampTimer controls the
acceleration/deceleration rate
        return

; use lock results to control RPM only if not manual mode

        bsf     AutoRPM                 ; preset flag
        movf    ADCRPM,w    ; compare RPM potentiometer...
        addlw   AutoThresh  ; ...to the auto control threshold
        btfss   CARRY                   ; CARRY is set if RPM is > auto
threshold
        bcf     AutoRPM                 ; not in auto range - reset flag

        btfss   BEMF1Low    ; is first BEMF below Supply/2
        goto    LT20            ; no - test second BEMF

LT10
; accelerate if BEMF at 1/4 T is below Supply/2

        movlw   B'10000000' ; indicate lock test results
        movwf   Status          ; status is OR'd with drive word later
        movlw   AccelDelay  ; set the timer for acceleration delay
        movwf   RampTimer   ;

        btfss   AutoRPM                 ; is RPM in auto range?
        goto    ManControl  ; no - skip RPM adjustment

        incfsz  RPMIndex,f  ; increment the RPM table index
        return                  ; return if Index didn't wrap around

        decf    RPMIndex,f  ; top limit is 0xFF
        return
LT20
```

```
        btfsc   BEMF2Low   ; BEMF1 was high...
        goto    ShowLocked ; ... and BEMF2 is low - show locked

; decelerate if BEMF at 3/4 T is above Supply/2

        movlw   B'01000000' ; indicate lock test results
        movwf   Status      ; status is OR'd with drive word later
        movlw   DecelDelay  ; set the timer for deceleration delay
        movwf   RampTimer   ;

        btfss   AutoRPM             ; is RPM in auto range?
        goto    ManControl  ; no - skip RPM adjustment

        decfsz  RPMIndex,f  ; set next lower RPM table index
        return              ; return if index didn't wrap around

        incf    RPMIndex,f  ; bottom limit is 0x01
        return

ShowLocked
        movlw   B'11000000' ; indicate lock test results
        movwf   Status      ; status is OR'd with drive word later
        movlw   DecelDelay  ; set the timer for deceleration delay
        movwf   RampTimer   ;

        btfsc   AutoRPM             ; was RPM set automatically?
        return              ; yes - we're done

ManControl
        movf    ADCRPM,w    ; get RPM potentiometer reading...
        movwf   RPMIndex    ; ...and set table index directly
        return

Commutate
;*****************************************************************
;
;       Commutation is triggered by PIR1<CCP1IF> flag.
;       This flag is set when timer1 equals the compare register.
;       When BEMF measurement is active the compare time is not
;       cleared automatically (special event trigger is off).
;     Ignore the PIR1<CCP1IF> flag when special trigger is off
;       because the flag is for BEMF measurment.
;       If BEMF measurement is not active then decrement phase table
;       index and get the drive word from the table. Save the
```

```
;       drive word in a global variable and output to motor drivers.
;
;*****************************************************************

        btfss   CCP1CON,0 ; is special event on compare enabled?
        return            ; no - this is a BEMF measurment, let state
machine handle this

        bcf     PIR1,CCP1IF       ; clear interrupt flag

        movlw   high OnTable      ; set upper program counter bits
        movwf   PCLATH
        decfsz  PhaseIndx,w ; decrement to next phase
        goto    $+2               ; skip reset if not zero
        movlw   D'6'              ; phase counts 6 to 1
        movwf   PhaseIndx   ; save the phase index
        addlw   LOW OnTable
        btfsc   CARRY             ; test for possible page boundry
        incf    PCLATH,f    ; page boundry adjust
        call    GetDrive
        movwf   Drive             ; save motor drive word
DriveMotor
        movf    Drive,w           ; restore motor drive word
        btfss   DriveOnFlag ; test drive enable flag
        andlw   OffMask           ; kill high drive if PWM is off
        iorwf   Status,w    ; show speed indicators
        movwf   DrivePort   ; output to motor drivers
        return

GetDrive
        movwf   PCL               ; computed goto
OnTable
        retlw   Invalid
        retlw   Phase6
        retlw   Phase5
        retlw   Phase4
        retlw   Phase3
        retlw   Phase2
        retlw   Phase1
        retlw   Invalid

SetTimer
;*****************************************************************
;
```

```
;       This sets the CCP module compare registers for timer 1.
;       The motor phase period is the time it takes timer 1
;       to count from 0 to the compare value. The CCP module
;       is configured to clear timer 1 when the compare occurs.
;       Get the timer1 compare variable from two lookup tables, one
;       for the compare high byte and the other for the low byte.
;
;*****************************************************************

        call    SetTimerHigh
        movwf   CCPR1H                  ; Timer1 High byte preset
        call    SetTimerLow
        movwf   CCPR1L                  ; Timer1 Low byte preset
        return

SetTimerHigh
        movlw   high T1HighTable    ; lookup preset values
        movwf   PCLATH                  ; high bytes first
        movlw   low T1HighTable         ;
        addwf   RPMIndex,w          ; add table index
        btfsc   STATUS,C            ; test for table page crossing
        incf    PCLATH,f            ;
        movwf   PCL                 ; lookup - result returned in W
SetTimerLow
        movlw   high T1LowTable         ; repeat for lower byte
        movwf   PCLATH                  ;
        movlw   low T1LowTable          ;
        addwf   RPMIndex,w          ; add table index
        btfsc   STATUS,C            ; test for table page crossing
        incf    PCLATH,f            ;
        movwf   PCL                 ; lookup - result returned in W

#include "BLDCspd4.inc"

        end
;_____________________________________________________
     LIST
; P16F877.INC  Standard Header File, Version 1.00    Microchip
Technology, Inc.
      NOLIST

; This header file defines configurations, registers, and other useful bits
of
```

```
; information for the PIC16F877 microcontroller.  These names are
taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16F877
;       2. LIST directive in the source file
;               LIST   P=PIC16F877
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================
;
;       Revision History
;
;==========================================================

;Rev:   Date:    Reason:

;1.12   01/12/00 Changed some bit names, a register name,
configuration bits
;               to match datasheet (DS30292B)
;1.00   08/07/98 Initial Release

;==========================================================
;
;       Verify Processor
;
;==========================================================

       IFNDEF __16F877
         MESSG "Processor-header file mismatch.  Verify selected
processor."
       ENDIF

;==========================================================
;
```

```
;       Register Definitions
;
;==================================================
====================

W                       EQU     H'0000'
F                       EQU     H'0001'

;----- Register Files------------------------------------------------------

INDF                    EQU     H'0000'
TMR0                    EQU     H'0001'
PCL                     EQU     H'0002'
STATUS                  EQU     H'0003'
FSR                     EQU     H'0004'
PORTA                   EQU     H'0005'
PORTB                   EQU     H'0006'
PORTC                   EQU     H'0007'
PORTD                   EQU     H'0008'
PORTE                   EQU     H'0009'
PCLATH                  EQU     H'000A'
INTCON                  EQU     H'000B'
PIR1                    EQU     H'000C'
PIR2                    EQU     H'000D'
TMR1L                   EQU     H'000E'
TMR1H                   EQU     H'000F'
T1CON                   EQU     H'0010'
TMR2                    EQU     H'0011'
T2CON                   EQU     H'0012'
SSPBUF                  EQU     H'0013'
SSPCON                  EQU     H'0014'
CCPR1L                  EQU     H'0015'
CCPR1H                  EQU     H'0016'
CCP1CON                 EQU     H'0017'
RCSTA                   EQU     H'0018'
TXREG                   EQU     H'0019'
RCREG                   EQU     H'001A'
CCPR2L                  EQU     H'001B'
CCPR2H                  EQU     H'001C'
CCP2CON                 EQU     H'001D'
ADRESH                  EQU     H'001E'
ADCON0                  EQU     H'001F'

OPTION_REG              EQU     H'0081'
```

```
TRISA                EQU     H'0085'
TRISB                EQU     H'0086'
TRISC                EQU     H'0087'
TRISD                EQU     H'0088'
TRISE                EQU     H'0089'
PIE1                 EQU     H'008C'
PIE2                 EQU     H'008D'
PCON                 EQU     H'008E'
SSPCON2              EQU     H'0091'
PR2                  EQU     H'0092'
SSPADD               EQU     H'0093'
SSPSTAT              EQU     H'0094'
TXSTA                EQU     H'0098'
SPBRG                EQU     H'0099'
ADRESL               EQU     H'009E'
ADCON1               EQU     H'009F'

EEDATA               EQU     H'010C'
EEADR                EQU     H'010D'
EEDATH               EQU     H'010E'
EEADRH               EQU     H'010F'

EECON1               EQU     H'018C'
EECON2               EQU     H'018D'

;----- STATUS Bits --------------------------------------------------------

IRP                  EQU     H'0007'
RP1                  EQU     H'0006'
RP0                  EQU     H'0005'
NOT_TO               EQU     H'0004'
NOT_PD               EQU     H'0003'
Z                    EQU     H'0002'
DC                   EQU     H'0001'
C                    EQU     H'0000'

;----- INTCON Bits --------------------------------------------------------

GIE                  EQU     H'0007'
PEIE                 EQU     H'0006'
T0IE                 EQU     H'0005'
INTE                 EQU     H'0004'
RBIE                 EQU     H'0003'
T0IF                 EQU     H'0002'
```

```
INTF                    EQU     H'0001'
RBIF                    EQU     H'0000'

;----- PIR1 Bits --------------------------------------------------------

PSPIF                   EQU     H'0007'
ADIF                    EQU     H'0006'
RCIF                    EQU     H'0005'
TXIF                    EQU     H'0004'
SSPIF                   EQU     H'0003'
CCP1IF                  EQU     H'0002'
TMR2IF                  EQU     H'0001'
TMR1IF                  EQU     H'0000'

;----- PIR2 Bits --------------------------------------------------------

EEIF                    EQU     H'0004'
BCLIF                   EQU     H'0003'
CCP2IF                  EQU     H'0000'

;----- T1CON Bits -------------------------------------------------------

T1CKPS1                 EQU     H'0005'
T1CKPS0                 EQU     H'0004'
T1OSCEN                 EQU     H'0003'
NOT_T1SYNC              EQU     H'0002'
T1INSYNC                EQU     H'0002'    ; Backward compatibility
only
T1SYNC                  EQU     H'0002'
TMR1CS                  EQU     H'0001'
TMR1ON                  EQU     H'0000'

;----- T2CON Bits -------------------------------------------------------

TOUTPS3                 EQU     H'0006'
TOUTPS2                 EQU     H'0005'
TOUTPS1                 EQU     H'0004'
TOUTPS0                 EQU     H'0003'
TMR2ON                  EQU     H'0002'
T2CKPS1                 EQU     H'0001'
T2CKPS0                 EQU     H'0000'

;----- SSPCON Bits ------------------------------------------------------
```

```
WCOL                    EQU     H'0007'
SSPOV                   EQU     H'0006'
SSPEN                   EQU     H'0005'
CKP                     EQU     H'0004'
SSPM3                   EQU     H'0003'
SSPM2                   EQU     H'0002'
SSPM1                   EQU     H'0001'
SSPM0                   EQU     H'0000'

;----- CCP1CON Bits --------------------------------------------------------

CCP1X                   EQU     H'0005'
CCP1Y                   EQU     H'0004'
CCP1M3                  EQU     H'0003'
CCP1M2                  EQU     H'0002'
CCP1M1                  EQU     H'0001'
CCP1M0                  EQU     H'0000'

;----- RCSTA Bits ----------------------------------------------------------

SPEN                    EQU     H'0007'
RX9                     EQU     H'0006'
RC9                     EQU     H'0006'    ; Backward compatibility only
NOT_RC8                 EQU     H'0006'    ; Backward compatibility
only
RC8_9                   EQU     H'0006'    ; Backward compatibility only
SREN                    EQU     H'0005'
CREN                    EQU     H'0004'
ADDEN                   EQU     H'0003'
FERR                    EQU     H'0002'
OERR                    EQU     H'0001'
RX9D                    EQU     H'0000'
RCD8                    EQU     H'0000'    ; Backward compatibility only

;----- CCP2CON Bits --------------------------------------------------------

CCP2X                   EQU     H'0005'
CCP2Y                   EQU     H'0004'
CCP2M3                  EQU     H'0003'
CCP2M2                  EQU     H'0002'
CCP2M1                  EQU     H'0001'
CCP2M0                  EQU     H'0000'

;----- ADCON0 Bits ---------------------------------------------------------
```

```
ADCS1                   EQU     H'0007'
ADCS0                   EQU     H'0006'
CHS2                    EQU     H'0005'
CHS1                    EQU     H'0004'
CHS0                    EQU     H'0003'
GO                      EQU     H'0002'
NOT_DONE                EQU     H'0002'
GO_DONE                 EQU     H'0002'
ADON                    EQU     H'0000'

;----- OPTION_REG Bits --------------------------------------------------

NOT_RBPU                EQU     H'0007'
INTEDG                  EQU     H'0006'
T0CS                    EQU     H'0005'
T0SE                    EQU     H'0004'
PSA                     EQU     H'0003'
PS2                     EQU     H'0002'
PS1                     EQU     H'0001'
PS0                     EQU     H'0000'

;----- TRISE Bits ------------------------------------------------------

IBF                     EQU     H'0007'
OBF                     EQU     H'0006'
IBOV                    EQU     H'0005'
PSPMODE                 EQU     H'0004'
TRISE2                  EQU     H'0002'
TRISE1                  EQU     H'0001'
TRISE0                  EQU     H'0000'

;----- PIE1 Bits -------------------------------------------------------

PSPIE                   EQU     H'0007'
ADIE                    EQU     H'0006'
RCIE                    EQU     H'0005'
TXIE                    EQU     H'0004'
SSPIE                   EQU     H'0003'
CCP1IE                  EQU     H'0002'
TMR2IE                  EQU     H'0001'
TMR1IE                  EQU     H'0000'

;----- PIE2 Bits -------------------------------------------------------
```

```
EEIE                    EQU     H'0004'
BCLIE                   EQU     H'0003'
CCP2IE                  EQU     H'0000'

;----- PCON Bits --------------------------------------------------------

NOT_POR                 EQU     H'0001'
NOT_BO                  EQU     H'0000'
NOT_BOR                 EQU     H'0000'

;----- SSPCON2 Bits -----------------------------------------------------

GCEN                    EQU     H'0007'
ACKSTAT                 EQU     H'0006'
ACKDT                   EQU     H'0005'
ACKEN                   EQU     H'0004'
RCEN                    EQU     H'0003'
PEN                     EQU     H'0002'
RSEN                    EQU     H'0001'
SEN                     EQU     H'0000'

;----- SSPSTAT Bits -----------------------------------------------------

SMP                     EQU     H'0007'
CKE                     EQU     H'0006'
D                       EQU     H'0005'
I2C_DATA                EQU     H'0005'
NOT_A                   EQU     H'0005'
NOT_ADDRESS             EQU     H'0005'
D_A                     EQU     H'0005'
DATA_ADDRESS            EQU     H'0005'
P                       EQU     H'0004'
I2C_STOP                EQU     H'0004'
S                       EQU     H'0003'
I2C_START               EQU     H'0003'
R                       EQU     H'0002'
I2C_READ                EQU     H'0002'
NOT_W                   EQU     H'0002'
NOT_WRITE               EQU     H'0002'
R_W                     EQU     H'0002'
READ_WRITE              EQU     H'0002'
UA                      EQU     H'0001'
BF                      EQU     H'0000'
```

```
;----- TXSTA Bits ----------------------------------------------------

CSRC                    EQU     H'0007'
TX9                     EQU     H'0006'
NOT_TX8                 EQU     H'0006'    ; Backward compatibility only
TX8_9                   EQU     H'0006'    ; Backward compatibility only
TXEN                    EQU     H'0005'
SYNC                    EQU     H'0004'
BRGH                    EQU     H'0002'
TRMT                    EQU     H'0001'
TX9D                    EQU     H'0000'
TXD8                    EQU     H'0000'    ; Backward compatibility only

;----- ADCON1 Bits ---------------------------------------------------

ADFM                    EQU     H'0007'
PCFG3                   EQU     H'0003'
PCFG2                   EQU     H'0002'
PCFG1                   EQU     H'0001'
PCFG0                   EQU     H'0000'

;----- EECON1 Bits ---------------------------------------------------

EEPGD                   EQU     H'0007'
WRERR                   EQU     H'0003'
WREN                    EQU     H'0002'
WR                      EQU     H'0001'
RD                      EQU     H'0000'

;=========================================================
;
;       RAM Definition
;
;=========================================================
        __MAXRAM H'1FF'
        __BADRAM H'8F'-H'90', H'95'-H'97', H'9A'-H'9D'
        __BADRAM H'105', H'107'-H'109'
        __BADRAM H'185', H'187'-H'189', H'18E'-H'18F'
;=========================================================
;
```

```
;       Configuration Bits
;
;==========================================================
====================
_CP_ALL                 EQU     H'0FCF'
_CP_HALF                EQU     H'1FDF'
_CP_UPPER_256           EQU     H'2FEF'
_CP_OFF                 EQU     H'3FFF'
_DEBUG_ON               EQU     H'37FF'
_DEBUG_OFF              EQU     H'3FFF'
_WRT_ENABLE_ON          EQU     H'3FFF'
_WRT_ENABLE_OFF         EQU     H'3DFF'
_CPD_ON                 EQU     H'3EFF'
_CPD_OFF                EQU     H'3FFF'
_LVP_ON                 EQU     H'3FFF'
_LVP_OFF                EQU     H'3F7F'
_BODEN_ON               EQU     H'3FFF'
_BODEN_OFF              EQU     H'3FBF'
_PWRTE_OFF              EQU     H'3FFF'
_PWRTE_ON               EQU     H'3FF7'
_WDT_ON                 EQU     H'3FFF'
_WDT_OFF                EQU     H'3FFB'
_LP_OSC                 EQU     H'3FFC'
_XT_OSC                 EQU     H'3FFD'
_HS_OSC                 EQU     H'3FFE'
_RC_OSC                 EQU     H'3FFF'

        LIST
```

ANEXO VI
PROGRAMA CONTROLADOR DEL MOTOR BLDC CON SENSORES PARA SIMULACIÓN.

```
;*****************************************************************
;                                                               *
;   Filename:    SensBLDC.asm                                   *
;   Date:        11 Feb 2002                                    *
;   File Version:  1.0                                          *
;                                                               *
;   Author:      W.R.Brown                                      *
;   Company:     Microchip Technolgy Incorporated               *
;                                                               *
;                                                               *
;*****************************************************************
;                                                               *
;   Files required: p16f877.inc                                 *
;                                                               *
;                                                               *
;                                                               *
;*****************************************************************
;                                                               *
;   Notes: Sensored brushless motor control                     *
;                                                               *
;                                                               *
;                                                               *
;                                                               *
;*****************************************************************


        list    p=16f877        ; list directive to define processor
        #include <p16f877.inc>      ; processor specific variable
definitions
        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON &
_PWRTE_ON & _HS_OSC & _WRT_ENABLE_OFF & _LVP_ON &
_DEBUG_OFF & _CPD_OFF

;*****************************************************************
;*
;* Define variable storage
;*
        CBLOCK 0x20

        ADC             ; PWM threshold is ADC result
```

```
        LastSensor   ; last read motor sensor data
        DriveWord    ; six bit motor drive data
        ENDC
;*****************************************************************
;*
;* Define I/O
;*
#define       OffMask              B'11010101'
#define       DrivePort     PORTC
#define DrivePortTris       TRISC
#define       SensorMask B'00000111'
#define       SensorPort   PORTE
#define DirectionBit PORTA,1
;*****************************************************************
        org     0x000          ; startup vector
        nop                    ; required for ICD operation
        clrf    PCLATH          ; ensure page bits are cleared
        goto    Initialize     ; go to beginning of program
        ORG     0x004           ; interrupt vector location
        retfie                 ; return from interrupt
;*****************************************************************
;*
;* Initialize I/O ports and peripherals
;*
org 0x10

Initialize
        clrf     DrivePort      ; all drivers off

        banksel TRISA
; setup I/O
        clrf     DrivePortTris; set motor drivers as outputs
        movlw B'00000011' ; A/D on RA0, Direction on RA1, Motor
sensors on RE<2:0>
        movwf TRISA          ;
; setup Timer0
;       movlw B'11010000' ; Timer0: Fosc, 1:2
        movlw B'11010111' ; Timer0: Fosc, 1:2
        movwf OPTION_REG
; Setup ADC (bank1)
        movlw B'00001110' ; ADC left justified, AN0 only
        movwf ADCON1
        banksel       ADCON0
; setup ADC (bank0)
```

```
        movlw   B'11000001' ; ADC clock from int RC, AN0, ADC on
        movwf   ADCON0
        bsf     ADCON0,GO       ; start ADC
        clrf    LastSensor  ; initialize last sensor reading
        call    Commutate   ; determine present motor positon
        clrf    ADC         ; start speed control threshold at zero until
first ADC reading
;*********************************************************************
;*
;* Main control loop
;*
Loop
        call    ReadADC         ; get the speed control from the ADC
        incfsz  ADC,w           ; if ADC is 0xFF we're at full speed -
skip timer add
        goto    PWM         ; add timer 0 to ADC for PWM
        movf    DriveWord,w ; force on condition
        goto    Drive       ; continue
PWM
        movf    ADC,w           ; restore ADC reading
        addwf   TMR0,w          ; add it to current timer0
        movf    DriveWord,w ; restore commutation drive data
        btfss   STATUS,C    ; test if ADC + timer0 resulted in carry
        andlw   OffMask         ; no carry - supress high drivers
Drive
        movwf   DrivePort   ; enable motor drivers
        call    Commutate   ; test for commutation change
        goto    Loop        ; repeat loop

ReadADC
;*********************************************************************
;*
;* If the ADC is ready then read the speed control potentiometer
;* and start the next reading
;*
        btfsc   ADCON0,NOT_DONE     ; is ADC ready?
        return              ; no - return

        movf    ADRESH,w    ; get ADC result
        bsf     ADCON0,GO       ; restart ADC
        movwf   ADC         ; save result in speed control threshold
        return              ;

;*********************************************************************
```

```
;*
;* Read the sensor inputs and if a change is sensed then get the
;* corresponding drive word from the drive table
;*
Commutate
        movlw   SensorMask ; retain only the sensor bits
        andwf   SensorPort,w        ; get sensor data
        xorwf   LastSensor,w        ; test if motion sensed
        btfsc   STATUS,Z    ; zero if no change
        return              ; no change - back to the PWM loop

        xorwf   LastSensor,f ; replace last sensor data with current
        btfss   DirectionBit  ; test direction bit
        goto    FwdCom              ; bit is zero - do forward commutation

                                    ; reverse commutation
        movlw   HIGH RevTable       ; get MS byte of table
        movwf   PCLATH              ; prepare for computed GOTO
        movlw   LOW RevTable        ; get LS byte of table
        goto    Com2
FwdCom                              ; forward commutation
        movlw   HIGH FwdTable       ; get MS byte of table
        movwf   PCLATH              ; prepare for computed GOTO
        movlw   LOW FwdTable        ; get LS byte of table
Com2
        addwf   LastSensor,w        ; add sensor offset
        btfsc   STATUS,C    ; page change in table?
        incf    PCLATH,f    ; yes - adjust MS byte

        call    GetDrive    ; get drive word from table
        movwf   DriveWord   ; save as current drive word
        return

GetDrive
        movwf   PCL
;*****************************************************************
;*
;* The drive tables are built based on the following assumptions:
;* 1) There are six drivers in three pairs of two
;* 2) Each driver pair consists of a high side (+V to motor) and low side
(motor to ground) drive
;* 3) A 1 in the drive word will turn the corresponding driver on
;* 4) The three driver pairs correspond to the three motor windings: A,
B and C
```

```
;* 5) Winding A is driven by bits <1> and <0> where <1> is A's high
side drive
;* 6) Winding B is driven by bits <3> and <2> where <3> is B's high
side drive
;* 7) Winding C is driven by bits <5> and <4> where <5> is C's high
side drive
;* 8) Three sensor bits consitute the address offset to the drive table
;* 9) A sensor bit transitions from a 0 to 1 at the moment that the
corresponding
;*    winding's high side forward drive begins.
;* 10) Sensor bit <0> corresponds to winding A
;* 11) Sensor bit <1> corresponds to winding B
;* 12) Sensor bit <2> corresponds to winding C
;*
FwdTable
        retlw   B'00000000' ; invalid
        retlw   B'00010010' ; phase 6
        retlw   B'00001001' ; phase 4
        retlw   B'00011000' ; phase 5
        retlw   B'00100100' ; phase 2
        retlw   B'00000110' ; phase 1
        retlw   B'00100001' ; phase 3
        retlw   B'00000000' ; invalid
RevTable
        retlw   B'00000000' ; invalid
        retlw   B'00100001' ; phase /6
        retlw   B'00000110' ; phase /4
        retlw   B'00100100' ; phase /5
        retlw   B'00011000' ; phase /2
        retlw   B'00001001' ; phase /1
        retlw   B'00010010' ; phase /3
        retlw   B'00000000' ; invalid
        END                 ; directive 'end of program'
```