ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

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

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Palma Sanchez Sergio Adrian

Toro Roman Geovanny Rodrigo

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios.

A la familia.

A todas las personas que apoyaron en el desarrollo de este trabajo.

A todos quienes fomentan el desarrollo tecnológico en Ecuador.

DEDICATORIA

A Dios por ser nuestro creador, amparo y fortaleza, cuando más lo necesitamos, y por hacer palpable su amor a través de cada uno de los que nos rodeó.

A nuestros padres, amigos, parejas y profesores, que sin esperar nada a cambio, han sido pilares en nuestro camino y así, forman parte de este logro que nos abre puertas inimaginables en nuestro desarrollo profesional.

TRIBUNAL DE SUSTENTACIÓN

Ing. Carlos Valdivieso.

PROFESOR DE SEMINARIO DE GRADUACIÓN

Ing. Hugo Villavicencio V.

PROFESOR DELEGADO POR LA UNIDAD ACADÈMICA

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesina, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

Palma Sanchez Sergio Adrian

_____

Toro Roman Geovanny Rodrigo

RESUMEN

El principal objetivo de este trabajo es el implementar técnicas aprendidas en nuestra vida académica con respecto al uso de los microcontroladores, sus características, programación y optimización para poder realizar con la ayuda de algunos elementos, varios proyectos que involucren el Control de motores sin escobillas (BLDC) con sensores como tema principal. Dándoles un enfoque más específico y práctico para de esta manera poder enlazar y comprender toda la teoría del funcionamiento de estos motores; haciendo uso de varias herramientas como el software LPCXPRESSO 4 para poder programar el cerebro del kit de desarrollo LPC1114 que consiste de un microcontrolador ARM Cortex3 con 32 bits de LPCXpresso.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El objetivo del proyecto es desarrollar e implementar un grupo de ejercicios claves que permitan comprender toda la teoría y el funcionamiento referente al control de motores sin escobillas. Tratando de darle un uso adecuado y diversificado a los ejercicios, aprovechando las diferentes herramientas que dispone el LPCXPRESSO y demás elementos; para de este modo facilitar la comprensión y entendimiento de esta interfaz de comunicación y control aplicado a la lectura y la enseñanza.

En el primer capítulo, se menciona una descripción general del proyecto, sus partes, la teoría detrás del funcionamiento del control y los diferentes elementos que deben configurarse en el microcontrolador principal que contiene el LPCXPRESSO.

En el segundo capítulo, se da un detalle sobre las herramientas de hardware: LPCXPRESSO Motor Control Kit y de las herramientas de software: LCPXPRESSO 4 con su compilador que permite usar archivos en C y la herramienta de simulación PROTEUS que nos servirá para nuestra primera interacción y comprensión de los motores sin escobillas con sensores de efecto Hall.

En el tercer capítulo se detalla, desarrollar e implementa de manera independiente cada ejercicio propuesto en el proyecto; descripción, diagrama

de bloques, diagrama de flujo, descripción del algoritmo y lista de materiales de los elementos que van a contener cada uno de los ejercicios planteados en el proyecto.

El cuarto y último capítulo, trata de pruebas y simulaciones necesarias para desarrollar nuestro proyecto con una breve descripción fundamentadas con imágenes reales y un diagrama de interconexión de cada una de las pruebas realizadas para alcanzar nuestro objetivo final.

CAPÍTULO 1

DESCRIPCIÓN GENERAL DEL PROYECTO

En los circuitos más modernos, es inevitable que sea necesario incorporar más de un solo chip en el diseño. Por ello, también es esencial que los diversos componentes sean capaces de comunicarse entre sí.

Hay dos formas principales de comunicación, serie y paralelo. En una implementación paralela, hay tantas conexiones como bits que necesitan ser enviados y recibidos. Los buses de comunicación paralelo son muy rápidos, sin embargo, llegan a ser extremadamente grandes y consumen muchos de los invaluables pines que necesita un microcontrolador.

Por la razón de consumo de espacio y de pines, la comunicación serial es preferible a paralelo al conectar dos o más chips, usando solo de dos a cuatro pines. Un bus de comunicación serial funciona mediante el envío de ceros y unos de forma secuencial sobre uno o dos cables. En la mayoría de

los microcontroladores modernos se debe tener un módulo de hardware para realizar un control serial en este caso de un motor BLDC.

En el siguiente trabajo desarrollaremos e implementaremos un banco de ejercicios muy intuitivos que faciliten el entendimiento y comprensión de este tipo de control sobre un motor aplicado a la lectura y la enseñanza del mismo.

1.1. Antecedentes

Los motores eléctricos sin escobillas se han venido utilizando desde hace años en la industria en general, aplicándose en grandes servos, aire acondicionado, ventiladores etc., y su ventaja es que al estar libres de mantenimiento pueden durar muchos años. También se han venido utilizando en los aviones y barcos a control remoto. Sin embargo hasta ahora no se ha dispuesto de una tecnología lo suficientemente pequeña y económica como para aplicarla a los coches de RC. Esto se debe a que los controles del motor son más exigentes en los coches, y a que en los barcos y aviones los frenos no tienen tanta importancia como en los coches.

Tipos de motores sin escobillas:

Los dos grandes grupos de motores son los que utilizan sensores y los que no. El variador de velocidad electrónico ha de tener información de algunos parámetros del motor, como la posición de los imanes para que así pueda enviar la electricidad a la bobina adecuada, en qué sentido está girando el

motor.

Los motores que llevan sensores se denominan de efecto tipo Hall y sus defensores dicen que son necesarios para que el motor tenga un buen par y una buena sincronización. El variador de velocidad controlará de esta forma la excitación del bobinado electromagnético.

Un buen control de la temporización es crítico para el rendimiento y eficiencia del motor. En general el motor con sensores reacciona más rápido, ya que los sensores incrementan el par motor en la salida y la sincronización es mejor en situaciones de alta carga. Los sensores están unidos a la parte trasera del motor y envían señales desde éste al variador de velocidad. El inconveniente de los sensores es que pueden fallar, pero permiten saber si el motor está girando. Esto es una ventaja, ya que si se bloqueara el motor, se evitarían daños a él, a la batería y al variador de velocidad.

Un variador de velocidad con sensor solo podrá trabajar con motores con sensores y además han de estar adaptados mutuamente. No parece que un tipo de motor tenga ventajas indudables sobre el otro, ya que los fabricantes no se han decidido de manera unánime por uno u otro tipo.

El primer sistema embebido reconocido fue el sistema de guía de Apolo desarrollado por el laboratorio de desarrollo del MIT para las misiones Apolo hacia la luna. La función era manejar el sistema de guía inercial de los módulos de excursión lunar. Este sistema de cómputo fue el primero en

utilizar circuitos integrados y utilizaba una memoria RAM magnética, con un tamaño de palabra de 16 bits. El software fue escrito en el lenguaje ensamblador propio y constituía en el sistema operativo básico, pero capaz de soportar hasta ocho tareas simultáneas. Actualmente los sistemas embebidos los encontramos a cada comento de nuestra vida en el celular, televisores, microondas, equipos de audios, aviones, etc. Los sistemas embebidos a pesar de no ser muy conocidos están en muchas partes de nuestro diario vivir, en realidad, es difícil encontrar algún dispositivo en cuyo interior no esté basado en algún sistema embebido, desde aviones hasta teléfonos celulares e incluso en algunos electrodomésticos comunes como refrigeradoras y hornos microondas.

Esto ha llevado a incursionar nuevas ideas al desarrollo de sistemas embebidos utilizando para este proyecto la tarjeta LPC1769 la cual es un sistema embebido que ha sido desarrollada en conjunto por las compañías NXP, LPCXPRESSO y Code Red.

1.2. Motivación

En el mundo cada vez innova nuevas tecnologías para la facilitación del manejo control, automatización, velocidad de comunicación, entre máquinas, aparatos electrónicos y los motores sin escobillas, son motores modernos de los cuales podemos obtener mucho provecho, facilidad, optimización y control pero para un buen control necesitamos de un hardware y software

completo en funciones y de fácil manejo para el programador y usuario como lo es la tecnología de NXP con su productos usando el microcontrolador ARM Cortex3 con 32 bits de LPCXPRESSO.

La motivación para la implementación de este sistema se debe a la poca utilización de energía que caracteriza al motor BLDC, al mantener un óptimo control sobre la velocidad de dicho motor el consumo de energía se ve reducido, a diferencia de los motores con escobillas.

El control motor BLDC nos permite realizar ajustes a ciertos parámetros tales como sentido de rotación del eje del motor y la velocidad del mismo, lo que nos ayudará a encontrar diferentes usos y aplicaciones en las que podríamos incursionar tales como mantener control en un determinado proceso.

Estos motores tienen un mayor tiempo de vida útil frente a los motores con escobillas debido a que en los motores BLDC en ausencia de escobillas no presentan el desgaste que es característico y muy común en los motores con escobillas, lo que implica también la disminución de ruido y calor generado por el frecuente rozamiento. Los motores BLDC trabajan a grandes velocidades debido a que la conmutación se basa en un circuito electrónico muy eficaz.

1.3. Identificación del problema

Empezaremos a trabajar en proyectos con motores BLDC con sensores utilizándolos en diferentes aplicaciones y comunicaciones como UART, I2C, SPI, utilizando una herramienta muy útil en muchos de estos proyectos la tarjeta AVR Butterfly (con microcontrolador ATMega169) unos de los problemas que tendremos que solucionar es el correcto funcionamiento del motor en la etapa final de estos trabajos con esta nueva tecnología de NXP que es el LPCXPRESSO mediante la tarjeta LPC1769 de 32 bits y el kit de control del motor, donde nosotros debemos manipular, corregir o modificar configuraciones y códigos con sus respectivas librerías para el correcto control del motor de una manera muy sencilla para que los otros proyectos puedan comprender fácilmente y hacer uso de nuestro trabajo de esta manera resuelvan sus problemas en sus respectivos proyectos.

En la actualidad los motores BLDC con sensores ha sido tomado en cuenta en diferentes sectores industriales por ejemplo el mercado de la climatización y refrigeración el cual ya ha puesto en el mercado equipos de acondicionamiento de aire de alta eficiencia (ahorran hasta un 60% de energía eléctrica en comparación con los equipos normales) denominados Inverter los cuales utilizan motores BLDC controlados electrónicamente, de esta manera el potencial comprador se ve decantado por las bondades que ofrece dicho producto no solo en el aspecto económico sino también en el aspecto ecológico preservando el medio ambiente del planeta.

1.4. Objetivos Principales

- Enseñar al estudiante el uso de técnicas utilizadas en el control de motores BLDC. Entender la utilidad e importancia que tienen los sistemas embebidos así como las grandes ventajas que nos ofrecen al momento de programar y controlar estos motores, además de obtener gran cantidad de conocimientos acerca del funcionamiento de este tipo de sistemas.
- Implementar un sistema de control de un motor BLDC con sensores utilizando los microcontroladores ARM Cortex3 con 32 bits de LPCXPRESSO LPC1114 y LPC1769 que conjuntamente trabajarán con el LPCXPRESSO MOTOR CONTROL BOARD el cual es una plataforma universal para control de motores en microcontroladores desarrollados por NXP.
- Analizar y determinar el comportamiento de motor brushless mediante sensores de temperatura y vibración a medida que aumenta o disminuye la velocidad inclusive cuando este cambie de giro.
- Comprender y aprender el lenguaje de programación de este software LPCXPRESSO de una manera muy eficiente y óptima para encaminar al desarrollo de nuestro proyecto o futuras aplicaciones útiles para la sociedad.

1.5. Limitaciones

AL desarrollar este proyecto tendremos algunas limitaciones en aspectos de interés tales como trabajar por primera vez con los controladores de la familia ARM CORTEX de LPCXPRESSO lo que implica el aprendizaje de los bloques, componentes, dispositivos y arquitectura que comprenden cada uno de las tarjetas, tanto de la LPC1114 como de la LPC1769.

La tarjeta LPCXPRESSO como en el caso de la tarjeta LPC1114 es limitada en el número de pines disponibles para la interacción con el MOTOR CONTROL BOARD, es por esto que nos vemos con cierta limitación de acuerdo al tipo de tarjeta que utilicemos para el desarrollo de este proyecto.

En los motores con escobillas la conmutación se hace mecánicamente a través del contacto entre el inducido y las escobillas. Este sistema es muy poco eficiente y el rozamiento y la resistencia eléctrica provocan que haya una gran pérdida de energía que se transforma en calor. Además esto supone una limitación al número máximo de r.p.m. de los motores, ya que a elevadas r.p.m. las escobillas rebotarían. Para evitarlo serían necesarios muelles más rígidos y esto crearía a su vez más fricción y frenaría el giro. En los motores sin escobillas la conmutación se controla de manera electrónica mediante el variador de velocidad, por lo que no se produce rozamiento mecánico ni pérdidas de energía y se consigue que los motores sin escobillas tengan una eficiencia muy superior, se habla de un 90%, frente a

un 60% de los motores con escobillas. Debido a esto los motores sin escobillas pueden alcanzar muchas más r.p.m. y con mucho más par, hasta cuatro o cinco veces más que los motores con escobillas, y a la vez con un ahorro de energía de hasta el 30 %. Además el calentamiento del motor es mínimo. En los motores sin escobillas la masa que gira es menor, casi la mitad, por lo que aceleran más rápidamente. Esto se debe a que el rotor lleva los imanes, que son menos pesados en estos motores que el bobinado en los clásicos. Por ello el funcionamiento es además más suave al reducirse las vibraciones. Al no haber chisporroteo eléctrico debido al roce de las escobillas con el conmutador, se eliminan las interferencias por el "ruido" eléctrico que podrían afectar al equipo de radio. Tampoco son necesarios ni los condensadores soldados al motor ni el diodo Schottky. En los motores con escobillas hay que realizar mantenimientos periódicos; nuevas escobillas, muelles, desgaste del conmutador y mucho tiempo dedicado a las labores de mantenimiento Por el contrario, en los motores sin escobillas el mantenimiento es mínimo, ya que al no tener ni estas ni conmutador, el mantenimiento es casi inexistente, solo sería necesario la limpieza y lubricado de los rodamientos. Adicionalmente, el concepto de los "sin escobillas" permite fabricar motores totalmente cerrados, protegiéndolos del polvo. Los motores sin escobillas son más sencillos y por ello más fiables: no llevan ni las escobillas, ni guías de escobillas, ni conmutador, ni muelles. Es decir menos mangas perdidas por culpa de fallos mecánicos del motor.

Además el peso de esos motores puede ser de hasta 50 gramos menos que los con escobillas. Por otra parte y gracias al control ejercido por el variador electrónico, que es de hecho un microprocesador digital, es posible regular el par y las r.p.m. e igualar las prestaciones de los motores. Se podrían hacer carreras con motores de prestaciones iguales y se podría controlar en la inspección técnica. Además no habría que disponer de varios motores, solo sería necesario programar el número de r.p.m. Pero ¿no tienen inconvenientes? La verdad es que pocos, pero se pueden apuntar los siguientes: El primero es su juventud en los coches RC. Debido a ello han de abrirse paso en un mercado dominado por marcas con intereses económicos no solo en la fabricación de motores, sino de muelles, escobillas etc., lo que hace suponer que habrá una cierta presión para que no cambien las cosas. También su juventud puede ser un obstáculo al principio para su participación en competiciones oficiales, ya que es de suponer que hasta que no haya una cierta difusión, no se organizarán competiciones oficiales. Otro inconveniente para los que disponen de equipo de RC puede ser el económico, ya que además de adquirir el nuevo motor, hemos de comprar un variador para motores sin escobillas, ya que los variadores clásicos no sirven. Por último y debido a que los motores sin escobillas dan más potencia y aceleración que los clásicos, por lo que las transmisiones de algunos coches que no sean de alto nivel podrían sufrir.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

El proyecto se lo puede dividir básicamente en dos partes esenciales: Software y Hardware, ambos indispensables para la elaboración del proyecto.

El software que se utilizará para la programación de los microcontroladores requeridos en el proyecto es el LCPXPRESSO 4.0, el cual es un Entorno de Desarrollo Integrado para escribir y depurar aplicaciones de control de motores en el entorno de Windows y posee dos compiladores para el lenguaje C usados para la creación de los códigos. También se utilizará el software de simulación PROTEUS 7.7 Service Pack 2, el cual permite implementar en forma simulada los códigos hechos en lenguaje C con los integrados y sus conexiones.

El hardware que se utilizará para desarrollar el proyecto principalmente es el LPCXpresso Motor Control Kit, el cual es un Kit de desarrollo, entrenamiento

y aprendizaje de microcontroladores Avanzado, que contiene: El kit contiene un tablero con LPCXpresso LPC1114 de destino, que tiene una cortex-M0 núcleo, como controlador del motor. Varios otros MCU NXP también se puede utilizar con la junta para demostrar el control del motor. Con esta plataforma es posible controlar BLDC, BLAC, paso a paso y de doble cepillado motores de corriente continua.

2.1. Herramientas de Software

Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros: Editores de texto, Compiladores, Intérpretes, Enlazadores, Depuradores, Simuladores.

2.1.1. LPCXPRESSO 4.0

LPCXPRESSO es una nueva plataforma de bajo costo de desarrollo disponible de NXP. El software consiste en un aumento, IDE basado en Eclipse, un compilador de C de GNU, enlazador, librerías, y un mayor depurador GDB. El hardware consiste en la placa de desarrollo LPCXPRESSO que tiene una interfaz de depuración LPC-Link y un NXP LPC basado en ARM microcontrolador objetivo.

LPCXPRESSO es una solución de extremo a extremo, permitiendo a los ingenieros incorporados a desarrollar sus aplicaciones desde la evaluación inicial hasta la producción final.

Figura 2.1.: LPCXPRESSO 4.0 [10]

El IDE LPCXPRESSO, impulsado por Code Red Technologies, se basa en la plataforma de desarrollo Eclipse y popular incluye varias mejoras específicas de LPC. Se trata de un estándar de la industria conjunto de herramientas GNU con una optimizada biblioteca de C que permite a los ingenieros de todas las herramientas necesarias para desarrollar soluciones de software de calidad superior de forma rápida y rentable. El entorno de programación C incluye características de nivel profesional. No es la sintaxis de colores, formato de origen, la función plegable, dentro y ayuda en línea, y una amplia automatización de gestión de proyectos. La placa de destino LPCXPRESSO, desarrollado conjuntamente por NXP, Code Red Technologies y Artistas Embedded , incluye un integrado depurador JTAG (LPC-Link), así que no hay necesidad de una depuración JTAG por separado sonda. La porción de destino de la junta se puede conectar a las juntas de expansión para proporcionar una mayor variedad de interfaces

y dispositivos I/O. El bordo LPC-Link proporción a depurador un USB de alta velocidad para JTAG / SWD interfaz para el IDE y puede ser conectado a otro objetivos de depuración, tales como un prototipo del cliente. Los usuarios también pueden usar el IDE LPCXPRESSO con la Red de la sonda JTAG adaptador de Code Red Technologies.

LPCXPRESSO IDE

IDE LPCXPRESSO es un entorno de desarrollo de software altamente integrado de NXP Microcontroladores LPC, que incluye todas las herramientas necesarias para el desarrollo de alta calidad soluciones de software de una manera efectiva en tiempo y costo. LPCXPRESSO está basado en Eclipse con muchas mejoras específicas LPC. También cuenta con la versión más reciente de la industria estándar de cadena de herramientas GNU con una biblioteca propia optimizado C proporcionando profesionales herramientas de calidad a bajo costo. El IDE LPCXPRESSO puede construir un ejecutable de cualquier tamaño con la optimización del código completo y es compatible con un límite de descarga de 128kB después de la matriculación. LPCXPRESSO apoya el ciclo completo de diseño embebido producto por ir más allá de chips placas de evaluación y desarrollo de apoyo en las juntas externo objetivo.

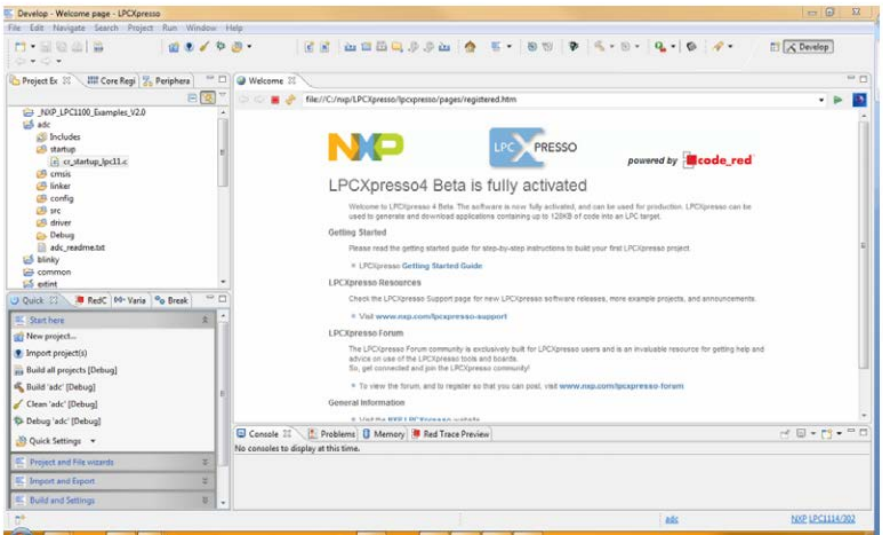

Figura 2.2.: LPCXPRESSO IDE [10]

2.1.2. PROTEUS

PROTEUS es una herramienta software que permite la simulación de circuitos electrónicos con microcontroladores como el PIC16F877 para dominar el control de motor sin escobillas con sensores. Sus reconocidas prestaciones lo han convertido en el más popular y completo simulador software para este tipo de microsPICS y motores BLDC.Esta herramienta es una solución de extremo a extremo, permitiéndonos desarrollar, analizar y comprender sus aplicaciones desde la evaluación inicial hasta la producción final sobre el control de un motor las cuales pueden ser incrementar y decrementar la velocidad del mismo, como también cambiar el sentido de giro del motor BLDC, permitiéndonos simular este tipo de circuitos

electrónicos complejos, integrando inclusive desarrollos realizados con microcontroladores de varios tipos, en una herramienta de alto desempeño con unas capacidades graficas impresionantes.

En este software tenemos todas las piezas necesarias para el control un motor sin sensor. Podemos medir el voltaje aplicado luego comparar entre sí para determinar la posición del rotor.

Se puede variar el voltaje eficaz aplicado con PWM y control de la velocidad del motor midiendo el tiempo de las fases de conmutación. Algunos eventos a medir debe ser precisamente el tiempo. El ADC debe ser cambiado de una fuente a otra y permitir suficiente tiempo de adquisición. Algunos eventos debe suceder rápidamente, con una latencia mínima. Estos incluyen PWM y la conmutación.

Presenta una filosofía de trabajo semejante al SPICE, arrastrando componentes de una barra e incrustándolos en la aplicación, es muy sencillo de manejar y presenta una interfaz gráfica amigable para un mejor manejo de las herramientas proporcionadas por el Proteus.

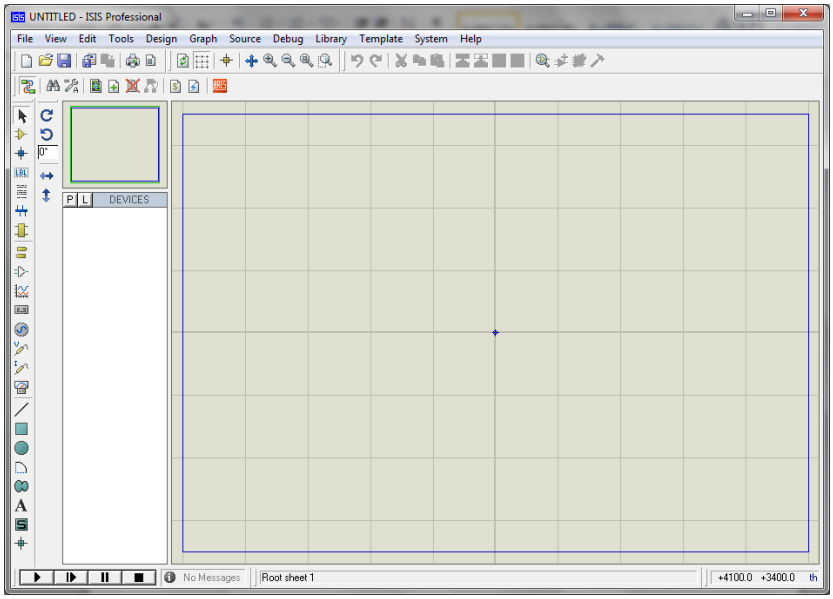

Figura 2.3.: Interfaz Gráfica Proteus.

2.2. Herramientas de Hardware

Las herramientas de hardware corresponden a todas las partes tangibles de un sistema informático: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos; sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado.

2.2.1 BLDC control del motor con LPC1769

La Corriente continua sin escobillas (BLDC) los motores están ganando popularidad rápidamente. Ofrecen una vida más larga y menos mantenimiento que los convencionales cepillado motores de corriente continua. Algunas otras ventajas sobre cepillado motores de corriente continua y motores de inducción son: una mayor velocidad en comparación con las características de par, operación silenciosa y más altos rangos de

velocidad. Además, el radio de torque suministrado al tamaño del motores mayor, haciéndolos útiles en aplicaciones donde el espacio y peso son factores críticos.

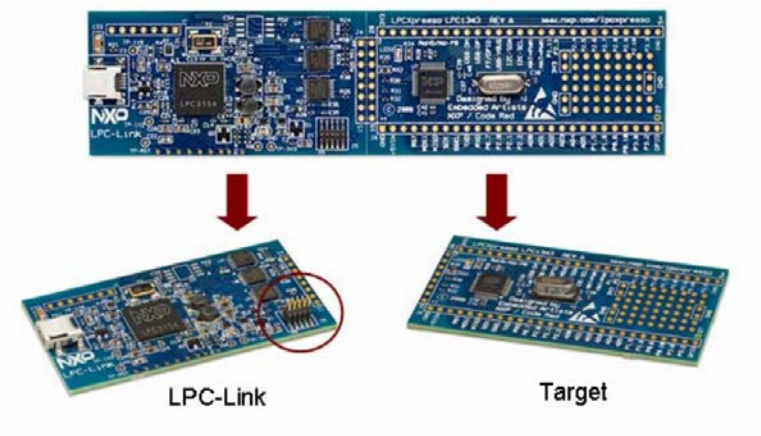

Figura 2.3.: Tarjeta LPCXpresso [9]

Esta aplicación muestra cómo implementar de seis pasos de conmutación o de corriente continua sin escobillas de control del motor en la familia LPC1769. EL PC1769 está equipado con un motor dedicado de control PWM que reduce la utilización de la CPU durante el control del motor mientras que también reduce el tiempo de desarrollo. Esta nota de aplicación también pretende ser una referencia y punto de partida para que los desarrolladores del motor del sistema de control que utilizan los microcontroladores de la familia LPC1769.

Un control completo del motor solución de sistema, no sólo muestra el uso de los microcontroladores NXP sino también NXP circuitos MOSFET conductor y los OSFETs.

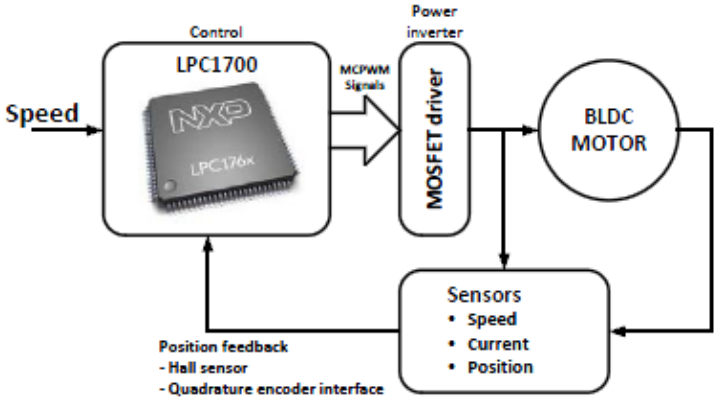

Figura 2.4.: LPC1769 Diagrama de bloques del control del motor [9]

Se muestra un diagrama de bloques de control del motor, que es adecuado para una amplia variedad de aplicaciones, incluyendo aire acondicionado, bombas eléctricas, ventiladores, impresoras, robots, motos eléctricas, batidoras, procesadores de alimentos, licuadoras, aspiradoras, cepillos de dientes, maquinillas de afeitar, molinillos de café, etc.

2.2.2 LPCXpresso de control de motores

Es una plataforma universal para el control de baja tensión del motor sobre la base de Microcontroladores de NXP. Está diseñada específicamente para controlar BLDC, BLAC, paso a paso y de doble cepillado motores de

corriente continua. La plataforma apoya directamente el control a través de la LPCXPRESSO LPC1114.

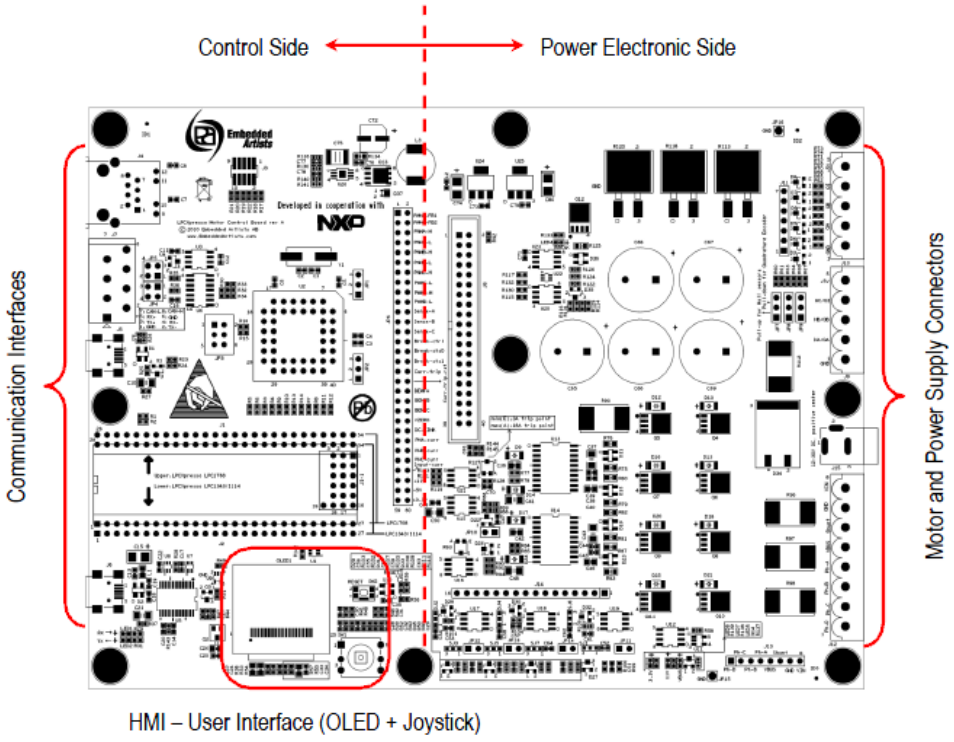

Figura 2.5.: Vista de los principales componentes de la tarjeta del motor control [7]

El motor control LPCXPRESSO tiene una estructura tal como se indica en la imagen siguiente. El lado derecho contiene la electrónica de potencia para la conducción de las fases del motor. El lado izquierdo es el lado de control con zócalos para tarjetas de LPCXPRESSO diferentes, así como una toma de control del procesador PLCC434. El lado izquierdo contiene también un

OLED y una palanca de mando que puede servir como una interfaz de usuario para el sistema.

El LPC1114 es un procesador basado en ARM Cortex-M0, de bajo costo de 32-bit, diseñado para aplicaciones de microcontrolador 8/16-bit, que ofrece un rendimiento buen rendimiento, bajo consumo de potencia, simple conjunto de instrucciones y fácil direccionamiento de memoria, junto con el tamaño de código reducido en comparación con arquitecturas existentes 8/16-bit. El LPC1114 opera en frecuencias de la CPU de hasta 50MHz. El complemento periférico de la LPC1114 incluye un máximo de 32 kB de memoria flash de memoria, hasta 8 KB de memoria de datos, un modo de interfaz I2C-bus,una conexión para RS-485/EIA-485 UART, y posee dos interfaces SPI con las características del SSP, cuatro contadores / temporizadores de propósito general, un ADC de 10 bits, y 42 entradas o salidas I / O pins.

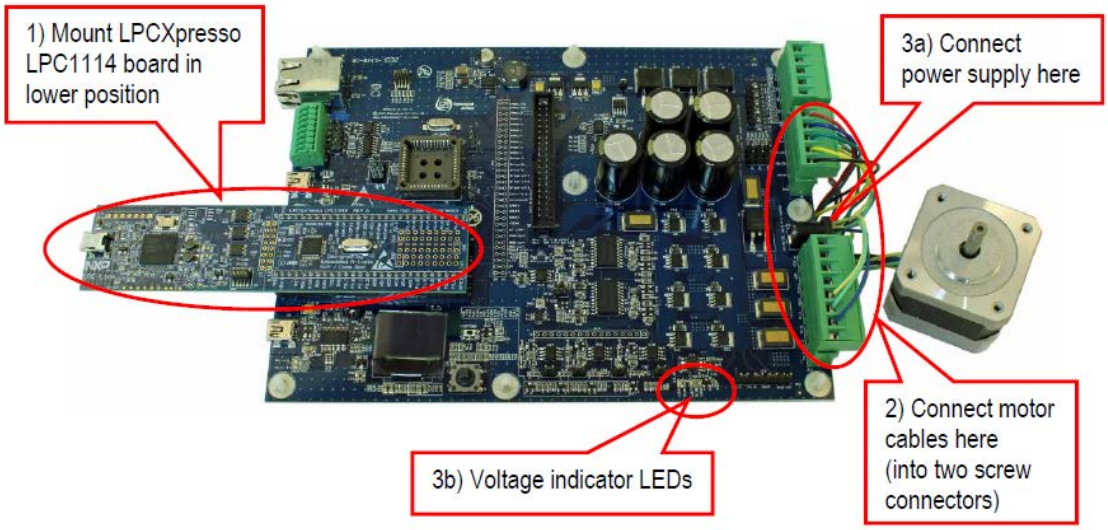

Figura 2.6.: LPCXpresso Motor Control Kit [7]

2.3. Motores de Corriente Continua sin Escobillas (BLDC).

2.3.1. Reseña Histórica.

Un motor en si es un transductor, el cual transforma un determinado tipo de energía (eléctrica, hidráulica, eólica, química, etc.) en energía mecánica, haciéndolo en unos casos de manera más eficiente que en otros casos.

Los motores son parte esencial de nuestra vida, han transformado nuestro mundo, inclusive nos han llevado a la luna, los podemos encontrar desde autos hasta podadoras, de embarcaciones a ferrocarriles.

Los motores más antiguos conocidos son los motores hidráulicos, los cuales aprovechaban una diferencia de potencial (altura) para mover una rueda hidráulica, actualmente una turbina. Luego de esto aparecen los molinos de viento los cuales transforman la energía del viento en energía mecánica.

La revolución industrial llevo a que se dejaran de lado estas formas primitivas de energía para dar paso a la máquina de vapor (motor de combustión externa) en el siglo XVIII donde Thomas Newcomen y John Calley diseñaron el motor a vapor, pero en 1782 el ingeniero escocés James Watt implementa una máquina de vapor de mayor eficiencia a las anteriores.

Conforme a la creciente necesidad del ser humano de realizar trabajos con mayor eficiencia se logró el desarrollo del motor eléctrico el cual transforma energía eléctrica en energía mecánica. Este avance se logró gracias al científico británico Michael Faraday mediante el descubrimiento de la inducción magnética. Los motores tanto de corriente continua como de corriente alterna funcionan bajo el mismo principio el cual dice que si un conductor por el cual circula corriente eléctrica se encuentra dentro de la acción de un campo magnético, este tiende a moverse en dirección perpendicular a las líneas de campo del campo magnético.

En tiempo presente, donde nos vemos en la necesidad de la conservación de recursos y reducir los niveles de contaminación, el motor eléctrico como nueva alternativa de fuente de potencia es bien vista por los consumidores, aunque estos presentan una desventaja la cual es un bajo nivel de autonomía. Los continuos avances para mejorar la eficiencia de estos motores, han sido utilizando nuevos motores con capacidad de control, entre los cuales los mas aptos para ser acogidos son los motores sin escobillas denominados también BRUSHLESS DC, que ofrecen una ventaja en el

control de conmutación electrónica contra la conmutación física dada por los motores con escobillas, reduciendo además el peso de los mismos y aumentando su eficiencia siendo optima su utilización en sistemas que requieran un adecuado consumo de la potencia eléctrica, lo cual se ve reflejado en un ahorro monetario para el consumidor.

2.3.2 Características (BLDC).

Estos motores llegan una con el desarrollo de los transistores y de otros dispositivos semiconductores de conmutación. Su función principal es proporcionar conmutación electrónica entre arrollamientos de fase, lo que conlleva a un mejor control de los motores, también ofrecen una mayor eficacia en la inducción y en el sincronismo del motor. Cabe recalcar una ventaja importante sobre los motores con escobillas, los cuales producen rozamiento, disminuyen el rendimiento, desprenden calor, son ruidosos y requieren una sustitución periódica y, por tanto, un mayor mantenimiento.

2.3.3. Ventajas.

Los motores BLDC tienen muchas ventajas sobre los motores convencionales con escobillas frente a los motores de inducción, unas de ellas las listamos a continuación:

- Mejor relación velocidad-par motor
- Mayor respuesta dinámica

- Mayor eficiencia
- Mayor vida útil
- Menor ruido
- Mayor rango de velocidad

Además la relación par-tamaño del motor es mucho mayor lo que conlleva a que se pueden emplear en aplicaciones donde se trabaje en un espacio reducido.

2.3.4. Desventajas.

Las desventajas de los motores BLCD son mínimas en contraste con las ventajas, estas desventajas son:

- Mayor costo
- Mayor complejidad en el control

2.3.5. Motores BLCD vs. Motores de Corriente Continua con Escobillas.

Los A continuación presentamos la tabla 1.2 en la que se ilustra un contraste entre motores BLCD y motores de corriente continua con escobillas.

	MOTOR BLCD	MOTOR DC CON ESCOBILLAS
Conmutación	Conmutación electrónica basada en sensores de posición de efecto Hall	Conmutación por escobillas
Mantenimiento	Mínimo	Periódico
Durabilidad	Mayor	Menor
Curva Velocidad/Par	Plana. Operación a todas las velocidades con la carga definida	Moderada. A altas velocidades la fricción de las escobillas se incrementa, reduciendo el par
Eficiencia	Alta. Sin caída de tensión por las escobillas	Moderada
Potencia de Salida/Tamaño	Alta. Menor tamaño debido a menores características térmicas porque los bobinados están en el estator, que al estar en la carcasa tiene una mejor disipación de calor	Baja. El calor producido en la armadura es disipado en el interior aumentando la temperatura y limitando las características
Inercia del Rotor	Baja. Debido a los imanes permanentes en el Rotor	Alta. Limita las características dinámicas
Rango de Velocidad	Alto. Sin limitaciones mecánicas impuestas por escobillas/conmutador	Bajo. El límite lo imponen principalmente las Escobillas
Ruido eléctrico generado	Bajo	Arcos en las escobillas
Coste de construcción	Alto. Debido a los imanes permanentes	Bajo
Control	Complejo y caro	Simple y barato

Tabla 2.1: Motor BLCD vs. Motor DC con Escobillas [2] [3]

2.3.6. Estructura del Motor BLCD.

Los Su estructura es similar a la de los motores de corriente continua con escobillas, además pueden construirse en configuraciones físicas diferentes:

- Configuración convencional llamada también Inrunner: aquí los imanes permanentes son parte del rotor.
- Configuración Outrunner: aquí la relación radial entre las bobinas e imanes se invierte y las bobinas del estator forman el núcleo del motor.

Este tipo de motores son capaces de sensar la posición de los polos magnéticos y asi poder realizar el control mediante conmutaciones electrónicas. Para sensar la posición de los polos magnéticos generalmente se la realiza con sensores de efecto Hall, aunque existen modelos que lo hacen con sensores ópticos similares a los encoders.

Figura 2.7.: Motor BLDC [4]

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN DEL PROYECTO

En el presente capitulo detallaremos todas las etapas y experiencias vividas para la consecución de nuestro objetivo de lograr controlar un motor BLCD, desde la primera interacción con el software a utilizar así como los ejemplos antes realizados para la llegar a familiarizarnos con este software y hardware. Así también todas las pruebas que se desarrollaron previos al desarrollo del proyecto.

El conjunto de ejercicios que se especifican más adelante estará compuesto de varios elementos, muchos de los cuales se les da el mismo uso en todos los ejercicios como 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.

3.1. Descripción de los Ejercicios

A continuación se describen detalladamente uno a uno los ejercicios del proyecto que involucran comunicaciones seriales dedicado al trabajo con microcontroladores ARM Cortex3 con 32 bits de LPCXPRESSO con aplicaciones específicas.

3.1.1. Controlar la rotación de Leds.

- Descripción:

La primera interacción con el LPCXPRESSO y con la tarjeta LPC1769 fue con la elaboración de un pequeño código para familiarizarnos con la manipulación y configuración de entradas y salidas de la tarjeta LPC1769.

Dicha codificación estaba destinada a controlar una secuencia de luces de 8 led's los cuales están conectados a 8 salidas de la tarjeta, esta secuencia es controlada por una botonera la cual será nuestra entrada.

La botonera trabaja con lógica negativa:

- Sin pulsar = '1'
- Pulsando = '0'

Las salidas están asignadas a: GPIO2.0 hasta GPIO2.7

La entrada está asignada a: GPIO2.8

La secuencia 1 se activa por defecto y se la detalla en la siguiente imagen:

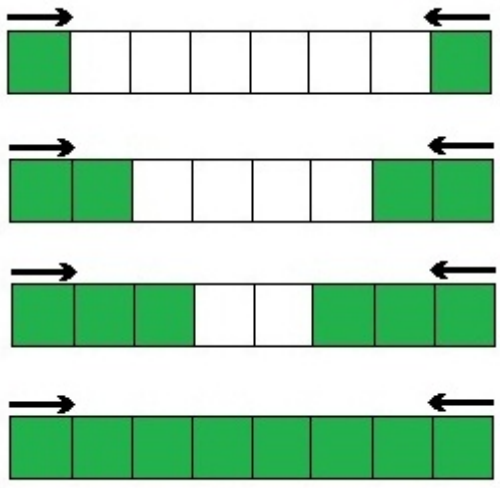

Figura 3.1.: Secuencia 1

La secuencia 2 se activa al presionar la botonera y se la detalla en la siguiente imagen:

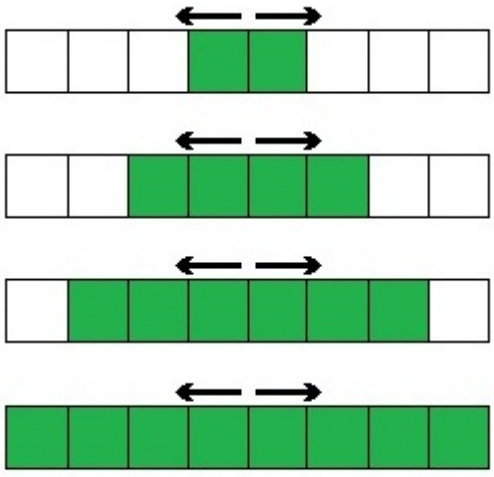

Figura 3.2.: Secuencia 2

- Diagrama de Bloques

Una botonera será conectada a la tarjeta para que esta detecte el momento en que el sentido de secuencia de los diodos leds. Dispositivo principal es la tarjeta LPC1769 quien enviará las diferentes señales de voltaje a través del puerto dos para de esta manera encender cada uno de los diodos led en la secuencia indicada.

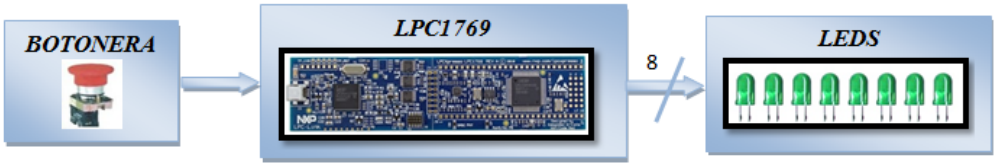

Figura 3.3.: Bloques de Secuencia de Leds.

- Diagrama de Flujo

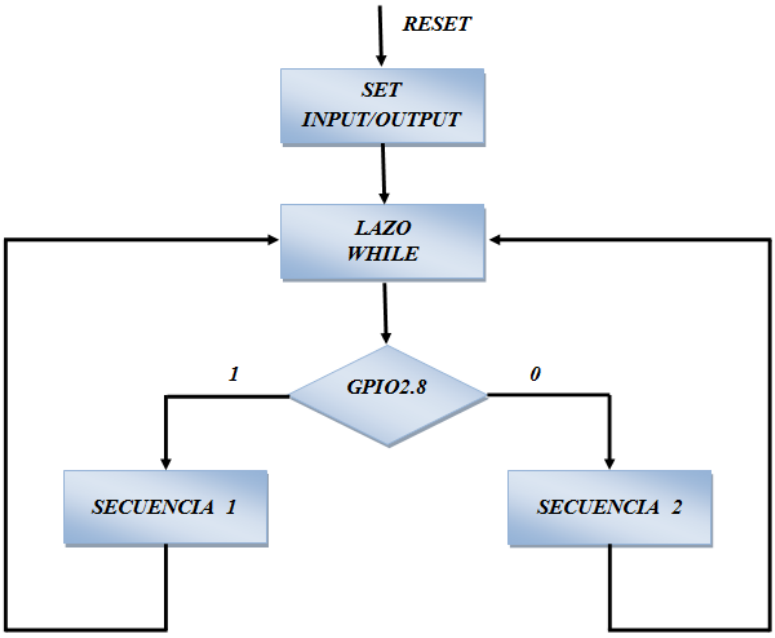

Figura 3.4.: Flujo de Secuencia de Leds.

- Descripción del Algoritmo

1) Inicialización de variables.
2) Definición de puertos a utilizar declarándolos ya sea como entradas o salidas en cada uno de sus respectivos pines.
3) Seteo de los valores en cada pin.
4) Ingreso en un lazo infinito.
5) Pregunta si la botonera que está conectada en el pin 8 ha sido activada.
6) Si la botonera no ha sido activada los bits se encienden de afuera hacia adentro.
7) Si la botonera ha sido activada los bits en la próxima secuencia se encenderán de adentro hacia afuera.
8) Los pasos 5,6 y 7 se repiten indefinidamente ya que se encuentran en el lazo infinito.

- Código del Programa Principal

```
/*************************************************************
 *  SECUENCIA DE ENCENDIDO DE LEDS
 *  AUTORES:        GEOVANNY RODRIGO TORO ROMAN
 *                  SERGIO ADRIAN PALMA SANCHEZ
 *
 *  DESCRIPCION:
 *   ESTE PEQUEÑO PROGRAMA EJECUTA UNA SECUENCIA DE LUCES
 *   8 LEDS, CONECTADOS EN EL PUERTO 2 GPIO2.0 HASTA GPIO2.7
 *   LA SECUENCIA 1 SE ACTIVA POR DEFAULT, EN CAMBIO LA
 *   SECUENCIA 2 SE ACTIVAAL PRESIONAR LA BOTONERA CONECTADA
```

```
 *     A LA ENTRADA GPIO2.8
**************************************************************/
#include <cr_section_macros.h>
#include <NXP/crp.h>
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
#include "lpc17xx.h"
#include "type.h"
int main (void)
{
  uint32_t i, j, k;//DEFINICION DE VARIABLES
  SystemClockUpdate();
  LPC_GPIO2->FIODIR = 0xFFFFFEFF;      // DEFINICION DE
SALIDAS/ENTRADAS
  LPC_GPIO2->FIOCLR = 0xFFFFFFFF;     // LEDS APAGADOS
  LPC_GPIO2->FIOMASK= 0x00000000;
  while(1){
            k=4;
          if(LPC_GPIO2->FIOPIN==0x00000100){
                    for(i = 7; i >3; i--){
                              LPC_GPIO2->FIOSET = 1 << k;
                              LPC_GPIO2->FIOSET = 1 << (i-4);
                              k++;
                              for(j = 1000000; j > 0; j--);
                    }
                    LPC_GPIO2->FIOCLR = 0x000000FF;
```

```
            for(j = 1000000; j > 0; j--);
        }
      if(LPC_GPIO2->FIOPIN==0x00000000){
        for(i = 0; i < 4; i++){
            LPC_GPIO2->FIOSET = 1 << i;
            LPC_GPIO2->FIOSET = 1 << (7-i);
            for(j = 1000000; j > 0; j--);
        }
            LPC_GPIO2->FIOCLR = 0x000000FF;
            for(j = 1000000; j > 0; j--);
      }
  }
}
```

- **Lista de Materiales:**

Para la implementación de este pequeño proyecto necesitamos lo siguiente:

- 1 Protoboard
- 1 LPC1769
- 1 Cable USB (para programación y alimentación de la tarjeta LPC1769)
- 8 Leds
- 9 Resistencias de 330Ω
- 1 Resistencia de 10KΩ
- Cable UTP

3.1.2. Deshabilitación del joystick y seteo de nuevos pines de entrada para el control del motor BLCD.

-Descripción:

El primer obstáculo en la elaboración del presente proyecto fue deshabilitar el joystick y setear nuevas entradas para el control del motor BLCD para que pueda interactuar la tarjeta LPC1114 que viene incluida en el Motor control kit con la tarjeta LPC1769.

Esto se logró cambiando las entradas asignadas al joystick por nuevas disponibles en la LPC1114.

Las nuevas entradas para el control del motor BLCD son:

- PIO2.4 => On/Stop: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD inicia su funcionamiento si está detenido, pero si está girando detiene su funcionamiento.

- PIO3.1 => Invertir giro: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD invierte el sentido de su giro.

- PIO3.2 => Decrementar velocidad: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD decrementa su velocidad en pasos de 50 RPM.

- PIO3.3 => Incrementar velocidad: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD incrementa su velocidad en pasos de 50 RPM.

El joystick fue sustituido por 4 botoneras que cumplen las funciones antes mencionadas para el control del motor BLCD.

- Diagrama de Bloques

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 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, 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, etc.

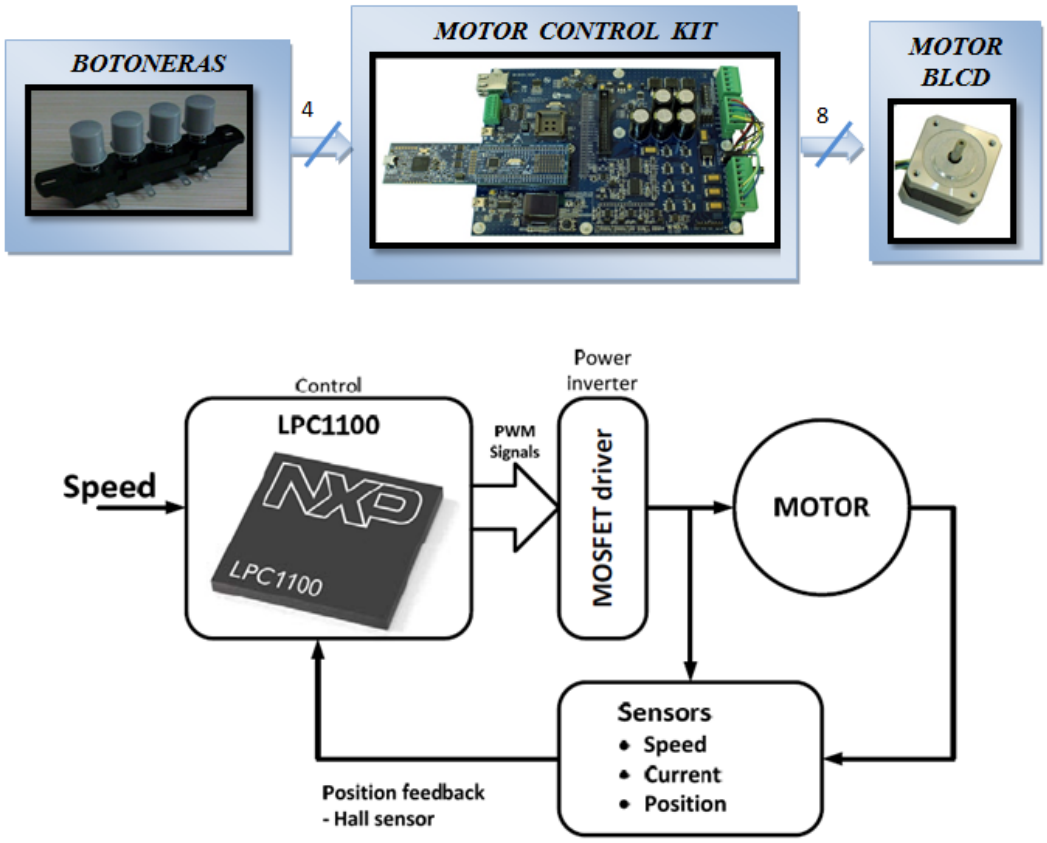

Figura 3.5: Diagrama de bloques del Motor Control Kit

- Diagrama de Flujo

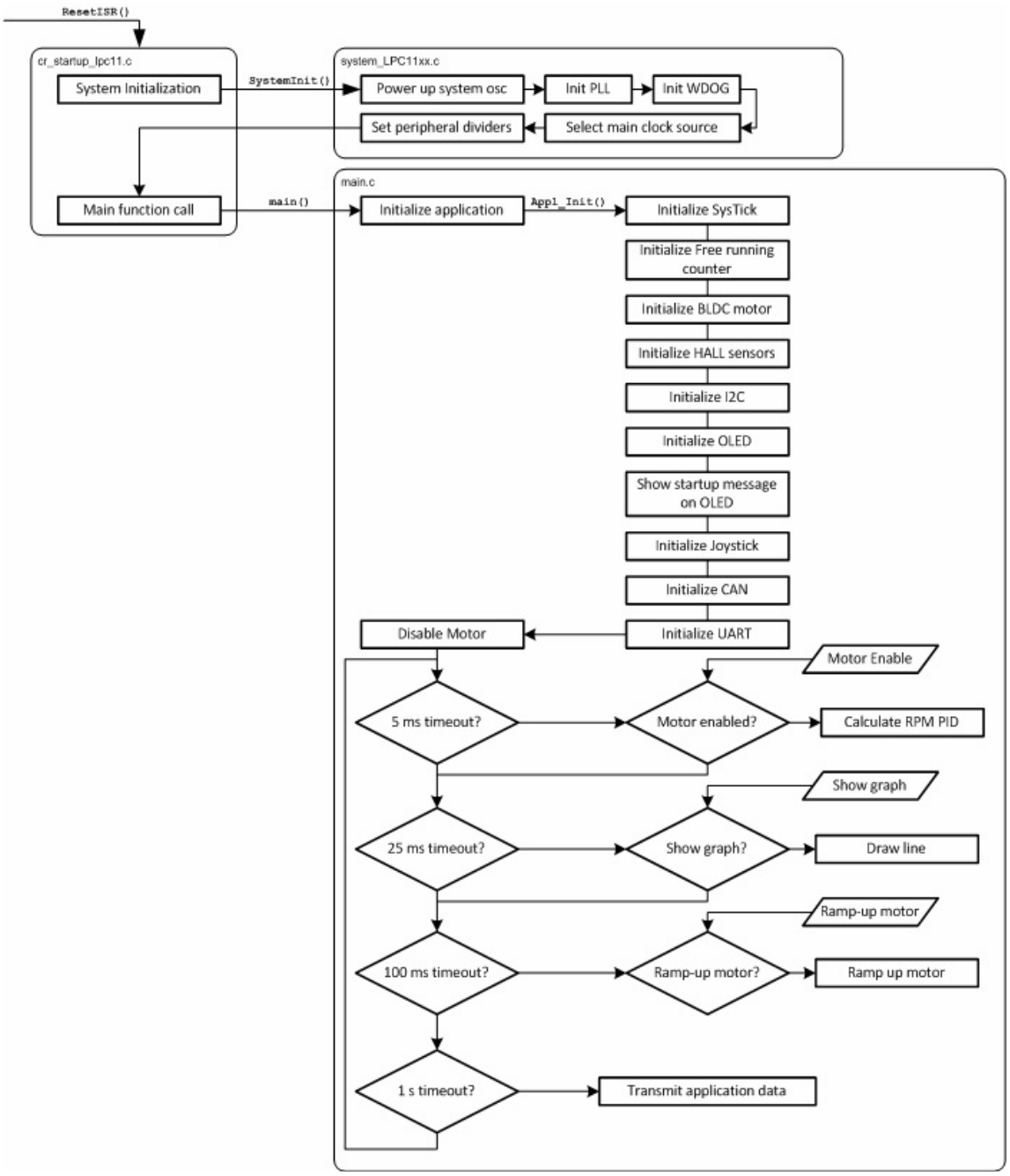

Figura 3.6: Diagrama de Flujo [7].

- Descripción del Algoritmo

1) Inicialización de variables.
2) Definición de puertos a utilizar como entradas y salidas, en este caso hemos seleccionado el puerto dos, los primeros cuatro pines serán definidos como salidas y los siguientes cuatro pines serán definidos como entradas, también es necesario asignarle un valor de ‘1’ lógico a los primeros cuatro pines de salida, ya que estamos trabajando con lógica negativa.
3) Si el usuario presiona el botón asignado al pin P2.4, éste recibe un ‘0’ lógico lo que activa el motor o detiene su movimiento.
4) Si el usuario presiona el botón asignado al pin P3.1, éste recibe un ‘0’ lógico lo que ocasiona la inversión del sentido de giro.
5) Si el usuario presiona el botón asignado al pin P3.3, éste recibe un ‘0’ lógico lo que ocasiona que se incremente la velocidad de giro del motor.
6) Si el usuario presiona el botón asignado al pin P3.2, éste recibe un ‘0’ lógico lo que ocasiona que disminuya la velocidad de giro del motor.
7) Los pasos 3 al 6 se repiten indefinidamente ya que este procedimiento se encuentra en un lazo infinito.

- Programa Principal del controlador

Main.c

```
/*****************************************************************************
 * PROYECTO: CONTROL DE MOTOR BLCD
/*****************************************************************************
 * Includes
 *****************************************************************************/
#include "application.h"
#define GRAPH_LIMIT        64-8
/*****************************************************************************
 * External global variables
 *****************************************************************************/
extern volatile uint32_t SysTick_cntr;
extern volatile MOTOR_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_t connTimer=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
 *****************************************************************************/
void MemCopyBYTE(uint8_t *src, uint8_t *dst, uint32_t count);
void MemCopyDWORD(uint32_t *src, uint32_t *dst, uint32_t count);
void InitSyncPacket (void);
void ProcessRxPacket(uint8_t index);
/*****************************************************************************
** Function name: Call_5ms
*****************************************************************************/
void Call_5ms(void)
{
        if (Motor.Enable && (Motor.RampingUp == FALSE))
        {
        /* Do the PID calculations */
        vPID_RPM(&Motor);
```

```
        }
}
/*****************************************************************************
** Function name:25ms
*****************************************************************************/
void Call_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
}
/*****************************************************************************
** Function name:100ms
*****************************************************************************/
void Call_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);
        }
}
/*****************************************************************************
** Function name:1s
*****************************************************************************/
void Call_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);
        }
        else firstTime = 1;
#endif //MC_BOARD_ENABLE_OLED
#endif //MC_BOARD_ENABLE_LM75
}
void Appl_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, seg: 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)
}
void MemCopyBYTE(uint8_t *src, uint8_t *dst, uint32_t count)
{
while(count--)*dst++ = *src++;
}
void MemCopyDWORD(uint32_t *src, uint32_t *dst, uint32_t count)
{
while(count--)*dst++ = *src++;
}
/*****************************************************************************
 * MAIN
 *****************************************************************************/
int main (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
        }
}
```

Joystick.h

```
#ifndef __JOYSTICK_H
#define __JOYSTICK_H
#define JOYSTICK_CENTER 0x01
#define JOYSTICK_UP     0x02
#define JOYSTICK_DOWN   0x04
#define JOYSTICK_LEFT   0x08
#define JOYSTICK_RIGHT  0x10
#define JOYSTICK_PORT   2
#define JOYSTICK_PORT_L         3
#define JOY_C_PIN           4
#define JOY_U_PIN           2
#define JOY_D_PIN           3
#define JOY_R_PIN           1
#define JOY_L_PIN           0
/* Enable the interrupts on joystick actions */
#define USE_JOYSTICK_INTERRUPT
void joystick_init (void);
uint8_t joystick_read(void);
#endif /* end __JOYSTICK_H */
```

Joystick.c

```
#include "application.h"
#if MC_BOARD_ENABLE_JOYSTICK == 1
void joystick_init (void)
{
        /*                                HYST   |  PU/PD | FUNC */
        LPC_IOCON->PIO2_4 =      (1<<5) | (2<<3) | (0);
        LPC_IOCON->PIO3_2 =      (1<<5) | (2<<3) | (0);
        LPC_IOCON->PIO3_3 =      (1<<5) | (2<<3) | (0);
        LPC_IOCON->PIO3_1 =      (1<<5) | (2<<3) | (0);
        LPC_IOCON->PIO3_0 =      (1<<5) | (2<<3) | (0);
        /* set the GPIOs as inputs */
   GPIOSetDir( JOYSTICK_PORT, JOY_C_PIN, 0 ); // JOY_CENTER
   GPIOSetDir( JOYSTICK_PORT_L, JOY_U_PIN, 0 ); // JOY_UP
   GPIOSetDir( JOYSTICK_PORT_L, JOY_D_PIN, 0 ); // JOY_DOWN
   GPIOSetDir( JOYSTICK_PORT_L, JOY_L_PIN, 0 ); // JOY_LEFT
   GPIOSetDir( JOYSTICK_PORT_L, JOY_R_PIN, 0 ); // JOY_RIGHT
#ifdef USE_JOYSTICK_INTERRUPT
        /* Setup the interrupt, seq: portNum, bitPosi, sense = edge, single = both,
event*/
        GPIOSetInterrupt(JOYSTICK_PORT, JOY_C_PIN ,0,0,0);
        GPIOSetInterrupt(JOYSTICK_PORT_L, JOY_U_PIN ,0,0,0);
        GPIOSetInterrupt(JOYSTICK_PORT_L, JOY_D_PIN ,0,0,0);
        GPIOSetInterrupt(JOYSTICK_PORT_L, JOY_L_PIN ,0,0,0);
        GPIOSetInterrupt(JOYSTICK_PORT_L, JOY_R_PIN ,0,0,0);
        /* Enable the interrupts */
```

```
        GPIOIntEnable(JOYSTICK_PORT, JOY_C_PIN );
        GPIOIntEnable(JOYSTICK_PORT_L, JOY_U_PIN );
        GPIOIntEnable(JOYSTICK_PORT_L, JOY_D_PIN );
        GPIOIntEnable(JOYSTICK_PORT_L, JOY_L_PIN );
        GPIOIntEnable(JOYSTICK_PORT_L, JOY_R_PIN );
#endif
}
uint8_t joystick_read(void)
{
}
#endif //MC_BOARD_ENABLE_JOYSTICK
```

Handlers.c

```
#include "application.h"
extern volatile uint32_t bldc_Tick;
extern volatile uint32_t bldc_SySTickold;
extern volatile uint32_t bldc_SySTicknew;
extern volatile MOTOR_TypeDef Motor;
extern volatile uint8_t show_graph;
extern volatile uint8_t show_temperature;
volatile uint32_t SysTick_cntr = 0;

#ifndef GPIO_GENERIC_INTS
void PIOINT2_IRQHandler(void)
{
        uint32_t regVal;
        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_C_PIN):
   case (1<<JOY_L_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;
        case (1<<JOY_U_PIN):
                Motor.sp+=50;
                if (Motor.sp >= 4100)
                Motor.sp = 4100;
                 break;
         case (1<<JOY_R_PIN):
                if (Motor.Direction == CW)
                        Motor.Direction = CCW;
                        else
```

```
                        Motor.Direction = CW;
                        break;
                case (1<<JOY_D_PIN):
                        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
void SysTick_Handler (void)
{
        SysTick_cntr++;
        bldc_Tick++;
}
```

- **Lista de Materiales:**

  Para la implementación de este proyecto fueron necesarios los siguientes materiales:

  - 1 Protoboard
  - 1 Motor control kit (incluye LPC1114 y motor BLCD)
  - 4 Botoneras
  - 4 Resistencias de 10KΩ
  - 4 Resistencias de 330Ω
  - 4 Capacitores de 100nF
  - Cable UTP
  - Conectores

3.1.3. Control del motor BLDC utilizando la lpc1769 como interfaz para controlar la lpc1114.

- Descripción:

Para probar la compatibilidad de nuestro proyecto con otros proyectos que no utilizan la LPC1114 sino otras tarjetas tales como la LPC1769, creamos un pequeño programa para controlar el motor utilizando como interfaz la LPC1769.

La LPC1769 recibe una trama de 4 bits los cuales serán utilizados para el control de las funciones del motor BLCD las cuales se detallan en la tabla 3.1.

PIN	BIT	FUNCION
GPIO2.4	0	On/Stop
GPIO2.5	1	Invertir giro
GPIO2.6	2	Incrementa velocidad
GPIO2.7	3	Decrementa velocidad

Tabla 3.1: Descripción de pines de entrada y la función de cada bit.

Las salidas de la Tarjeta LPC1769 serán conectadas con las entradas de la tarjeta PC1114 que controlan las funciones del motor BLCD.

La trama recibida va a ser simulada por cuatro botoneras conectadas a las entradas de la LPC1769.

La tabla 3.2 detalla las conexiones entre tarjetas y sus funciones.

SALIDA LPC1769	ENTRADA LPC1114	FUNCION
GPIO2.0	PIO2.4	On/Stop
GPIO2.1	PIO3.1	Invertir giro
GPIO2.2	PIO3.3	Incrementa velocidad
GPIO2.3	PIO3.2	Decrementa velocidad

Tabla 3.2: conexión entre tarjetas y su función.

- Diagrama de Bloques

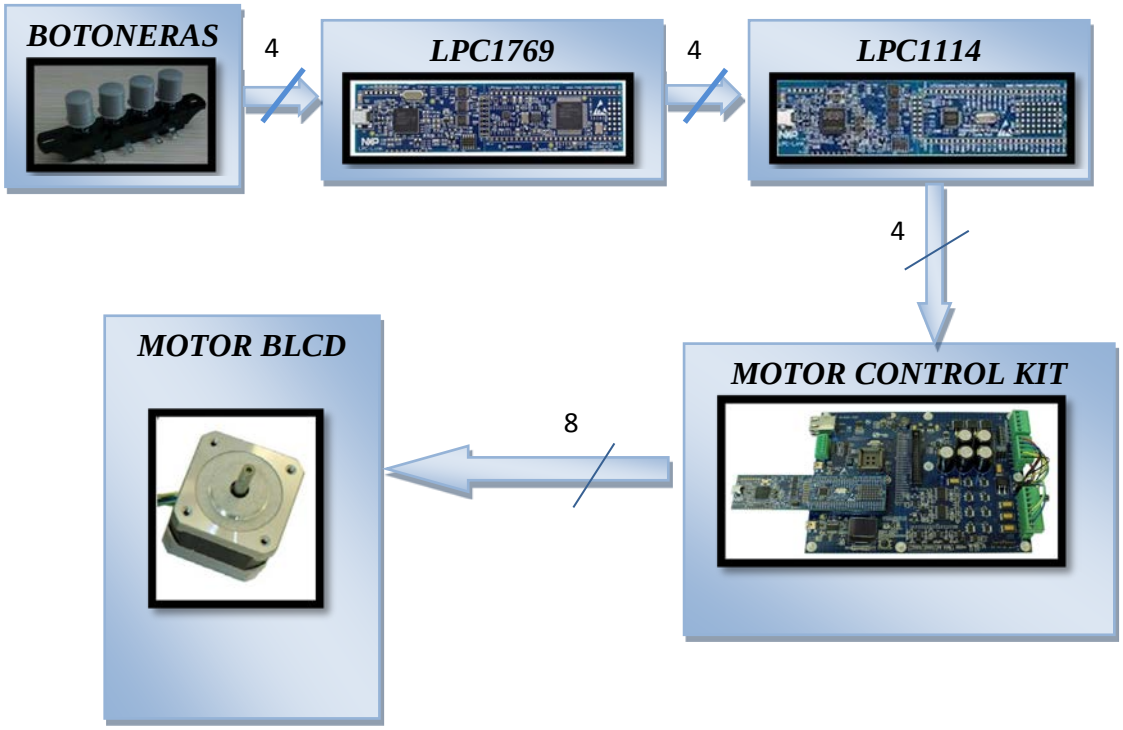

Figura 3.7: Bloques del Control con motor BLDC con LPC1769.

- Diagrama de Flujo

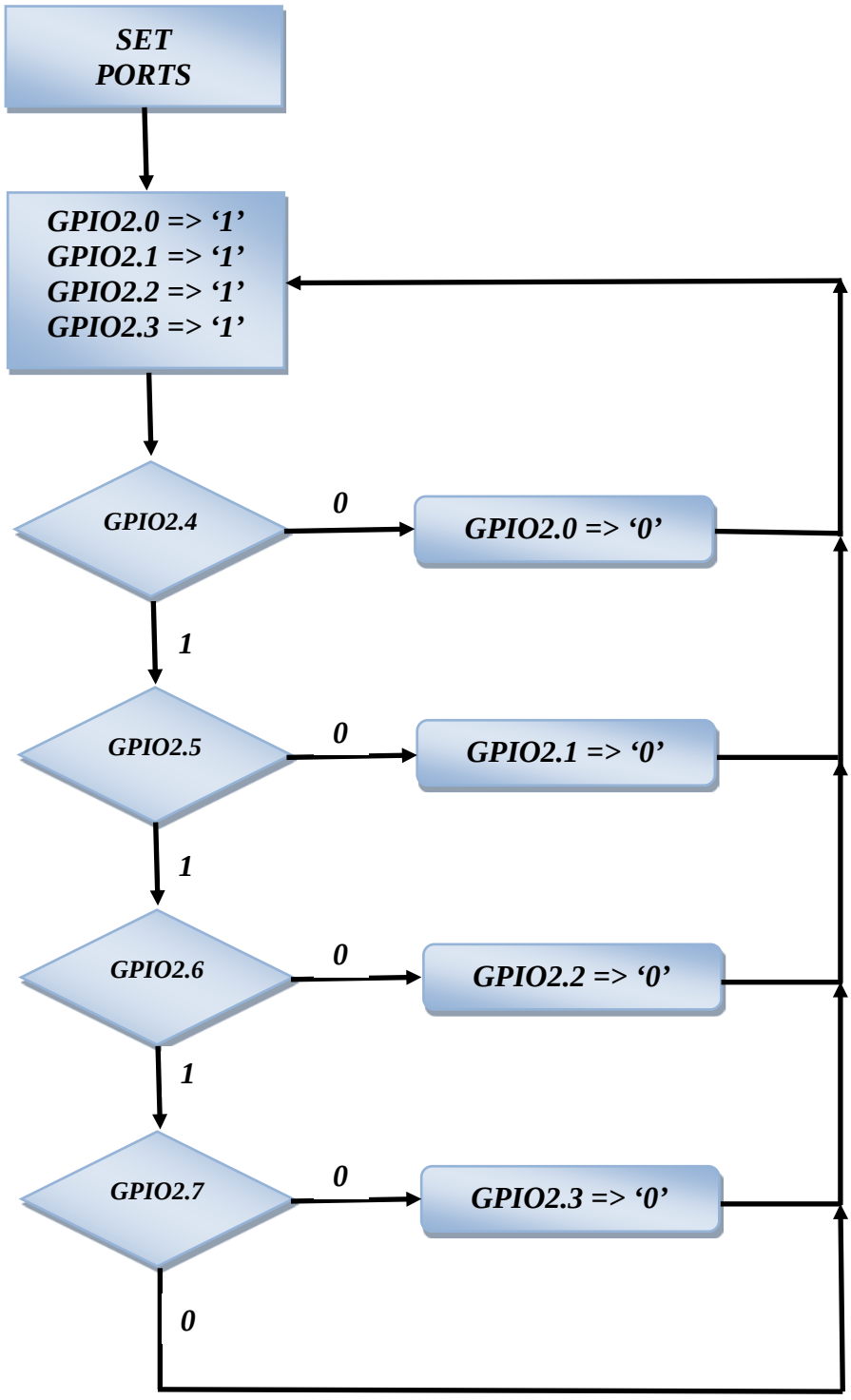

Figura 3.8: Flujo del Control con motor BLDC con LPC1769.

-Descripción del Algoritmo

1) Inicialización de variables.
2) Definición de puertos a utilizar como entradas y salidas, en este caso hemos seleccionado el puerto dos, los primeros cuatro pines serán definidos como salidas y los siguientes cuatro pines serán definidos como entradas, también es necesario asignarle un valor de ‘1’ lógico a los primeros cuatro pines de salida, ya que estamos trabajando con lógica negativa.
3) 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.
4) 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.
5) 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.
6) 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.
7) Los pasos 3 al 6 se repiten indefinidamente ya que este procedimiento se encuentra en un lazo infinito.

-Programa Principal del controlador

INTERFAZ.C

```
#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_t i,j;

  /* SystemClockUpdate() updates the SystemFrequency variable */
  SystemClockUpdate();

  LPC_GPIO2->FIODIR = 0xFFFFFF0F;              /* P2.xx defined as
Outputs */
  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--);
    }
}
```

- **Lista de Materiales:**

  Para la implementación de este proyecto fueron necesarios los siguientes materiales:

  - 1 Protoboard
  - 1 Motor control kit (incluye LPC1114 y motor BLCD)
  - 1 LPC1769
  - 4 Botoneras
  - 4 Resistencias de 10KΩ
  - 4 Resistencias de 330Ω
  - 4 Capacitores de 100nF
  - Cable UTP
  - Conectores

CAPÍTULO 4

PRUEBAS Y SIMULACIONES

4.1. Desarrollo de los Ejercicios

En esta etapa se describen los diferentes modos de operación de los elementos que conforman los ejercicios y su funcionamiento en conjunto para la aplicación implementada. Además se realizó una simulación básica de control de motores BLDC sin escobillas con sensores mediante la plataforma PROTEUS.

4.1.1. Controlar la rotación de Leds.

- Descripción

La primera interacción con el LPCXPRESSO y con la tarjeta LPC1769 fue con la elaboración de un pequeño código para familiarizarnos con la manipulación y configuración de entradas y salidas de la tarjeta LPC1769.

Dicha codificación estaba destinada a controlar una secuencia de luces de 8 led’s los cuales están conectados a 8 salidas de la tarjeta, esta secuencia es controlada por una botonera la cual será nuestra entrada.

La botonera trabaja con lógica negativa:

- Sin pulsar = ‘1’
- Pulsando = ‘0’

Las salidas están asignadas a: GPIO2.0 hasta GPIO2.7

La entrada está asignada a: GPIO2.8

La secuencia 1 se activa por defecto y se la detalla en la siguiente imagen:

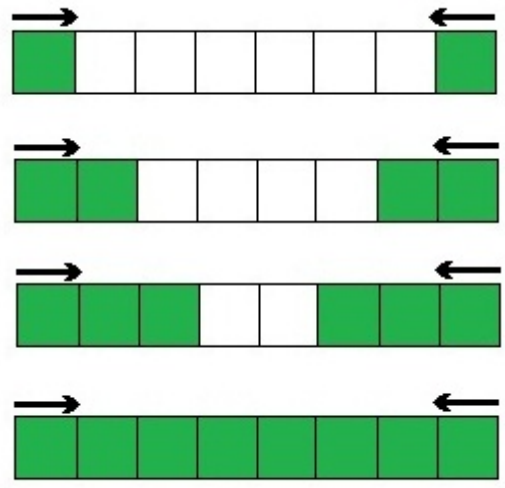

Figura 4.1: Secuencia 1

La secuencia 2 se activa al presionar la botonera y se la detalla en la siguiente imagen:

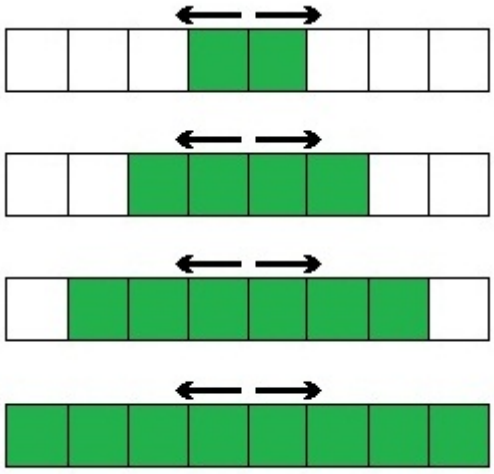

Figura 4.2: Secuencia 2

- **Imagen:**

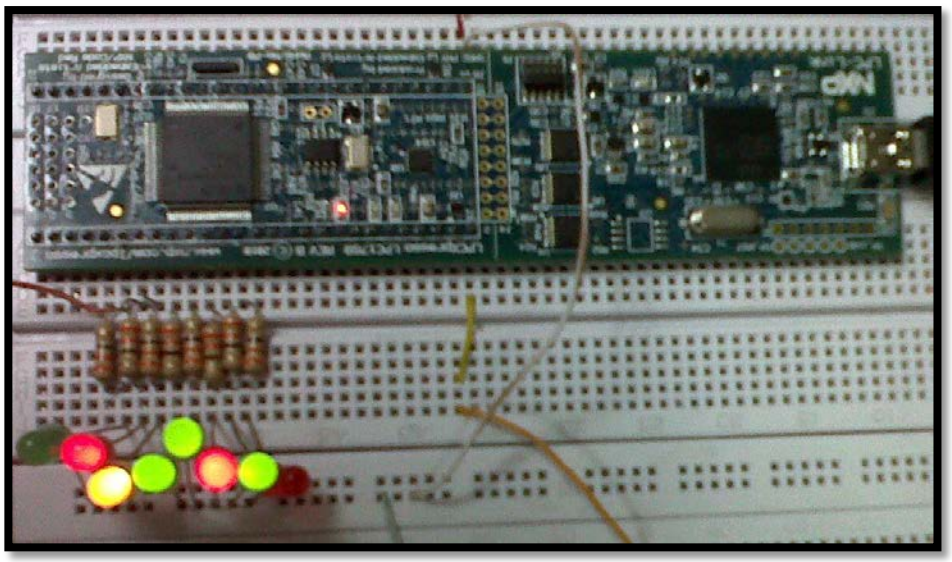

Figura 4.3: Implementación de secuencia de luces.

- **Diagrama de conexiones:**

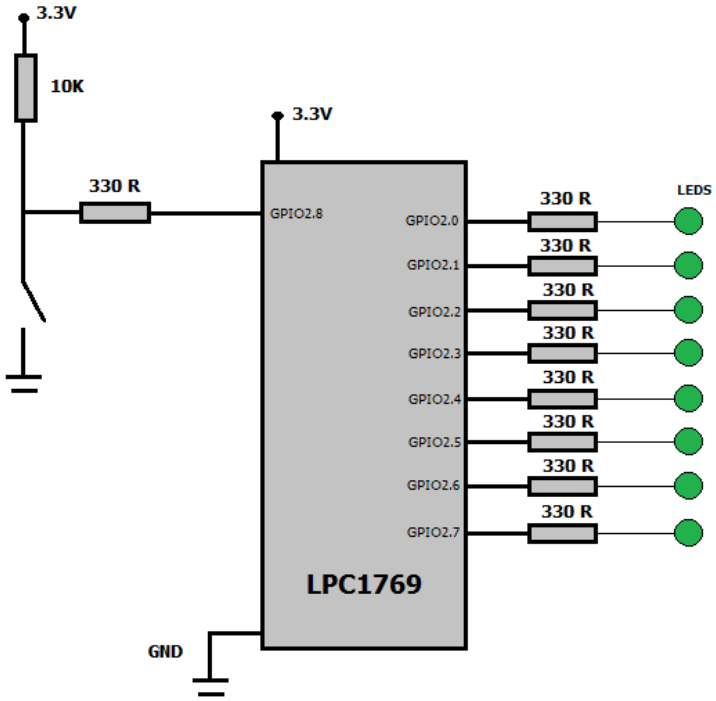

Figura 4.4: Conexiones de secuencia de luces

- **Conclusiones:**

  Pudimos concluir que la tarjeta LPC1769 es una herramienta que nos puede ayudar a solucionar o controlar un sistema, desde aplicaciones simples como es la rotación de Leds hasta aplicaciones industriales. En este ejercicio se utilizó un programa sencillo como es el GPIO para manipular y controlar las rotaciónes de Leds de una manera muy rápida y óptima.

4.1.3. Deshabilitación del joystick y seteo de nuevos pines de entrada para el control del motor BLCD.

- Descripción

El primer obstáculo en la elaboración del presente proyecto fue deshabilitar el joystick y setear nuevas entradas para el control del motor BLCD para que pueda interactuar la tarjeta LPC1114 que viene incluida en el Motor control kit con la tarjeta LPC1769.

Esto se logró cambiando las entradas asignadas al joystick por nuevas disponibles en la LPC1114.

Las nuevas entradas para el control del motor BLCD son:

- PIO2.4 => On/Stop: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD inicia su funcionamiento si está detenido, pero si está girando detiene su funcionamiento.
- PIO3.1 => Invertir giro: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD invierte el sentido de su giro.
- PIO3.2 => Decrementar velocidad: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD decrementa su velocidad en pasos de 50 RPM.

- PIO3.3 => Incrementar velocidad: al notar un cambio de estado de alto a bajo en esta entrada el motor BLCD incrementa su velocidad en pasos de 50 RPM.

El joystick fue sustituido por 4 botoneras que cumplen las funciones antes mencionadas para el control del motor BLCD.

- **Imagen:**

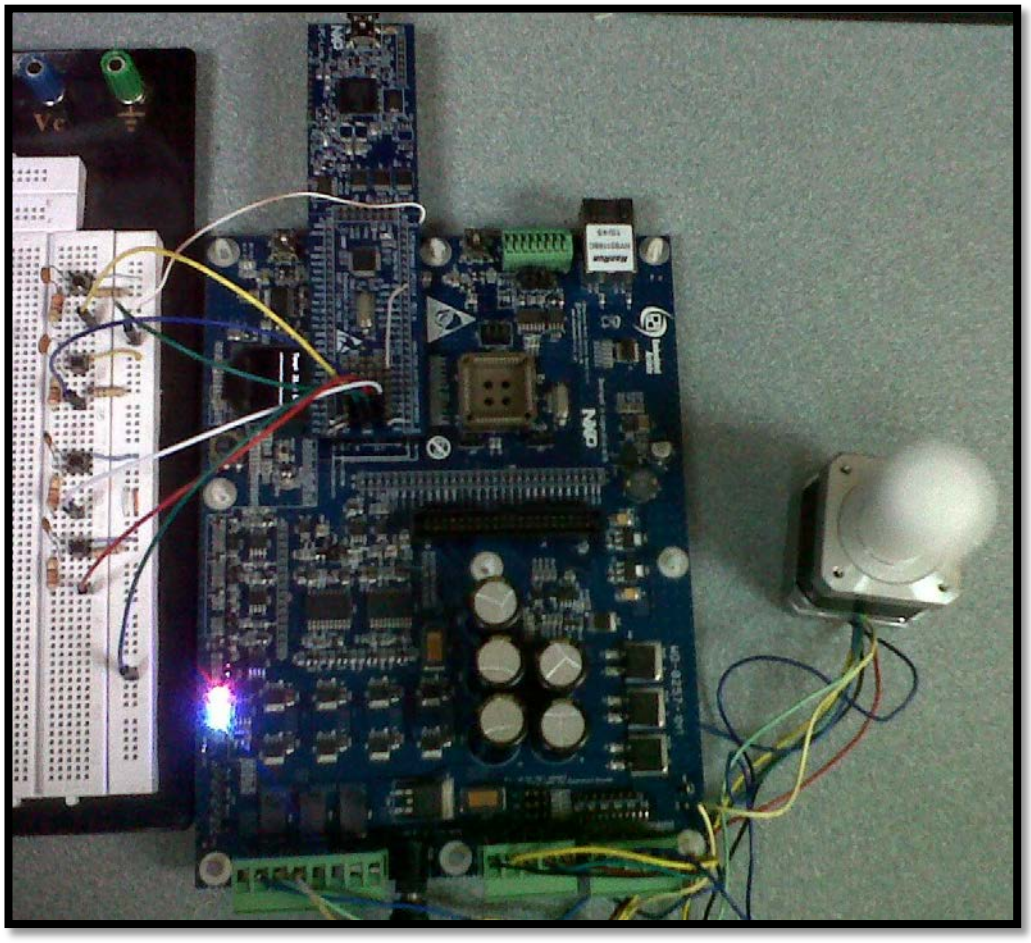

Figura 4.5: Implementación primera etapa.

- **Diagrama de conexiones:**

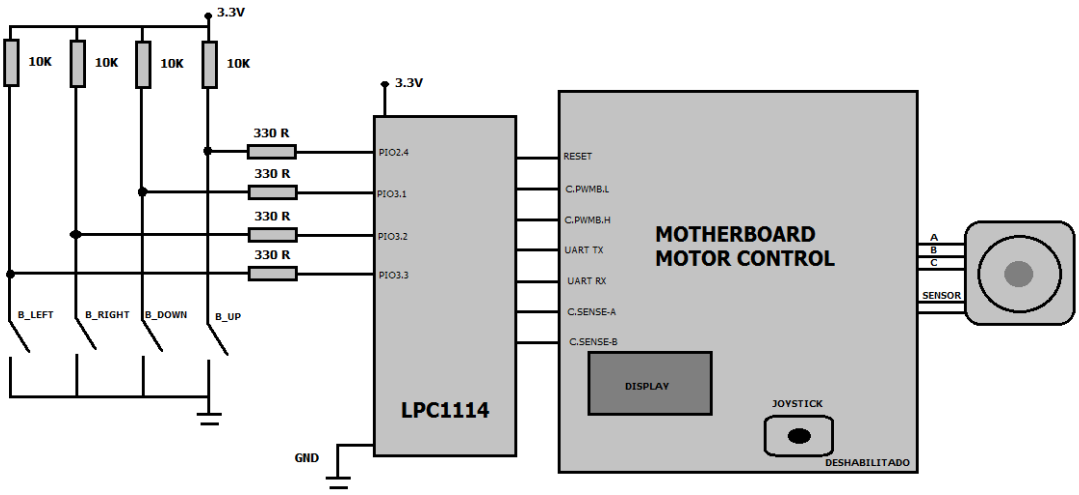

Figura 4.6: Conexiones de la primera etapa

- **Conclusiones:**

  Pudimos concluir, que estudiando y revisando las especificaciones del diagrama esquemático de los puertos disponibles y habilitados para este tipo de tarjetas pudimos deshabilitar el joystick y controlar el motor mediante botoneras externas.

4.1.4. Control del motor BLDC utilizando la lpc1769 como interfaz para controlar la lpc1114.

- **Descripción**

Para probar la compatibilidad de nuestro proyecto con otros proyectos que no utilizan la LPC1114 sino otras tarjetas tales como la LPC1769,

creamos un pequeño programa para controlar el motor utilizando como interfaz la LPC1769.

La LPC1769 recibe una trama de 4 bits los cuales serán utilizados para el control de las funciones del motor BLCD las cuales se detallan en la tabla 3.1.

PIN	BIT	FUNCION
GPIO2.4	0	On/Stop
GPIO2.5	1	Invertir giro
GPIO2.6	2	Incrementa velocidad
GPIO2.7	3	Decrementa velocidad

Tabla 4.1: Descripción de pines de entrada y la función de cada bit.

Las salidas de la Tarjeta LPC1769 serán conectadas con las entradas de la tarjeta PC1114 que controlan las funciones del motor BLCD. La trama recibida va a ser simulada por cuatro botoneras conectadas a las entradas de la LPC1769.La tabla 4.2 detalla las conexiones entre tarjetas y sus funciones.

SALIDA LPC1769	ENTRADA LPC1114	FUNCION
GPIO2.0	PIO2.4	On/Stop
GPIO2.1	PIO3.1	Invertir giro
GPIO2.2	PIO3.3	Incrementa velocidad
GPIO2.3	PIO3.2	Decrementa velocidad

Tabla 4.2: conexión entre tarjetas y su función.

- **Imagen:**

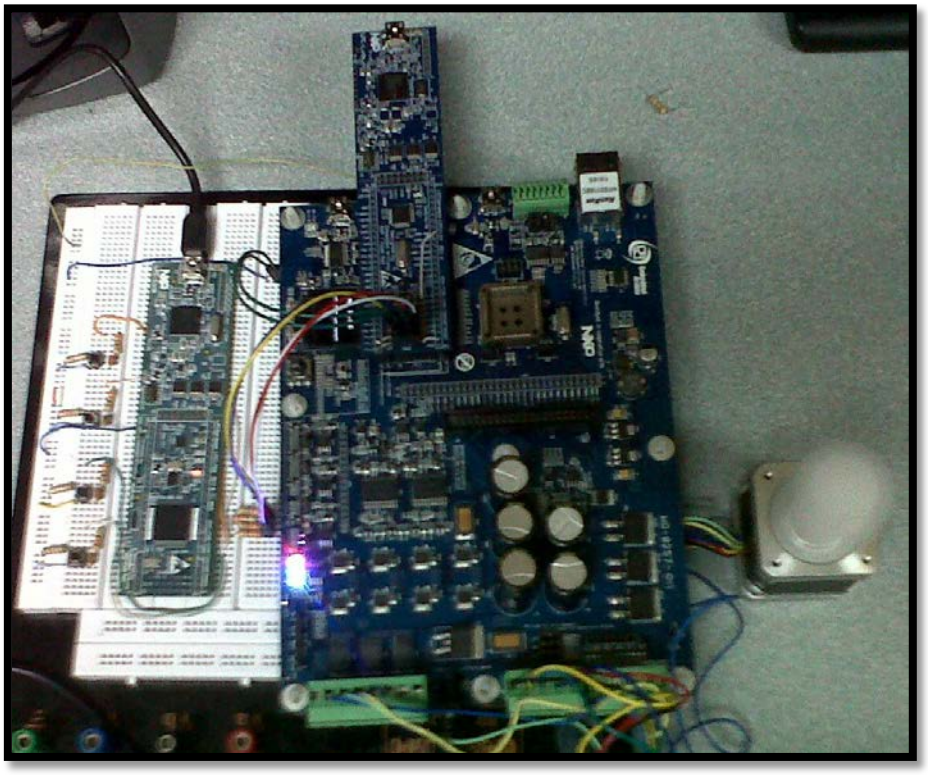

Figura 4.7.: Implementación etapa final

- **Diagrama de conexiones:**

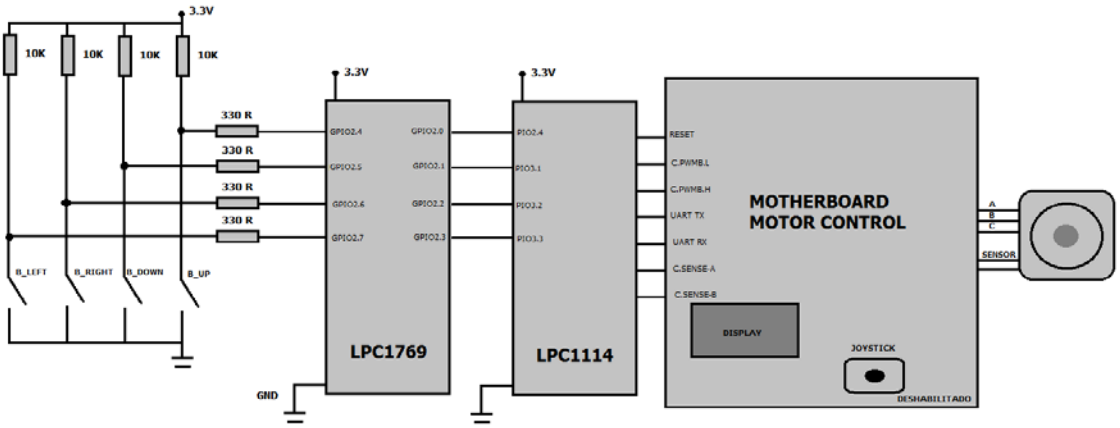

Figura 4.8: Conexiones de la etapa final.

- **Conclusiones:**

  Llegamos a la conclusión que debido a las diferentes tecnologías de ambas tarjetas LPC1769 con Cortex3 y LPC1114 con Cortex0 fue un poco difícil poder comunicarlas entre si debido a las diferentes librerías que manejan cada una de estas tarjetas pero al final llegamos a desarrollar bien el interfaz de comunicación y solucionar este problema para futuras aplicaciones.

4.2. Simulación en PROTEUS

En esta etapa se pueden observar que las simulaciones que pudimos hacer para nuestro proyecto fue el analizar el control de un motor BLDC con efecto hall que encontramos en la herramienta PROTEUS que se realizaron previos

a la implementación del proyecto en físico y que nos ayudó a tener una mejor idea de sus modos de operación y correctos funcionamientos.

4.2.1. Simulación de control de un Motor BLDC

Para guiarnos en el desarrollo del control del motor BLDC, especialmente en el control de la velocidad utilizando PWM nos ayudamos de la AN857 que nos proporciona el software PROTEUS como ejemplo de diseño.

En esta simulación describe los pasos del desarrollo de varios controladores de motores sin escobillas. Cubrimos sensores, bucle de sensores, abierto, y el diseño de circuito cerrado. Hay incluso un controlador con una tensión independiente y controla la velocidad para que pueda descubrir las características de su motor de forma empírica. El código en este ejercicio de aplicación se ha desarrollado con PIC16F877 de Microchip PICmicro en conjuntamente con el depurador en circuito (ICD). Esta combinación fue elegida debido a que el ICD es de bajo costo, y el código se puede depurar en el prototipo sin necesidad de un programador extra o emulador. Como el diseño se desarrolla, se programa el dispositivo de destino y ejercer el código directamente desde la MPLAB. El código final puede ser uno de los más pequeño, menos costoso, PICmicro microcontroladores. La portabilidad tiene un mínimo de esfuerzo, porque el conjunto de instrucciones es idéntico para todos PICmicro de 14 bits de dispositivos básicos. También hay que señalar que el código se ha probado y

optimizado para una Pittman N2311A011 corriente continua sin escobillas del motor. Otros motores también fueron probados para asegurar que el código es generalmente útil.

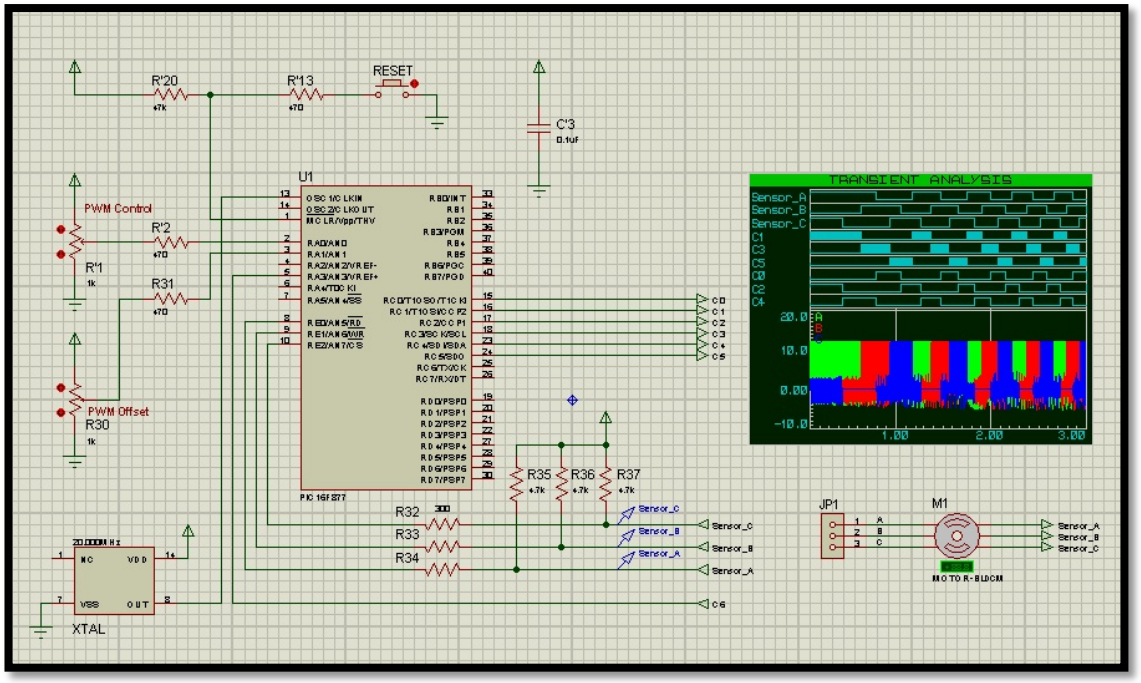

Figura 4.9: Simulación en Proteus.

Colocando un osciloscopio para visualizar las fases A, B y C en PWM obtuvimos las siguientes figuras.

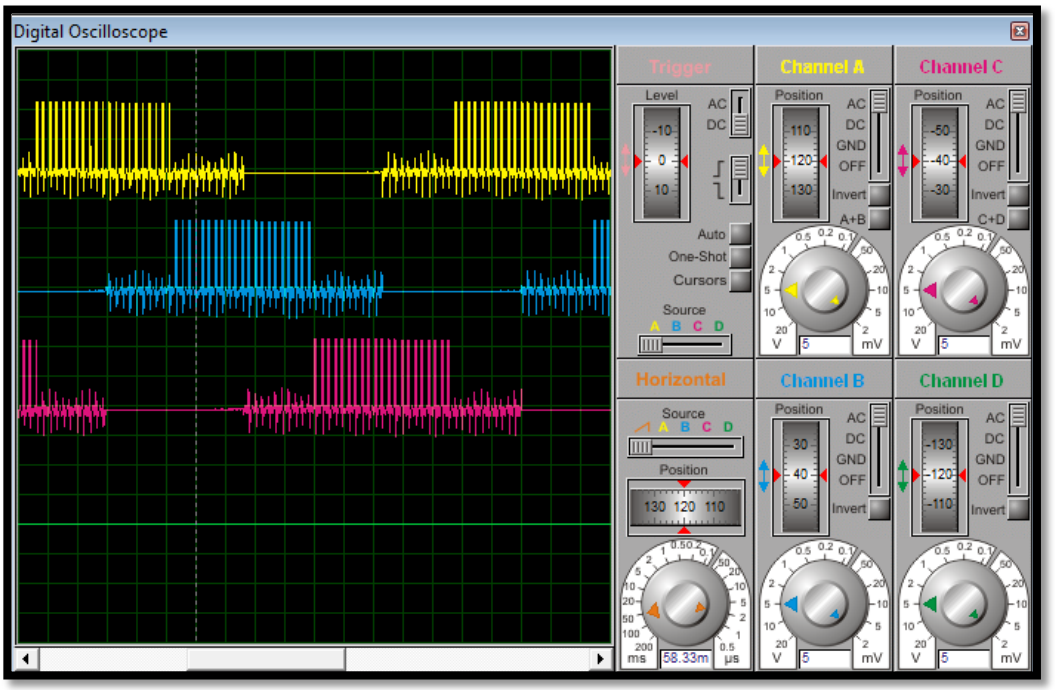

Figura 4.10: PWM al 20%

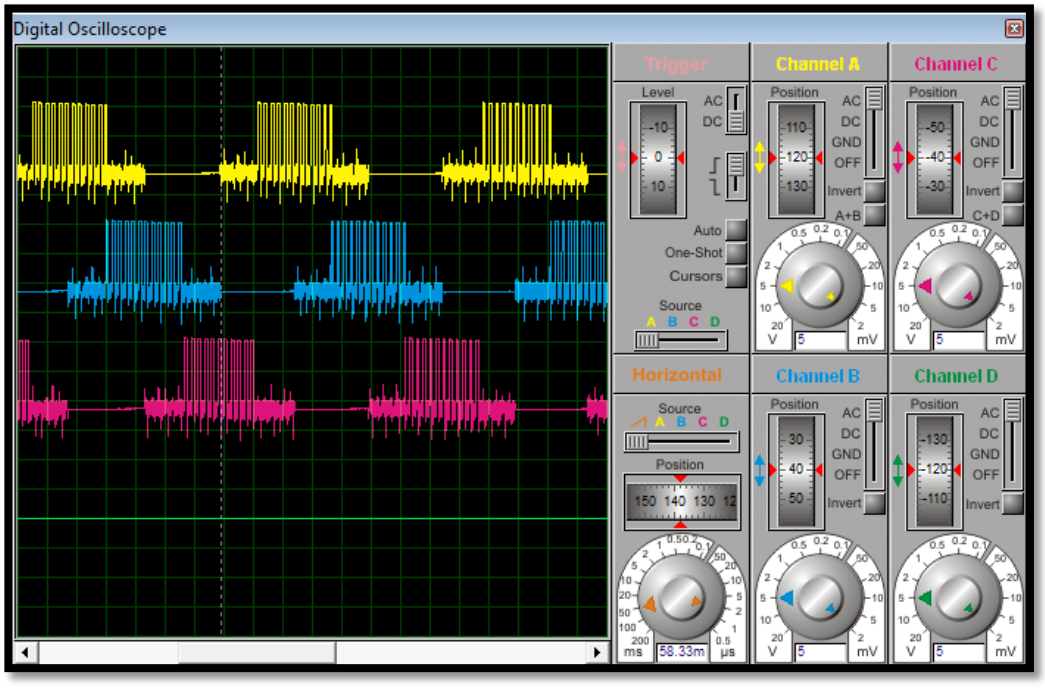

Figura 4.11: PWM al 40%

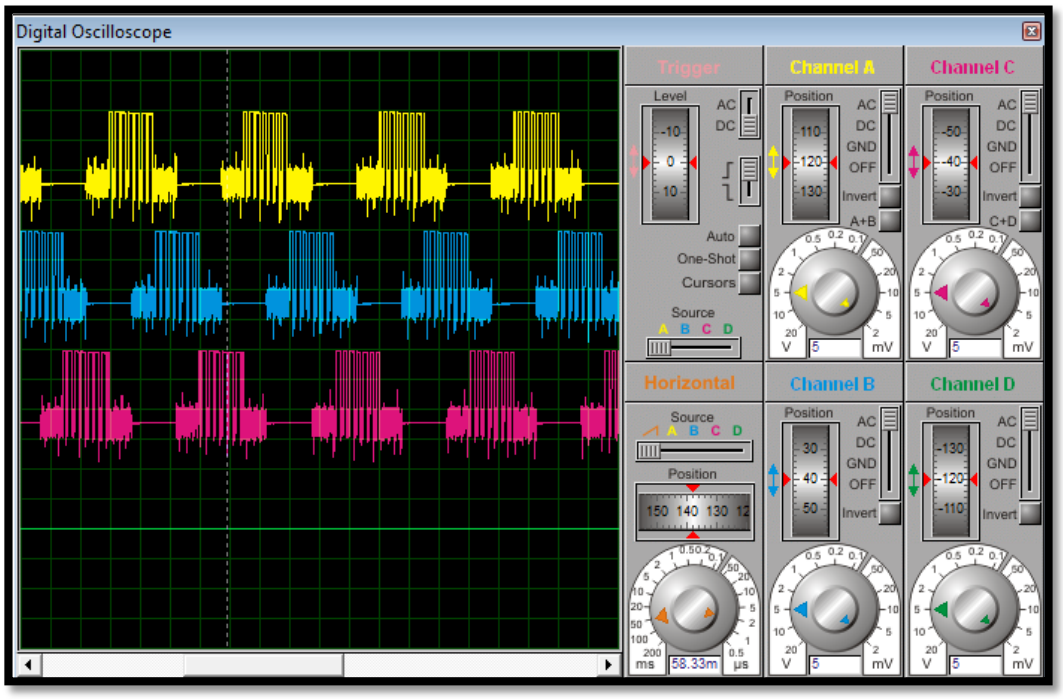

Figura 4.12: PWM al 60%

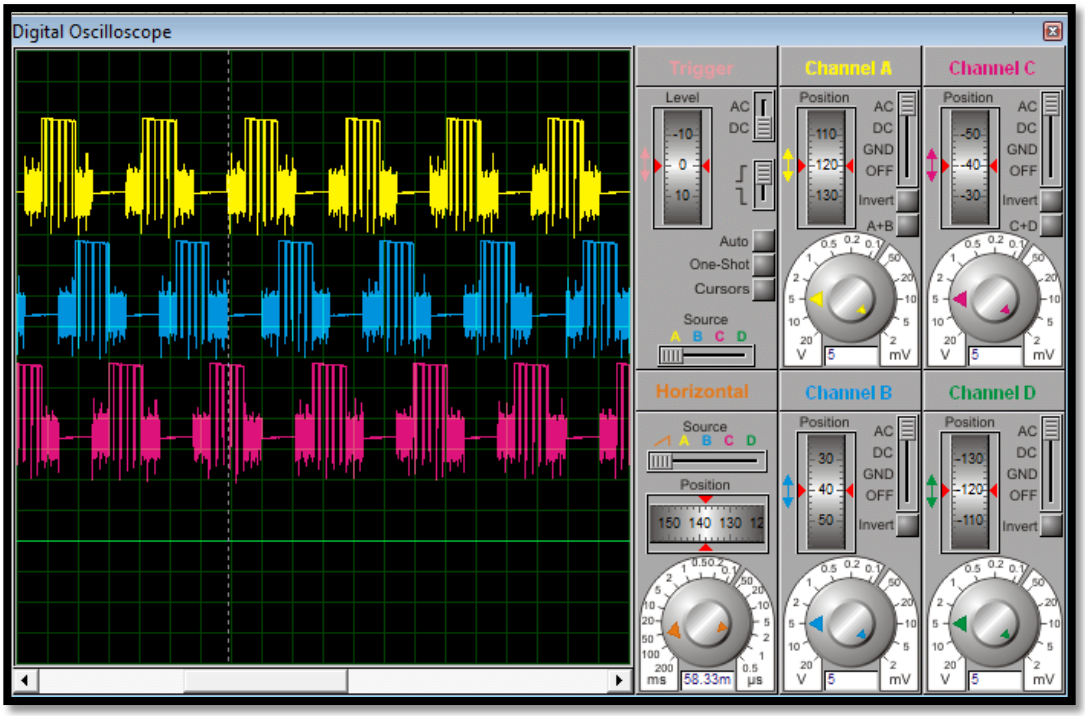

Figura 4.13: PWM al 80%

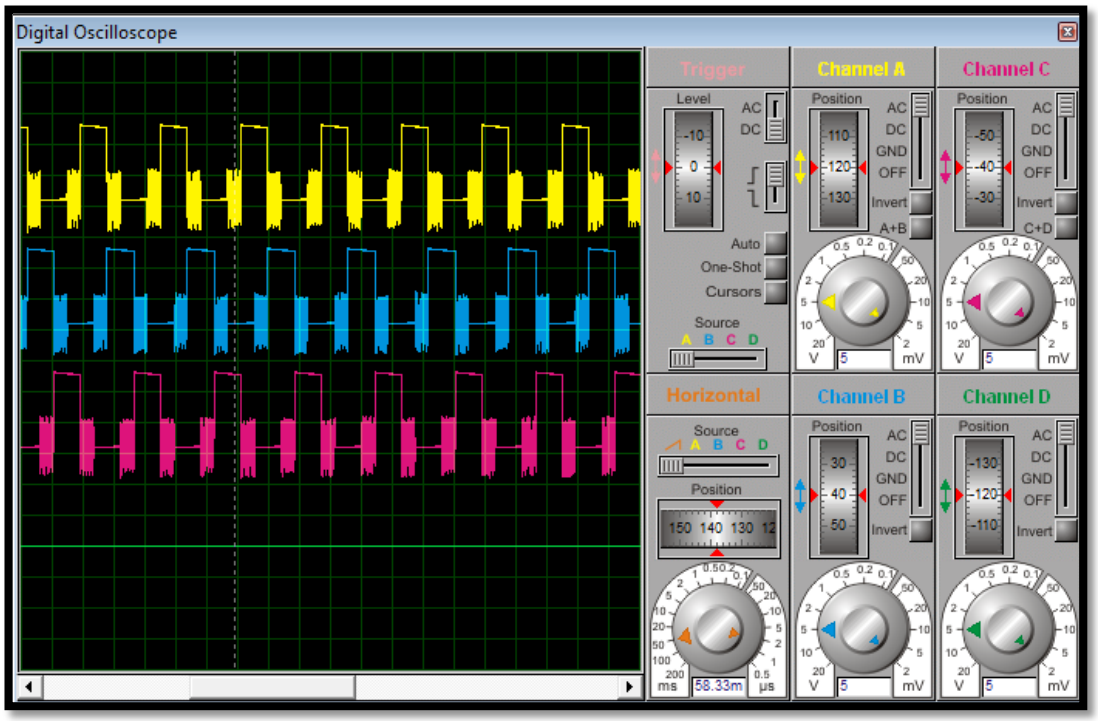

Figura 4.14: PWM al 100%

CONCLUSIONES

Las conclusiones que logramos obtener del proyecto son las siguientes:

1) En el siguiente trabajo en primera instancia no teníamos ningún conocimiento de este tipo de tecnología como es la de NXP usando el microcontrolador ARM Cortex3 con 32 bits de LPCXPRESSO pero estudiando, analizando y realizando pruebas con manuales y datasheet pudimos llegar a controlar un motor sin escobillas (BLDC) y con sensores de una manera muy sencilla y eficiente. De esta manera solucionamos el problema del control de los motores BLDC para que así los demás proyectos puedan hacer uso del mismo de una manera muy fácil o en aplicaciones futuras.

2) La modificación de un programa base nos ayudó a comprender y entender los diferentes conjunto de ejercicios que implementamos así de esta manera se simplificó en gran medida debido a que los comandos o instrucciones necesarios para operar el motor; son

específicos y relativamente simples. Bastó entender el funcionamiento de los mismos y ponerlos en práctica para así desarrollar cada uno de los ejercicios.

3) Ya en la implementación del proyecto, las modificaciones que realizamos para deshabilitar el joystick y nuevas configuraciones de la codificación del control del motor que trae el kit del motor con su tarjeta lpc1114 por default fueron en realidad lo más destacable en lo que al control se refiere y al desarrollo de los ejercicios; ya que serían lo que cualquier ingeniero hubiera implementado si se encontrara´ en la necesidad de diseñar un protocolo rápido, eficaz y sencillo.

4) Con el desarrollo de los ejercicios y su implementación práctica pudimos darnos cuenta de las ventajas y desventajas mencionadas a lo largo del trabajo que implica el uso de las librerías para la comunicación entre los diferentes dispositivos o tarjetas LPCXPRESO. Como por ejemplo: ciertas librerías no son incluidas en la LPCXPRESSO 1114 que necesita la lpc1769. Podríamos concluir así entonces que todo dependerá de la aplicación que estemos desarrollando y el uso que le queramos dar en la aplicación.

RECOMENDACIONES

Las recomendaciones que logramos obtener del proyecto son las siguientes:

1) Hay algunas cosas a tener en cuenta cuando se trata de poner en práctica el control de un motor sin escobillas mediante la lpc1769. Una de estas cuestiones es el cableado. Es importante asegurarse de que la entrada y salidas de cada uno de los pines en el dispositivo está conectada correctamente entrada de bus del microcontrolador ARM Cortex3 con 32 bits de LPCXpresso.

2) Por otra parte, muchos dispositivos externos, tales como el kit del motor tienen pines que puede desactivar el dispositivo de forma selectiva (para su uso en sistemas más grandes), es importante tomar

nota de los pines que no están siendo utilizados así como de los pines que están en uso.

3) Para poder obtener una excelente comunicación entre los microcontroladores utilizados es importante trabajar con una alimentación de 3.3V, ya que al trabajar con estos valores e garantiza el correcto funcionamiento de las tarjetas y por ende no dañar las mismas.

ANEXOS

Guía para programar el LPCXPRESSO

La aplicación de demostración debe ser descargada a la LPCXPRESSO LPC1114. Esta es una descripción de cómo descargar un nuevo archivo / aplicación actualizada a la versión parcial de programa LPCXPRESSO LPC1114.

Primero asegúrese de que la última versión del IDE LPCXPRESSO está instalada. El LPCXPRESSO LPC1114 placa se puede programar independiente o insertarse en la Junta de Control de Motores LPCXPRESSO. Si se monta en el último, asegúrese de que la fuente de alimentación externa está conectada y alimentación de 24V a la placa.

En segundo lugar, importar el proyecto de aplicación de demostración en el área de trabajo de Eclipse. La aplicación de demostración se puede descargar desde la página de soporte integrado Artistas después de registrar el producto.

En tercer lugar, haga clic en el "Programa de Flash" icono de la barra de herramientas, vea la imagen abajo. El icono puede estar en diferentes lugares dependiendo del tamaño de ventana.

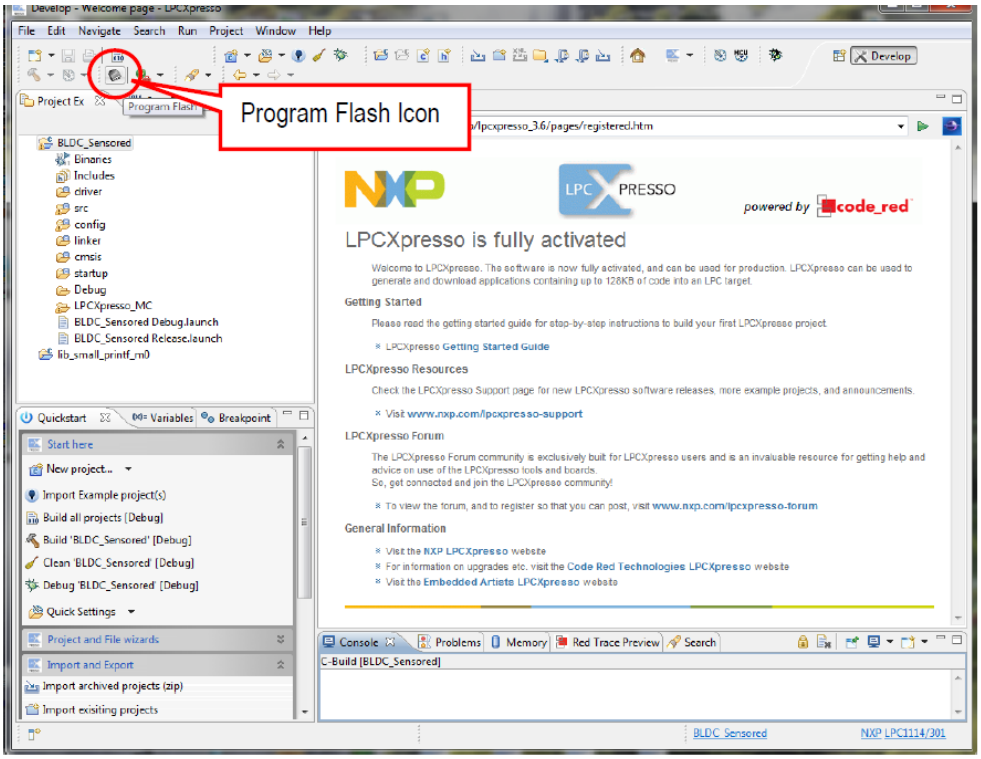

Figura 1.

El siguiente paso es seleccionar el procesador para descargar. Seleccione LPC1114/301 de la lista que se presenta. A continuación, pulse el botón OK. Tenga en cuenta que este paso a veces no es necesario porque el IDE se puede LPCXpresso detectar el procesador que está conectado.

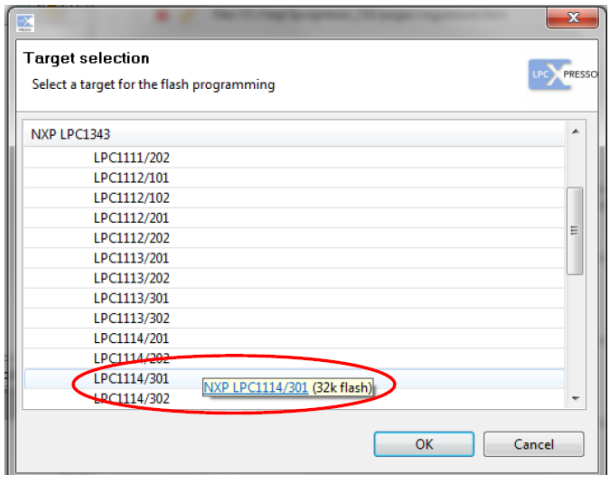

Figura 2.

El siguiente paso es buscar el archivo para descargar. Pulse el botón "Examinar".

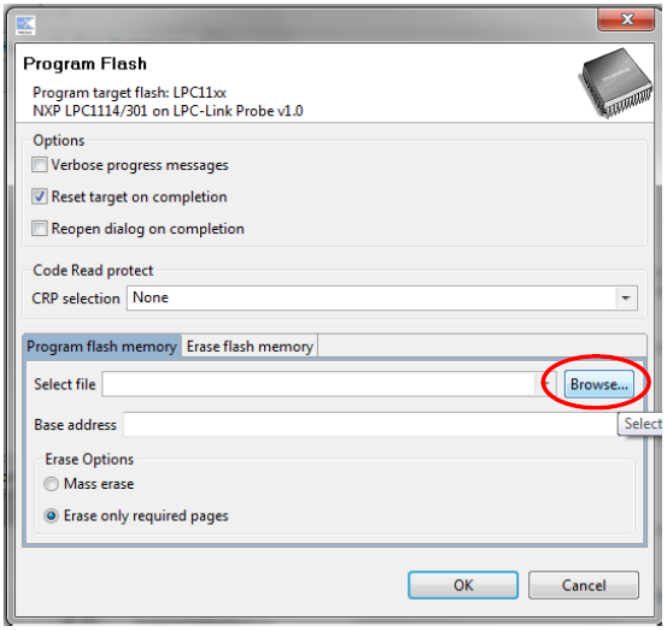

Figura 3.

Buscar en el directorio de proyectos y luego "Debug". En esta subcarpeta no es un archivo que termina con *. Axf o * bin.. Seleccione uno de estos archivos. Pulse el botón "Abrir".

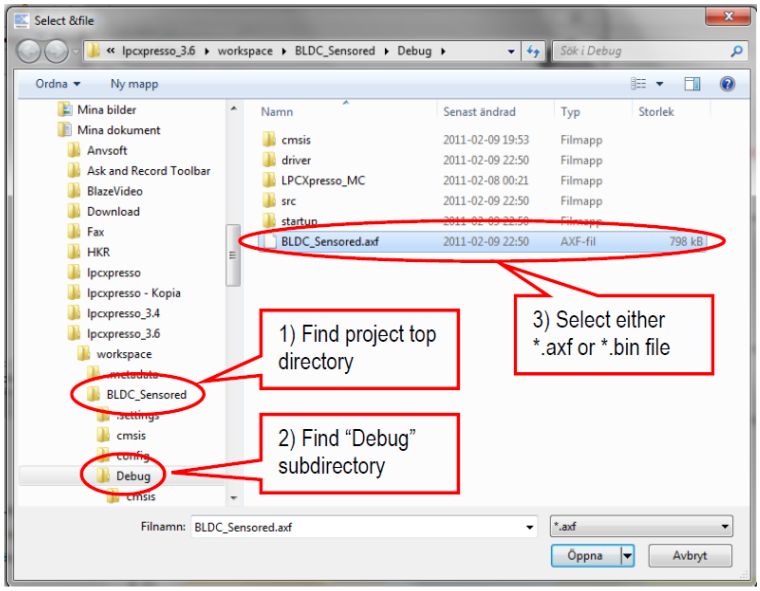

Figura 4.

La siguiente ventana nos muestra el proceso de flash:

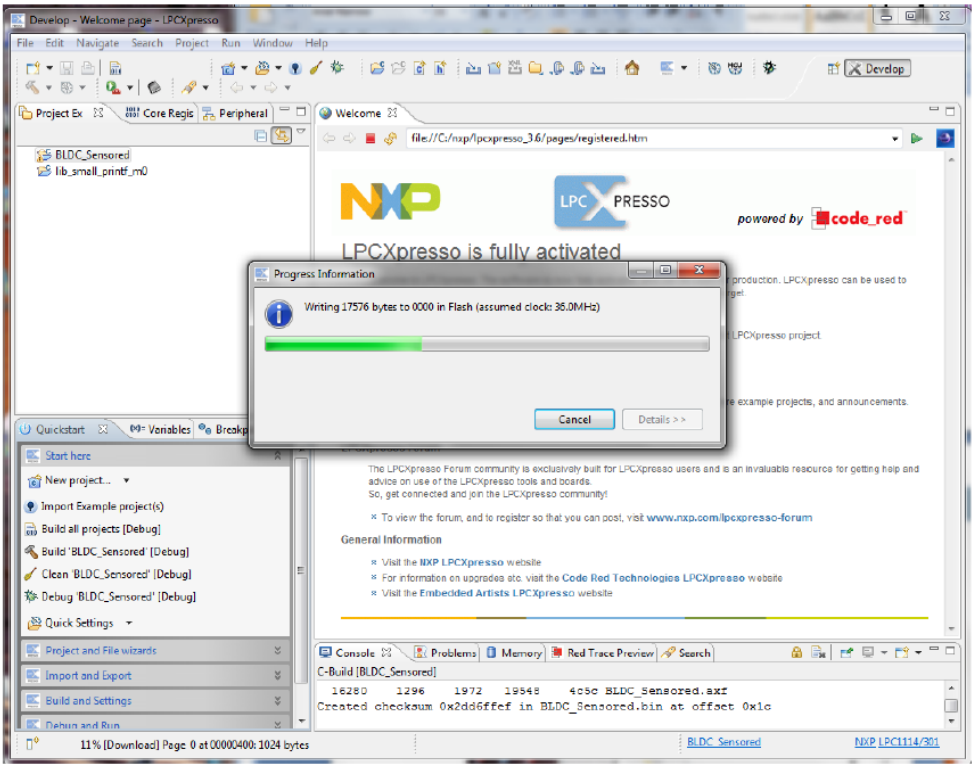

Figura 5.

Hay una forma alternativa de iniciar el proceso de descarga del programa. Desde el área de trabajo, haga clic derecho sobre el archivo *. Axf o *. Bin (que se encuentra bajo la "depuración" subdirectorio). A continuación, seleccione "Utilidad binario" y "Programa de Flash".

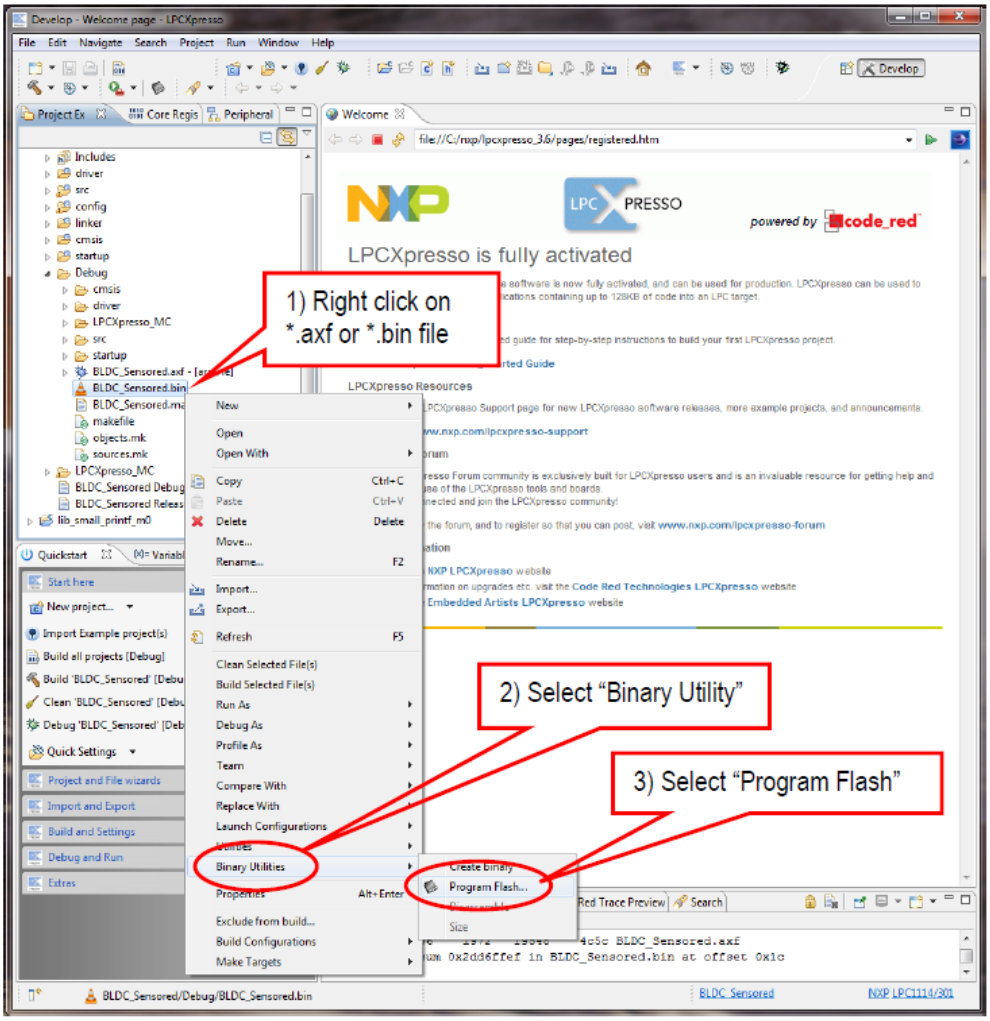

Figura 6.

Desconecte el cable USB a la LPCXpresso LPC1114 bordo y pulse el pulsador RESET. Un mensaje de inicio se debe mostrar en la pantalla OLED.

La Junta de Control de Motores LPCXpresso contiene un chip USB a UART puente (FT232R del FTDI) que conecta a un canal de la UART el control de MCU a un puerto COM virtual en el PC / ordenador portátil (mediante USB). Este canal UART se utiliza normalmente como el canal de la mesa para las aplicaciones. Salida printf () puede por ejemplo ser dirigida a este canal UART. El canal UART también se puede utilizar para el envío de comandos de control de motor para el control MCU. Un controlador USB debe estar instalado en el PC / portátil para que el puerto COM virtual que se creará. Consulte las guías de FTDI instalación para obtener detalles de cómo instalar el controlador para diferentes sistemas operativos

COMPILACION DE UN EJEMPLO

En esta sección se describe cómo compilar la aplicación de demostración para la LPCXPRESSO LPC1114.
Primero asegúrese de que la última versión del IDE LPCXPRESSO está instalada.
En segundo lugar, importar el proyecto de aplicación de demostración en el área de trabajo de Eclipse. La aplicación de demostración se puede descargar (como un archivo .zip de la página de soporte integrado Artistas después de registrar el producto. El archivo

zip contiene todos los archivos del proyecto y es la manera sencilla de distribuir los proyectos completos de Eclipse. Seleccione la pestaña Importar y Exportar en el menú de Inicio rápido y luego importarlos proyectos archivados (ZIP), ver la siguiente figura.

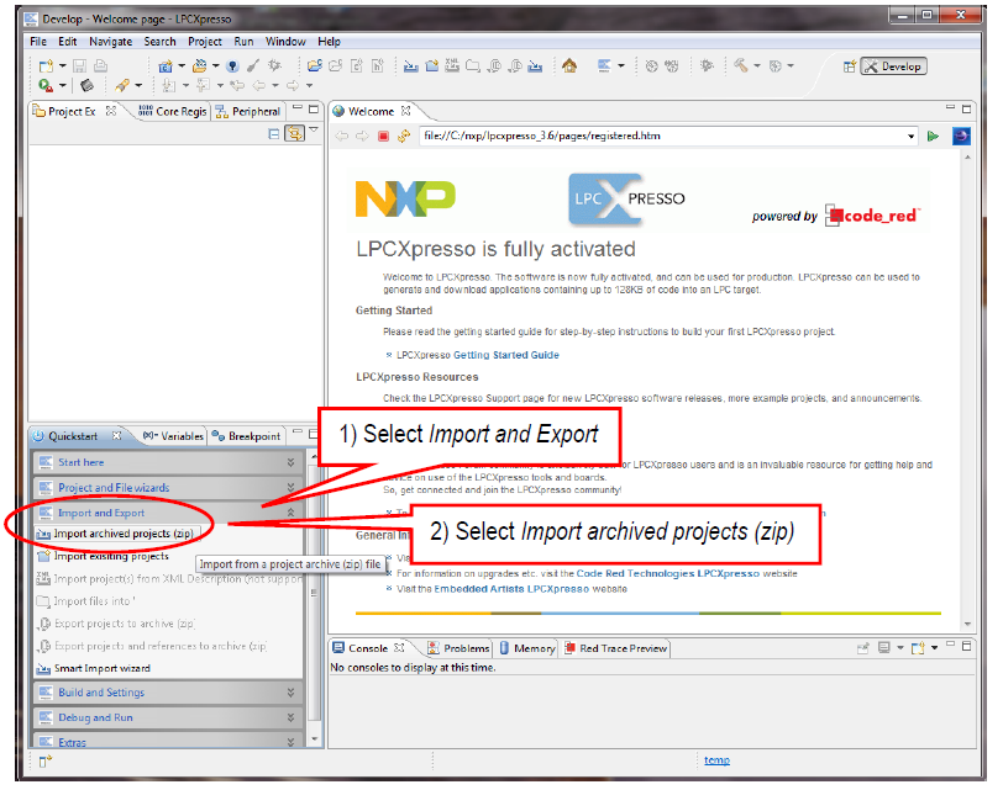

Figura 7.

A continuación, busque y seleccione el archivo zip descargado que contiene la aplicación de demostración archivado. Asegúrese de que ambos sub-proyectos son seleccionados, consulte la siguiente figura.

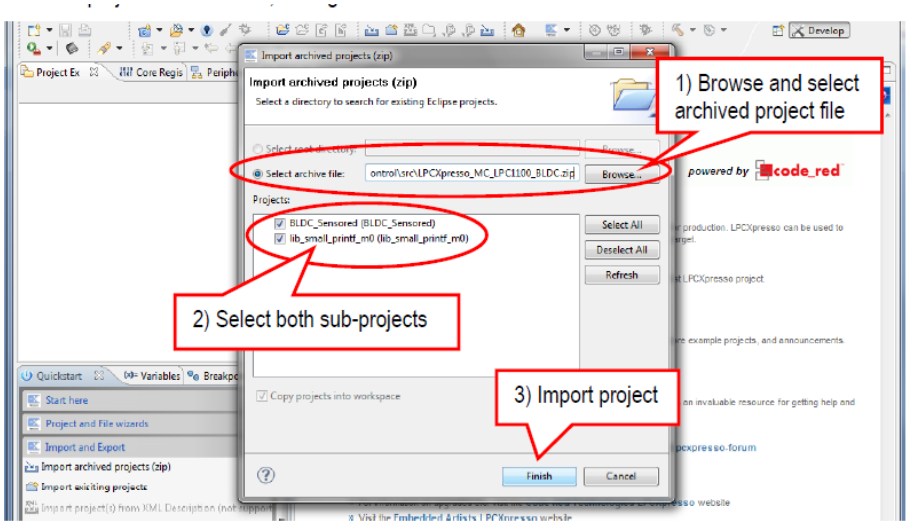

Figura 8.

El proyecto actualmente se importa. Haga clic (para seleccionar) el proyecto principal. Ver y editar los archivos del proyecto. Construir / limpieza / depurar el proyecto en el menú de inicio rápido (clic aquí), ver foto de abajo. Cuando se depura un proyecto, asegúrese de que el LPCXPRESSO LPC1114 tarjeta se conecta vía USB a la PC ya que la aplicación se descargará a la Junta a través de LPC-LINK (SWD / interfaz JTAG).

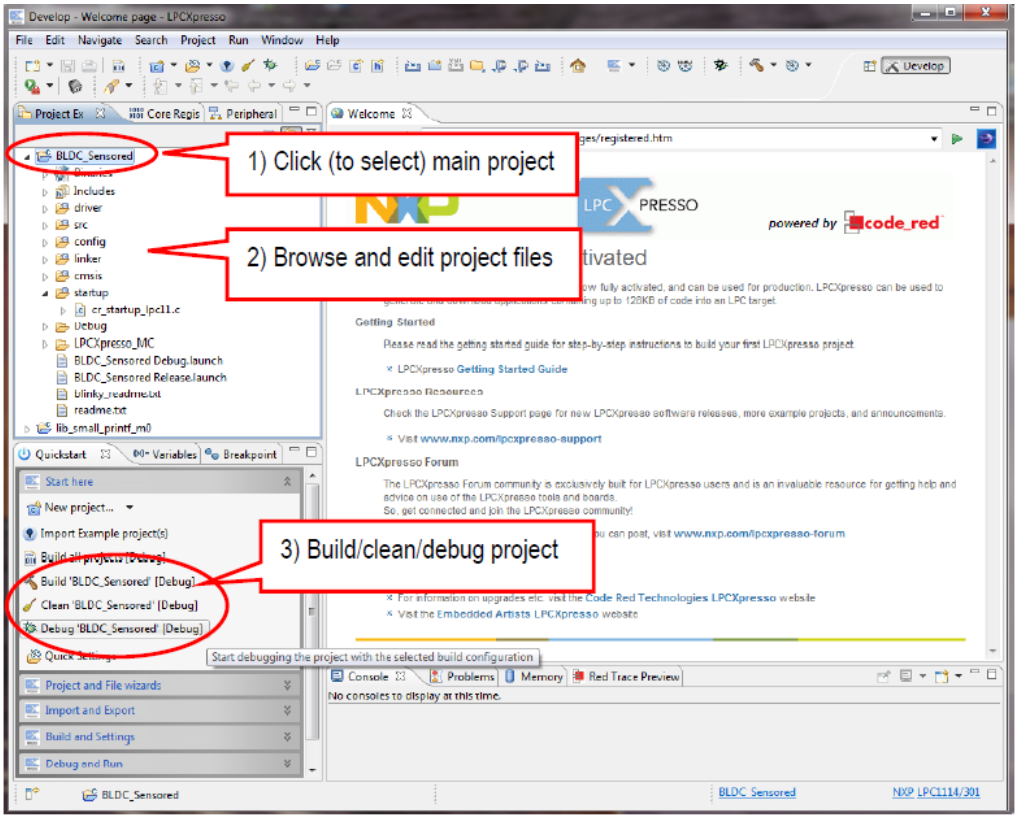

Figura 9.

LPC compatibles a los productos y los número que forman parte de la plataforma LPCXpresso son:

- **LPC1100:** LPC1102 LPC1111/101 LPC1111/102 LPC1111/201 LPC1111/202 LPC1112/101 LPC1112/102 LPC1112/201 LPC1112/202 LPC1113/201 LPC1113/202 LPC1113/301 LPC1113/302 LPC1114/201 LPC1114/202 LPC1114/301 LPC1114/302 LPC11C12/301 LPC11C14/301 LPC11C22/301

LPC11C24/301 LPC11U12/201 LPC11U13/201 LPC11U14/201 OM11049: LPC1114/302 OM13014: LPC11U14 OM13012: LPC11C24

- **LPC1200:** LPC1224/201 LPC1224/221 LPC1225/301 LPC1225/321 LPC1226/301 LPC1227/301

  OM13008: LPC1227
- **LPC1300:** LPC1311 LPC1311/01 LPC1313 LPC1313/01 LPC1342 LPC1343
- OM11048: LPC1343
- **LPC1700:** LPC1751 LPC1752 LPC1754 LPC1756 LPC1758 LPC1759 LPC1763 LPC1764 LPC1765 LPC1766 LPC1767 LPC1768 LPC1769 LPC1774 LPC1776 LPC1777 LPC1778 LPC1785 LPC1786 LPC1787 LPC1788

  OM13000: LPC1769
- **LPC1800:** Coming soon
- **LPC2000:** LPC2109 LPC2109/01 LPC2134 LPC2142 LPC2362 LPC2929
- **LPC3000:** LPC3130 LPC3250
- **LPC4000:** Coming soon

BIBLIOGRAFÍA

[1]Motores de Corriente Continua

(DC). http://www.todorobot.com.ar/documentos/dc-motor.pdf

Fecha de Consulta: 18/febrero/2012

Autor: s/a todorobot.com

[2] “Informe sobre motores” Departamento DSIE de la Universidad Politécnica de Cartagena.

www.masteringenieros.com/master/Ficheros/File/motor.pdf

Fecha de Consulta: 5/marzo/2012

Autor: Padmaraja Yedamale Microchip Technology Inc.

[3] Datasheet AN885 Microchip

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en012127

Fecha de Consulta: 18/marzo/2012

Autor: Padmaraja Yedamale Microchip Technology Inc.

[4] Msc. Jianwen Shao. “Direct Back EMF Detection Method for Sensorless Brushless DC(BLDC) Motor Drives”. Tesis, Virginia Polytechnic Institute and the State University.

http://scholar.lib.vt.edu/theses/available/etd-09152003-171904/unrestricted/T.pdf

Fecha de Consulta: 20/marzo/2012

Autor: Msc. Jianwen Shao.

[5] Agustin Llamas. “Montacargas Automático”. Tesis, ESIME

http://itzamna.bnct.ipn.mx:8080/dspace/handle/123456789/82

Fecha de Consulta: 25/marzo/2012

Autor: Agustin Llamas.

[6] Datasheet AN857 Microchip.

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en012037

Fecha de Consulta: 5/abril/2012

Autor: Ward Brown Microchip Technology Inc.

[7] LPCXpresso Motor Control Kit UserManual

http://www.nxp.com/documents/other/LPCXpresso_Motor_Control_Kit_UserManual.pdf

Fecha de Consulta: 26/marzo/2012

Autor: David shallsgatan 16 211 45 Malmö Sweden.

[8] NXP LPC1114

Datasheet http://ics.nxp.com/products/lpc1000/datasheet/lpc1110.lpc1111.lpc1112.lpc1113.lpc1114.pdf

Fecha de Consulta: 27/marzo/2012

Autor: David shallsgatan 16 211 45 Malmö Sweden.

[9] BLDC motor control with

LPC1769 http://www.nxp.com/documents/application_note/AN10898.pdf

Fecha de Consulta: 27/marzo/2012

Autor : David shallsgatan 16 211 45 Malmö Sweden.

[10] Getting started with NXP LPCXpresso

http://www.nxp.com/documents/other/LPCXpresso_Getting_Started_Guide.pdf

Fecha de Consulta: 5/abril/2012

Autor: Harbison, S.P. & Steele, G.L..