Librería C para el Control de un Motor BLDC basado en un ATmega64M1 para Docencia

Himar A. Fabelo, José Cabrera, Aurelio Vega, Víctor Déniz
Departamento de Electrónica y Automática (DIEA)
Instituto Universitario de Microelectronica Aplicada (IUMA)
Universidad de Las Palmas de Gran Canaria (ULPGC), España
hfabelo@iuma.ulpgc.es, jose.cabrera@ulpgc.es, avega@iuma.ulpgc.es, vdgonzalez@iuma.ulpgc.es

***Abstract*—Este artículo describe la arquitectura de una librería C desarrollada para realizar el control de un motor de corriente continua sin escobillas. La librería se ha elaborado siguiendo una metodología de programación modular. El sistema de control está basado en un microcontrolador ATmega64M1, integrado en una controladora para este tipo de motores de 3 fases. La controladora ha sido diseñada y fabricada, especialmente, para este proyecto. El objetivo principal de esta librería es el de ser utilizada como recurso educativo en la impartición de clases prácticas de programación de microcontroladores y sistemas de control de motores. Con ello, se inicia al alumno en los sistemas de control que actualmente están siendo más utilizados en cualquier tipo de aplicaciones, sustituyendo a los tradicionales motores de corriente continua con escobillas.**

Keywords— brushless direct current motor (BLDCM), BLDC motor control C library, microcontroller ATmega64M1, analog to digital converter (ADC), Power Stage Controller (PSC).

I. INTRODUCCIÓN

Los motores BLDC (*Brushless Direct Current Motors)* reemplazan la conmutación de las escobillas por una conmutación electrónica señalizada por sensores Hall de estado sólido. Estos sensores indican al microcontrolador (µC) de la controladora la posición en la que se encuentra el rotor del motor. La controladora, a partir de estas señales, debe provocar la excitación de las bobinas del estator del motor siguiendo una lógica de conmutación correcta. Es por esto, por lo que se hace necesario disponer de un software que realice estas operaciones. Además, dicho software debe tener en cuenta otros aspectos como el ajuste de velocidad del sistema, el sensor de temperatura del motor (en caso de poseerlo) o las comunicaciones hacia el exterior.

En la actualidad, la tendencia a la hora de realizar casi cualquier tipo de software es la de desarrollarlo de forma modular. Uno de los beneficios de esta metodología de desarrollo es obtener la posibilidad de reutilizar el código elaborado, consiguiéndose una gran productividad y reducción de tiempo en futuros trabajos en los que se puedan utilizar los módulos creados anteriormente.

La programación modular se basa en dividir un problema grande y complejo en varios subproblemas más pequeños y simples. Estos subproblemas, a su vez, deben ser divididos en otros más simples. Así se debe repetir estos pasos hasta obtener elementos lo suficientemente simples como para poder ser resueltos fácilmente. Esta técnica se denomina "refinamiento sucesivo o análisis descendente".

Con el proyecto descrito en este artículo, se pretende conseguir que los alumnos de electrónica y robótica comprendan y estudien la programación modular y la asocien con los sistemas de control de motores BLDC. Así, mediante la realización de prácticas con esta librería y su sistema de control completo, comprenderán el funcionamiento del mismo y se introducirán en la programación en C de los µCs.

II. LA LIBRERÍA C DE CONTROL DEL MOTOR

El µC elegido para el desarrollo de este proyecto es el ATmega64M1 [1], que posee una capacidad de memoria *flash* de 64 kB y una memoria interna SRAM (*Static Random Access Memory*) de 4 kB. Dentro de esta serie de ATmega, también existen el Atmega16M1 y el ATmega32M1 con una capacidad de memoria *flash* y SRAM de 16 kB y 1 kB, y 32 kB y 2 kB respectivamente. La elección del modelo de µC se realiza en función de la aplicación y del tamaño del programa. La razón de escoger el ATmega64M1 para este proyecto es la de poseer mayor capacidad de memoria *flash* donde alojar futuras ampliaciones del software.

El desarrollo de la librería de funciones para el control de motores BLDC se ha basado en la técnica de control en lazo abierto con ajuste de velocidad [2][3][4][5]. En el diagrama de bloques de la Fig. 1 se muestran los elementos más importantes utilizados del µC ATmega64M1.

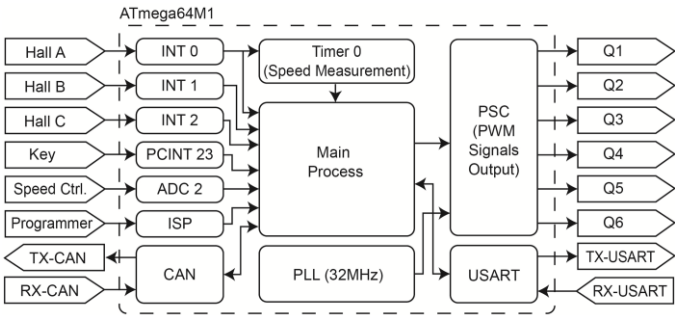

Fig. 1. Diagrama de las partes utilizadas en el diseño de la controladora BLDCM

Las señales de los sensores Hall (A, B y C), provenientes del motor, se introducen en el µC a través de los puertos que poseen las interrupciones externas prioritarias. Estas interrupciones actúan sobre el proceso principal del sistema activando la conmutación de las salidas Q1 a Q6 del módulo PSC (*Power Stage Controller*) en el orden correcto. La interrupción del sensor Hall A se encarga de realizar la medición de velocidad del motor mediante el temporizador 0.

La señal de entrada "*Key*" se encarga de arrancar o parar el motor. Esta señal, activa a nivel bajo, se introduce a través de la interrupción por cambio de pin número 23. Esta interrupción es ejecutada cuando se produce un cambio de nivel en la señal. Dentro del proceso principal, es la encargada de desactivar o activar el módulo PSC. Este módulo se encarga de generar las 6 señales PWM (*Pulse Width Modulation*), Q1 a Q6, que atacan los drivers de los transistores MOSFETs. Estos transistores son los responsables de excitar las bobinas del motor BLDC.

El ajuste de velocidad del motor se realiza mediante una señal analógica digitalizada a través de uno de los canales del ADC (*Analog to Digital Converter*). Esta señal digitalizada es la encargada de establecer el ancho de pulso (*Duty Cycle*) de la señal PWM que utiliza el módulo PSC. Esta señal PWM de 16 kHz se genera a partir de la señal de reloj del PLL (*Phase-Locked Loop*) interno del µC, establecida a 32 MHz.

Por otra parte, se dispone de los módulos de comunicaciones CAN (*Controller Area Network*) y UART (*Universal Asynchronous serial Receiver and Transmitter*) con sus respectivas señales de transmisión y recepción de datos.

A. Proceso Principal

En este proceso se realiza la inicialización de las variables y del hardware necesario del microcontrolador. Mediante la función *micro_modules_initialization()*, se realiza la configuración inicial de los puertos de entrada y salida del µC y se ejecutan las siguientes funciones:

- *adc_initialization()*: Configura y habilita el módulo ADC para la lectura del potenciómetro analógico de ajuste de velocidad por el canal ADC2.
- *timer1_initialization()*: Inicializa el temporizador 1 para realizar la tarea de muestreo de la velocidad y ajuste del ancho de pulso de la salida PWM. Se habilita la interrupción por comparación para que se ejecute cada 256 µs.
- *timer0_initialization()*: Configura el temporizador 0 como contador para realizar la medición de velocidad del motor. Habilita la interrupción por desbordamiento para convertir el temporizador 0 de 8 bits en uno de 16 bits. Esto se consigue gracias a una variable auxiliar que se incrementa cada vez que se ejecute esta interrupción. Con esta conversión se consigue una mayor precisión en la medición de la velocidad del motor.
- *uart_initialization()*: Configura el módulo UART del µC como transmisor y receptor. Para ello, utiliza los parámetros especificados por el usuario en el fichero de configuración.
- *start_pll_32mhz()* y *wait_pll_ready()*: Corresponde a las funciones encargadas de configurar y activar el PLL a una frecuencia de 32 MHz. Una vez activado, se mantiene a la espera de que el PLL esté preparado y se continúe ejecutando el programa.
- *psc_initialization()*: Realiza la configuración del módulo PSC según los parámetros detallados por el usuario en el fichero de configuración.
- *external_interrupt_initialization()*: Habilita las interrupciones externas utilizadas por los sensores Hall (INT1, 2 y 3). Además, habilita las interrupciones ejecutadas por cambio de pin para el interruptor de arranque del motor y para el interruptor de cambio de sentido de giro (en el caso de estar habilitada esta opción).

Una vez inicializado el sistema, se ejecuta un bucle infinito donde, cada 256 µs, se ejecuta la función *launch_adc()*. Esta función es la encargada de realizar la lectura del potenciómetro externo de ajuste de velocidad. Este valor es almacenado y utilizado para establecer la velocidad de referencia del motor. Mediante esta velocidad de referencia, se ejecuta el bucle de regulación en lazo abierto para establecer el valor del ancho de pulso de la señal PWM.

Obtenido el valor del ancho de pulso, se actualizan los registros del módulo PSC con ese valor. Posteriormente, se envía el valor de la velocidad medida a través del módulo UART para su visualización en el software de monitorización (Fig. 2).

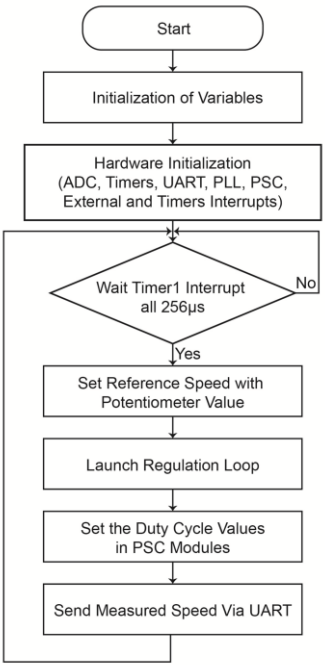

Fig. 2. Diagrama de flujo del proceso principal

B. Interrupciones de los Sensores Hall

Estas interrupciones son las encargadas de detectar los flancos de subida de las señales de los sensores Hall. Son las

interrupciones con más alta prioridad de todo el programa. En cada una, la función *output_psc_switch_commutations(value)* es ejecutada. Esta función requiere como parámetro el valor en ese instante de los sensores Hall, correctamente formateado. Mediante ese valor y el sentido de giro establecido en ese momento, se realiza la activación de las salidas Q correspondientes del módulo PSC. Estas señales de salida PWM tendrán el ancho de pulso establecido por el potenciómetro de ajuste de velocidad y serán las encargadas de realizar la conmutación de las bobinas del motor BLDC.

En la Fig. 3 se muestra la secuencia de conmutación de las salidas del módulo PSC en función de las señales de los sensores Hall cuando el sentido de giro es CW (*ClockWise*). Los tres estados posibles en los que se pueden encontrar las bobinas U, V y W del motor son los siguientes:

- VCC: bobina conectada a la tensión de alimentación.
- GND: bobina conectara a tierra.
- NC: bobina no conectada.

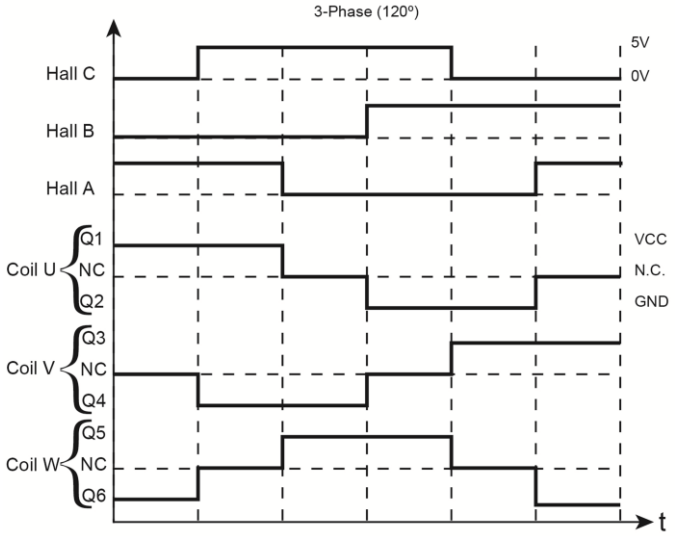

Fig. 3. Señales de los sensores Hall para la rotación CW

En la Tabla I se detalla la secuencia de conmutación de las salidas PSC en relación con el valor de los sensores Hall y el sentido de giro del motor, CCW (*CounterClockWise*) o CW.

TABLA I. SECUENCIA DE CONMUTACIÓN DE LAS SALIDAS PSC

Hall Sensors Value (C,B,A)	**PSC Outputs Active (CCW)**	**PSC Outputs Active (CW)**
001	Q5 – Q2	Q1 – Q6
101	Q3 – Q2	Q1 – Q4
100	Q3 – Q6	Q5 – Q4
110	Q1 – Q6	Q5 – Q2
010	Q1 – Q4	Q3 – Q2
011	Q5 – Q4	Q3 – Q6

En el caso particular de la interrupción del sensor Hall A (INT 0), por cada flanco de subida se realiza el cálculo de las revoluciones por minuto del motor mediante la función *calculate_estimated_speed()*. Esta función utiliza el valor de 16 bits obtenido del temporizador 0 y una constante, definida por el usuario y que varía en función del motor utilizado, para calcular las revoluciones por minuto del motor (Fig. 4).

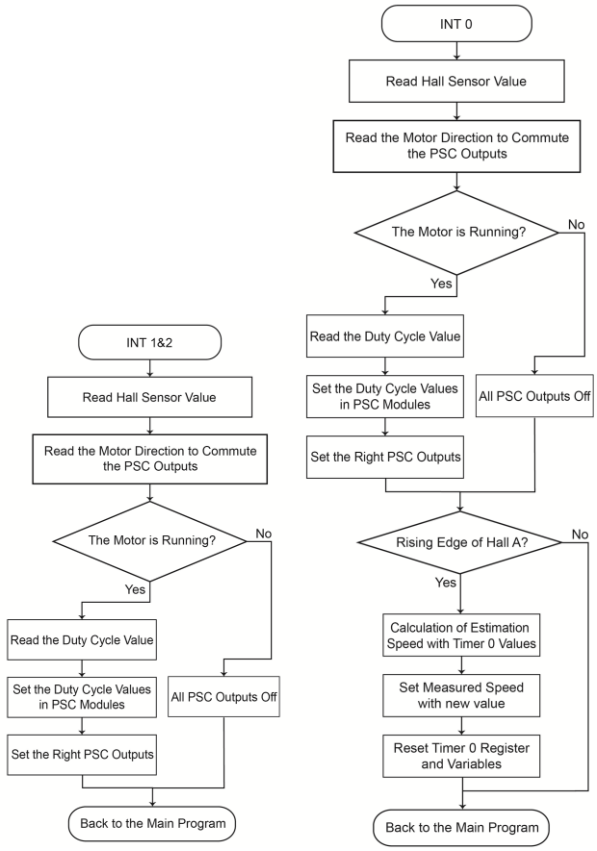

Fig. 4. Diagrama de flujo de las interrupciones de los sensores Hall

C. Interrupción Convertidor Analógico/Digital (ADC)

Es la interrupción encargada de realizar la lectura del potenciómetro analógico externo de ajuste de velocidad y convertir su valor a un valor digital (Fig. 5). Esta interrupción es ejecutada cuando se completa la conversión de la señal del potenciómetro conectada al canal 2 del ADC.

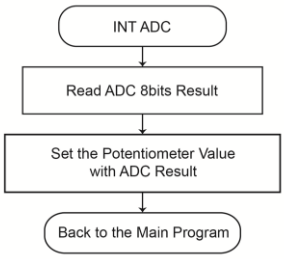

Fig. 5. Diagrama de flujo de la interrupción ADC

D. Interrupción Timer 0 por desbordamiento

Como ya se ha explicado anteriormente, se encarga de incrementar la variable auxiliar que convierte el temporizador 0 de 8 bits en uno de 16 bits (Fig. 6). En el caso de que se detecte que la variable auxiliar sea mayor de un cierto valor calculado, se resetea la variable y el valor de la velocidad de referencia. Si se detecta que la variable que indica si el motor está activo tiene un valor verdadero, se intenta reanudar la ejecución del motor. Para ello, se llama a la función *retry_run_motor()* que se encarga de lanzar el bucle de regulación, actualizar el valor del ancho de pulso y ejecutar la función de activación de las salidas del módulo PSC.

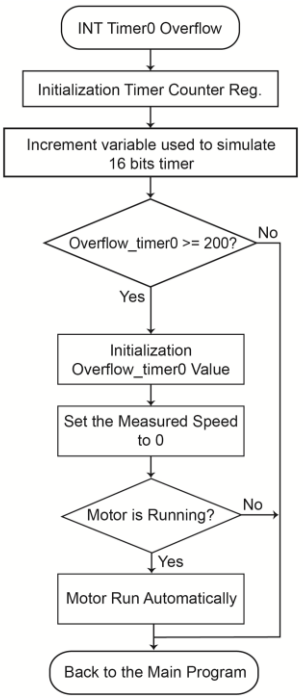

Fig. 6. Diagrama de flujo de la interrupción del *Timer* 0 por *Overflow*

E. Interrupción por cambio de pin 1 (PCINT 1)

Es la interrupción encargada de controlar el arranque o la parada del motor. Cuando la interrupción detecta un cambio en la señal del interruptor de arranque, comprueba si la señal se encuentra a nivel alto o bajo. En el caso de que el nivel sea alto, se realiza la desactivación de todas las salidas del módulo PSC. Cuando la señal se encuentra a nivel bajo, se realiza una llamada a la función *output_psc_switch_commutations(value)* con el valor de los sensores Hall en ese instante (Fig. 7).

F. Fichero de configuración

En este fichero se encuentran las constantes que el usuario debe configurar en función de sus necesidades. Para el módulo UART, ha de configurar el *baudrate*, el número de bits de stop y el número de bits a transmitir que utilizará en la conexión serie con el PC.

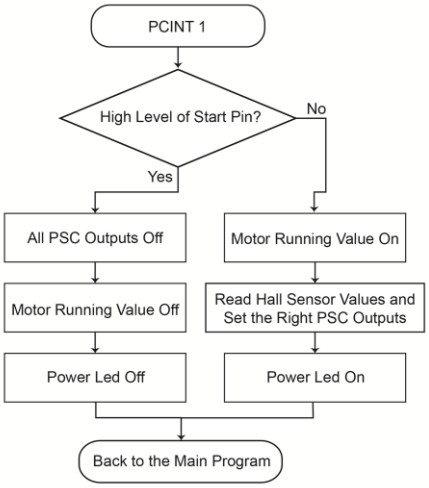

Fig. 7. Diagrama de flujo de la interrupción arranque y parada del motor

Otro aspecto que se debe configurar es la constante de velocidad del motor. Esta constante se calcula siguiendo la ecuación (1). Dependerá del tiempo, en segundos, configurado para el temporizador 0 (*t_timer0*), la velocidad máxima, en revoluciones por minuto, (*max_speed*) y el número de pares de polos del motor BLDC utilizado.

$$speed\_const = \frac{60 \cdot 255}{n \cdot t\_timer0_{(s)} \cdot \max\_speed_{(rpm)}} \quad (1)$$

Esta constante se utiliza para convertir la velocidad del motor medida a un dato acotado en un rango de 0 a 255 (8 bits). Este dato será el valor enviado al software de monitorización para visualizar la velocidad del motor.

G. Programación del ATmega64M1

En la Fig. 8 se muestra el mapa de memoria *flash* del µC ATmega64M1 utilizado en el proyecto. El código de la aplicación ocupa un espacio de 10,8 kB. Se observa que existe un espacio libre en la memoria para las futuras ampliaciones del proyecto.

La programación del microcontrolador se realiza mediante el software Atmel Studio 6.0 [6]. Este software es de descarga gratuita desde la web del fabricante del µC y permite el desarrollo y compilación de códigos elaborados en C/C++ o lenguaje ensamblador.

Para la realización de las pruebas, dentro de este proyecto, se ha desarrollado una controladora de motores BLDC de bajo coste [7]. Gracias a esta controladora (Fig. 9) se ha realizado el testeo de la aplicación. A través de la interfaz ISP (*In-System Programming*) del µC y el módulo programador AVRISP mkII [8], se realiza la carga del programa en memoria.

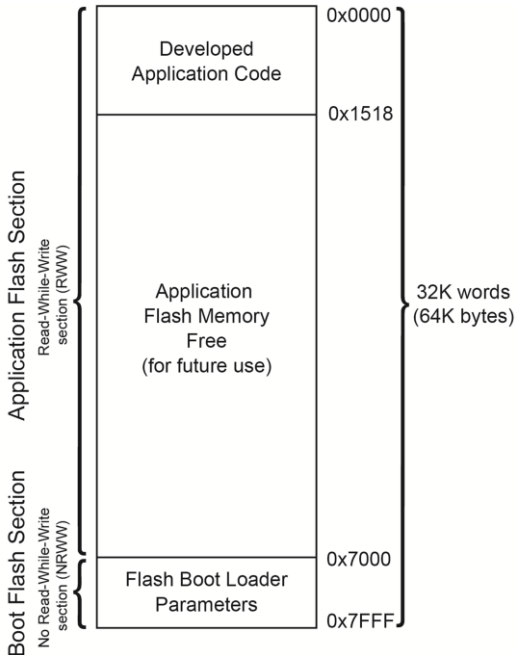

Fig. 8. Mapa de memoria *flash* del µC ATmega64M1 utilizado

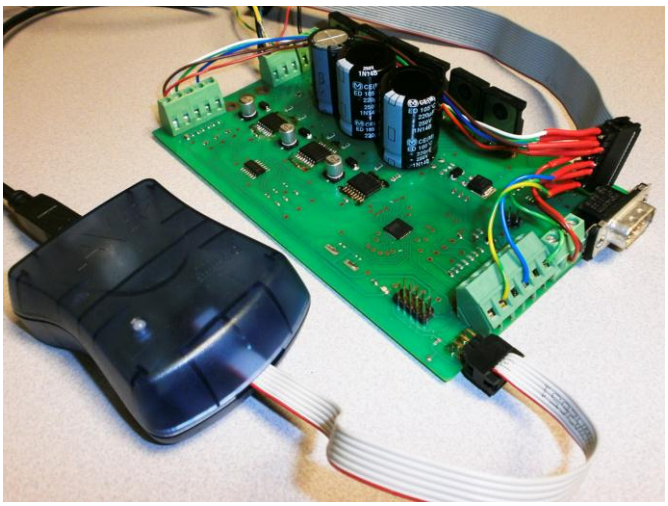

Fig. 9. Vista de la controladora BLDC y el programador AVRISP mkII

III. SOFTWARE DE CONTROL Y MONTORIZACIÓN EN LABVIEW

En este punto, se describe el software elaborado, utilizando NI LabVIEW [9], para el control y monitorización del motor BLDC desde un PC. El protocolo de comunicaciones empleado es RS-232. El software se compone de dos partes: un panel de control y monitorización del motor BLDC, y un panel para la configuración de la conexión UART y la constante de cálculo de velocidad en revoluciones por minuto.

A. Control y monitorización

En este panel se encuentran los controles y el visualizador de las revoluciones por minuto del motor (Fig. 10). Dispone de los controles de encendido y apagado (*ON/OFF*), cambio de sentido de giro (*CCW/CW*) y un potenciómetro virtual para el ajuste de velocidad. Estos controles virtuales se han incorporado para dar la posibilidad de controlar el motor desde la aplicación. En el panel de configuración se dispone de un interruptor que habilita estos controles. Si este interruptor está activado, los controles físicos del banco de pruebas quedarán anulados.

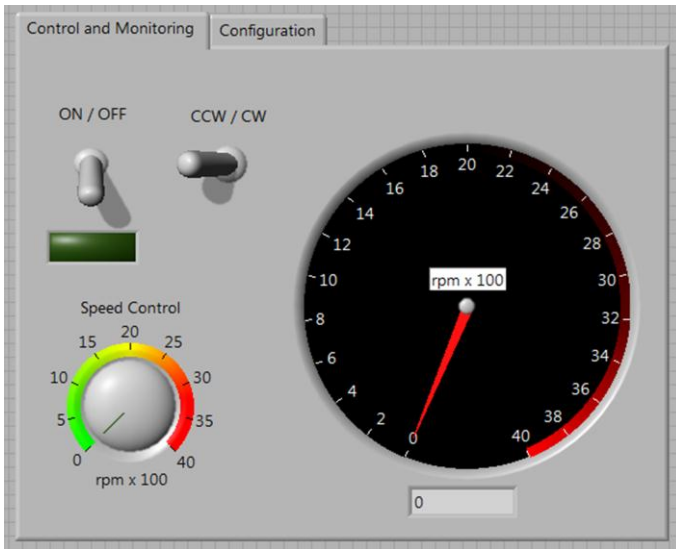

Fig. 10. Vista del panel de control y monitorización del motor BLDC

B. Configuración UART

En este panel se muestran los controles para configurar la conexión del puerto serie del PC con el µC (Fig. 11). Los parámetros a configurar de este protocolo son los siguientes:

- Selección del puerto serie usado en el PC.
- Selección del *baudrate* utilizado en la conexión.
- Número de bits por paquete en la conexión.
- Tipo de paridad establecida.
- Número de bits de stop utilizados.
- Control de flujo empleado en la conexión.
- Carácter de inicio de transmisión (*XON*).
- Carácter de parada de transmisión (*XOFF*).
- Carácter de fin de trama.
- Tiempo de espera máximo en milisegundos (*Timeout*).

Además, se dispone de un campo donde ha de introducirse la constante *alpha* utilizada para hallar el valor real de la velocidad del motor. Esta constante se calcula siguiendo la expresión (2), donde *n* es el número de pares de polos del motor BLDC, *speed_const* la constante calculada anteriormente y *t_timer0* el valor del temporizador 0 en segundos.

$$alpha = \frac{60}{n \cdot speed\_const \cdot t\_timer0_{(s)}} \qquad (2)$$

La velocidad real del motor BLDC se calcula mediante la expresión (3), donde *alpha* es la constante hallada en la ecuación (2) y *measured_speed* es el valor de 8 bits de la velocidad medida y recibida por el puerto serie.

$$real\_speed_{(rpm)} = alpha \cdot measured\_speed \quad (3)$$

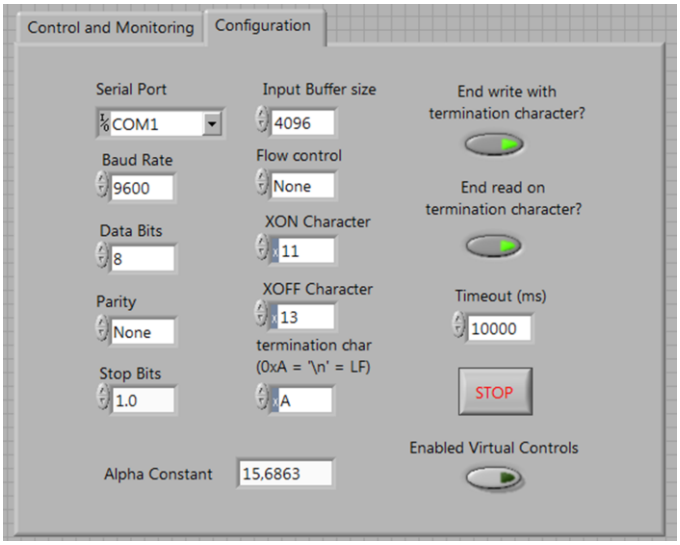

Fig. 11. Vista del panel de configuración

Conclusiones y Trabajo Futuro

Hoy en día, este tipo de motores BLDC son cada vez más utilizados gracias a su alto rendimiento y a sus excelentes características, sustituyendo a los motores DC (*Brushed Direct Current Motors*) tradicionales. Por ello, vemos una necesidad la adquisición de conocimientos de los alumnos en este campo. Con este sistema, se unifica el aprendizaje de los cálculos matemáticos del control del motor BLDC con los cálculos y configuraciones a realizar para programar el µC. De este modo, se pretende conseguir una mayor motivación y aprovechamiento de las prácticas por parte de los alumnos.

Una de las vías futuras de ampliación del proyecto es la de implementar las funciones para el control del motor en modo *Current-Loop* y *Speed-Loop*. Esto permitirá que el alumno aprenda todo lo necesario sobre el control del PID (*Proportional Integral Derivative controller*) de un motor BLDC.

Por otro lado, se está proyectando utilizar las funciones básicas del protocolo CAN como sistema de comunicaciones para el control y monitorización del motor. Este protocolo es más robusto y fiable que el protocolo RS-232. Implementando las funciones necesarias para la configuración del módulo CAN del microcontrolador, el alumno podrá realizar prácticas y entender el funcionamiento de este bus de comunicaciones. Posteriormente, se añadiría al proyecto la capa de aplicación utilizando CANOpen [10].

En el momento de presentación de este trabajo se están fabricando varias controladoras para su utilización en prácticas del próximo curso académico 2014/2015.

Referencias

[1] Atmel ATmega16/32/64M1 microcontrollers family: http://www.atmel.com/devices/ATMEGA64M1.aspx Last accesed 2013, December.

[2] Wang Dongmei, Guo Haiyan, and Yu Jing, "Modeling and Simulation Research of Brushless DC Motor open-loop Speed-adjustment System", The 2nd International Conference on Intelligent Control and Information Processing, ISBN 978-1-4577-0816-9, pp: 394 – 398, 2011.

[3] Alphonsa Roslin Paul, and Prof. Mary George, "Brushless DC motor control using digital PWM techniques", Proceedings of 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN 2011), ISBN 978-1-61284-653-8, pp: 733 – 738, 2011.

[4] Hai-tao Wang, Ze Zhang, and Xiang-yu Liu, "Design of control system for brushless DC motor based on TMS320F28335", Third International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), ISBN 978-1-4244-9010-3, pp 954 – 958, 2011.

[5] Radu Duma, Petru Dobra, Mirela Dobra, and Ioan Valentin Sita, "Low cost embedded solution for BLDC motor control", 15th International Conference on System Theory, Control, and Computing (ICSTCC), ISBN 978-1-4577-1173-2, pp 1 – 6, 2011.

[6] Atmel Studio 6.0 Manual. [Online]. Available: http://atmel.no/webdoc/atmelstudio/

[7] H. A. Fabelo, J. M. Cabrera, A. Vega, and V. Déniz, "Sistema de control de motores BLDC de bajo coste aplicado a docencia", XI TAEE, 2014.

[8] Atmel AVRISP mkII microcontroller programmer: http://www.atmel.com/tools/avrispmkii.aspx, Last accesed 2013, December.

[9] NI LabVIEW: http://www.ni.com/labview/esa/, Last accesed 2013, December.

[10] CANOpen protocol: http://www.can-cia.org/, Last accesed 2013, December.