Prácticas de Electrónica del Automóvil basadas en 'mbed' LPC1768

Carlos J. García-Orellana, Horacio González-Velasco, Ramón Gallardo-Caballero,
Miguel Macías-Macías y Antonio García-Manso
Dpto. de Ing. Eléctrica, Electrónica y Automática
Universidad de Extremadura
Badajoz - SPAIN
Email: cjgarcia@unex.es

***Resumen*—Presentamos un conjunto de prácticas diseñadas para una asignatura general de Electrónica del Automóvil utilizando el entorno 'mbed' y el microcontrolador LPC1768. Las prácticas tienen como objetivo abordar la interacción de un sistema basado en microcontrolador, y un sistema operativo en tiempo real, con sensores y actuadores, la introducción a los buses utilizados en vehículos y a los sistemas de diagnóstico OBD-II.**

I. Introducción

La incorporación de Sistemas Electrónicos a bordo de vehículos ha sufrido un crecimiento prácticamente exponencial en los últimos años [1], [2]. La Electrónica juega un papel muy relevante en los sistemas de inyección y ABS desde los años 80 del siglo pasado, pero sin lugar a dudas, ha sido desde comienzos del siglo XXI cuando el crecimiento ha sido más importante, principalmente de la mano de la interconexión de los sistemas a bordo del vehículo mediante buses de comunicación. Este hecho ha venido acompañado de una mayor relevancia del software y los sistemas de tiempo real, lo cual se hace patente en estándares como AUTOSAR [3], que permite diseñar el software a nivel de componentes, con una mayor abstracción del hardware y con una mejor interoperabilidad entre sistemas de distintos fabricantes.
En ese sentido se presenta este trabajo, que aborda una forma de acometer una serie de prácticas para una asignatura de 'Electrónica de Vehículos', con un presupuesto limitado y considerando los problemas asociados a este complejo mundo, sometido en muchas ocasiones al secreto industrial.

I-A. Contexto de la asignatura

La asignatura "Electrónica de Vehículos" es una optativa (6 créditos ECTS) del último curso (cuarto) del "Grado en Ingeniería Electrónica y Automática (Rama Industrial)" con competencia en Ingeniería Técnica Industrial. Al llegar a este curso los alumnos han recibido formación en Electrónica Analógica y Digital, Automática, Programación en C y Microcontroladores, por tanto, se encuentran en disposición de abordar montajes electrónicos y tareas de programación de microcontroladores. Sin embargo, su experiencia es limitada y en particular nunca han utilizado sistemas operativos en tiempo real, ni C++, para programar microcontroladores.

I-B. El sistema 'mbed'

Cuando hablamos de microcontroladores, en un primer momento, tenemos tendencia a pensar en el hardware únicamente. Sin embargo, al valorar y exponer las características de un microcontrolador debemos pensar tanto en el hardware como en el software (y el entorno de desarrollo), es decir, en el sistema completo. En este sentido 'mbed' [4] (desarrollado por ARM Ltd.) nos ofrece un sistema de desarrollo de microcontroladores con las siguientes características principales:

- Arquitectura ARM Cortex-Mx.
- Entorno de desarrollo basado en la nube (web).
- Fácil programación del microcontrolador.
- Orientado a *Internet de las Cosas*.

I-B1. Hardware - LPC1768: En los últimos años, 'mbed' ha aumentado de forma significativa las plataformas hardware disponibles [5]. Una de las primeras plataformas disponibles es el 'mbed' LPC1768, que es el que hemos utilizado en para el desarrollo de las prácticas descritas en el presente trabajo. En la figura 1 podemos ver el formato y el conexionado disponible en el 'mbed' LPC1768.
El LPC1768 es un microcontrolador fabricado por NXP, con las siguientes características fundamentales:

- Procesador de 32 bits ARM Cortex-M3.
- Frecuencia de reloj de 96 MHz, 32 KB de RAM y 512 KB de Flash para programas.
- Interfaces: Ethernet, SPI, I2C, UART, CAN, PWM, GPIO, USB (host y device).
- 6 canales ADC y un DAC.
- Funcionamiento a 3.3 V, pero las E/S son tolerantes a 5 V.

Como podemos ver se trata de un microcontrolador bastante potente y con una gran conectividad. En la figura 1, podemos observar como se distribuye esta conectividad, además, debemos considerar que todos los terminales marcados en azul son GPIO estándar.
'Mbed' incluye este microcontrolador en una pequeña placa (de 54x26 mm.) con formato DIP de 40 terminales, lo cual, la hace muy adecuada para su uso en placas de prototipo. La alimentación se obtiene a través del puerto USB o directamente en el terminal VIN (entre 4.5 y 9 V).

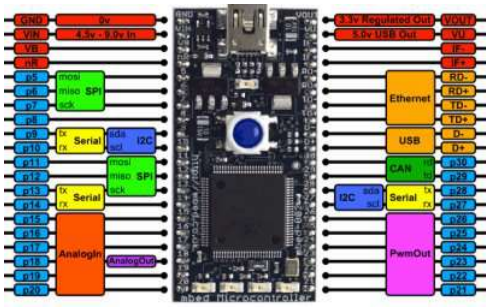

Figura 1. Distribución de las conexiones disponibles en el 'mbed' LPC1768.

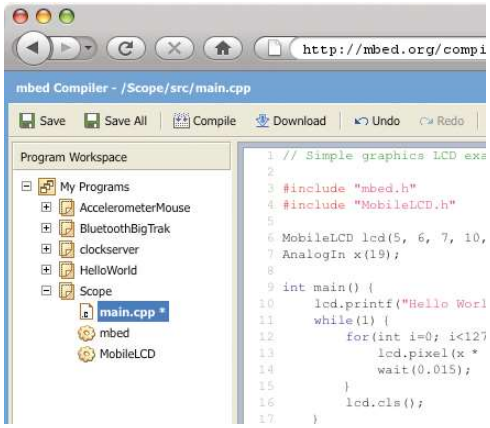

Figura 2. Detalle del entorno de desarrollo en la nube del sistema 'mbed'.

Otro aspecto muy importante del sistema 'mbed' es su forma de programación. Muchos de los dispositivos hardware de la familia 'mbed' permiten la programación *'Drag & Drop'*, es decir, al conectarlo al ordenador por USB nos muestran una unidad Flash en la que podemos copiar el binario de nuestro programa. Posteriormente, hacemos un 'reset' al 'mbed' y un 'chip de interfaz' incluido en la placa (otro microcontrolador) se encarga de programar el microcontrolador principal (en nuestro caso el LPC1768). Este 'chip de interfaz' además nos crea un puerto serie virtual (para mostrar mensajes) y un dispositivo CMSIS-DAP para poder utilizar un depurador externo.

I-B2. Software: Como hemos comentado anteriormente, el sistema 'mbed', además del microcontrolador, incluye un entorno de desarrollo en C++ basado en la nube. Es decir, a través de un navegador web nos conectamos a la dirección https://developer.mbed.org/compiler/, y previa autentificación, podemos utilizar el entorno on-line. Las credenciales para acceder se pueden obtener de forma gratuita.
En la figura 2 podemos observar el aspecto básico del entorno de desarrollo. Nos permite crear proyectos, incluir librerías, editar el código y compilarlo. Una vez compilado, se nos descargará el archivo a grabar en el microcontrolador.

Sin lugar a dudas el gran hándicap de este entorno de desarrollo es la imposibilidad de depurar el programa paso a paso. Esto es posible con entornos de desarrollo externos

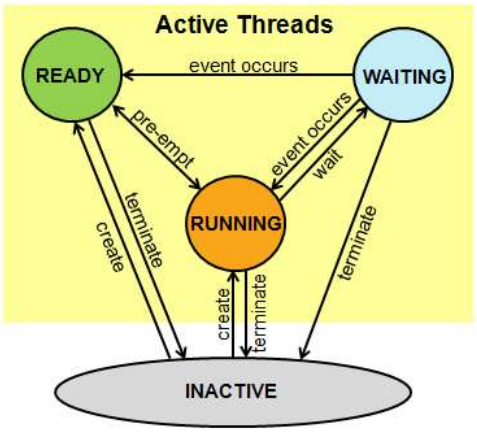

Figura 3. Posibles estados de laa tareas y sus transiciones en el RTOS de 'mbed'.

a través del interfaz CMSIS-DAP. Actualmente el sistema 'mbed' está inmerso en una evolución que lo dotará de nuevas herramientas.
Posiblemente uno de los puntos fuertes del sistema 'mbed' es la librería (en C++) para acceder a los recursos hardware del microcontrolador. La librería, conocida como 'mbed Official', está liberada como software libre y dispone de clases para acceder y utilizar E/S analógicas y digitales, interrupciones, salidas PWM, temporizadores, interfaces (serial, I2C, SPI, **CAN**), Sistema de ficheros local, USB, Ethernet, etc ... Es decir, tenemos a nuestra disposición una librería de clases muy completa y con una buena documentación.
Además, podemos utilizar un sistema operativo de tiempo real ('mbed RTOS') en forma, también, de código abierto. Este sistema en tiempo real tiene un interés en el trabajo que nos ocupa, ya que nos permite introducir a los alumnos en este campo de la programación en tiempo real y paralela, tratando muchos de los conceptos que se utilizan en los sistemas de tiempo real con los que se desarrolla el software de los automóviles actuales.

El 'mbed RTOS' está basado en el estándar *CMSIS-RTOS* y está desarrollado como un API de C++. Incluye, entre otros, los siguientes elementos básicos (vía clases de C++) para la programación con un RTOS:

- *Thread*: Es decir, tareas o hilos de ejecución. Estos 'hilos' pueden tener diferentes prioridades (7 en total). Es el ingrediente fundamental de cualquier RTOS. En la figura 3 podemos observar los distintos estados que puede tener un 'hilo' y las transiciones entre dichos estados.
- *Mutex y semaforos*: Los 'mutex' y los 'semaforos' permiten gestionar el acceso a recursos compartidos, como por ejemplo, canales de comunicación, zonas de memoria compartidas, etc ...
- *Queue, MemoryPool y Mail*: Estas clases permiten definir paso de datos entre 'hilos productores' e 'hilos consumidores'. *MemoryPool* permite definir y manejar zonas estáticas de memoria. Para el paso de mensajes entre

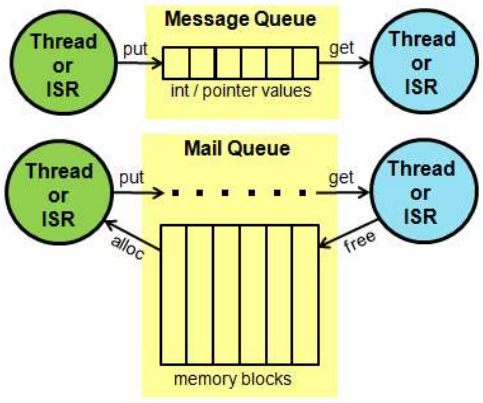

Figura 4. Funcionamiento de las *Queue* y los *Mail* del RTOS de 'mbed'.

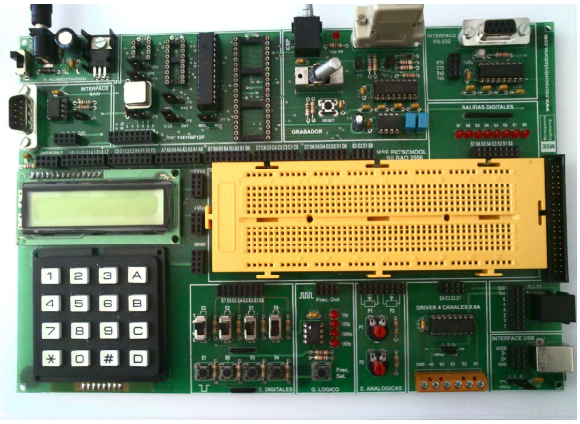

Figura 5. Entrenador PIC School utilizado como base de trabajo en las prácticas.

tareas podemos utilizar *Queue* o *Mail*. La diferencia entre ambos es que el primero permite enviar punteros, mientras que el segundo mensajes completos (ver figura 4).

- *RTOSTimer*: Esta clase implementa temporizadores dentro del RTOS, para que funciones puedan ser llamadas de forma periódica. Estos temporizadores no se ejecutan mediante interrupciones, sino dentro de un 'thread' de más prioridad que el resto.

A continuación expondremos el trabajo desarrollado y una descripción de las prácticas implementadas y finalizaremos con las conclusiones de la experiencia de estos años.

II. Descripción del trabajo

Para el desarrollo de la asignatura objeto del presente trabajo hemos creado cuatro prácticas de laboratorio basadas en el uso del microntrolador 'mbed' LPC1768. Estas prácticas han sido diseñadas para cubrir tres de las materias fundamentales de la asignatura:

- Sensores y actuadores.
- Buses de comunicación.
- Sistemas de diagnóstico.

En la mayoría de ellas hemos utilizado el sistema operativo en tiempo real disponible en el entorno 'mbed'. Las prácticas

Figura 6. Componentes principales utilizados en la Práctica 1 (módulo de ultrasonidos HC-SR04 y 'buzzer' activo.

se han desarrollado en sesiones de 2 y 3 horas, ocupando un total de 16 horas de laboratorio. Los alumnos están obligados a preparar un 'prelab' de forma previa a la sesión de prácticas. En la evaluación de cada práctica tomamos en consideración dicho 'prelab', el desarrollo de la práctica en el laboratorio y una pequeña memoria del trabajo realizado y de los resultados obtenidos.

La plataforma básica utilizada para el desarrollo de las prácticas es el entrenador conocido como *'PIC School'*, que podemos observar en la figura 5. Aunque estrictamente hablando este entrenador no es necesario (podríamos utilizar solamente una 'protoboard' normal), si es útil porque incorpora una serie de componentes ya conectados. En nuestro caso, serán de utilidad la pantalla LCD de 16x2, el 'transceiver' CAN y el conjunto de pulsadores.

A continuación, describiremos el grueso de las prácticas planteadas y los aspectos que se trabajan en ellas.

II-A. Práctica 1. Introducción al entorno 'mbed'

La primera práctica pretende, por un lado, iniciar a los alumnos en el entorno 'mbed' y en los aspectos básicos del RTOS, y por otro lado, trabajar con sensores y actuadores.

Para ello, se propone a los alumnos la realización de un indicador de distancia de aparcamiento, utilizando para ello un sensor de distancia por ultrasonidos y un 'buzzer' activo. El módulo de ultrasonidos utilizado es el HC-SR04 (muy conocido y de bajo coste), que podemos observarlo en la figura 6 junto al 'buzzer' activo.

En el 'prelab' de esta primera práctica se le pide a los alumnos que preparen el programa, utilizando el driver ya existente para el módulo de ultrasonidos, para que la cadencia del pitido varíe en función de la distancia y que para ello utilicen PWM. Además, se les pide que muestren la distancia medida por la pantalla LCD del PIC School.

En la práctica, los alumnos deben desarrollar el mismo programa pero utilizado el RTOS, para ello, se les pide que realicen tres 'threads' (tareas): una para iniciar la medida del sensor, otra para generar el pitido y otra para mostrar la distancia por el LCD. El objetivo es que con estas sencillas tareas comprendan los conceptos de 'thread' y de comunicación entre 'thread'. En concreto, se insta a los alumnos a realizar la comunicación entre tareas utilizando una estructura de datos compartida en la memoria y protegiendo el acceso a dicho

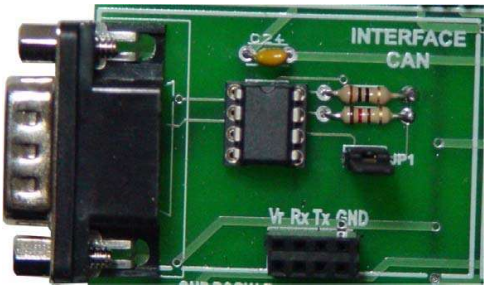

Figura 7. Módulo con el MCP2551 incluido en el 'PIC School'.

recurso utilizando un *'mutex'*. De está manera conseguimos introducir a los alumnos en el uso de un RTOS.

II-B. Práctica 2. Buses de comunicación

La segunda práctica aborda el uso de dos de los buses más utilizados en vehículos: LIN *(Local Interconnect Network)* [1], [6] y CAN *(Controller Area Network)* [1], [7]. El bus LIN es un bus basado en el protocolo RS232 (aunque no es RS232) y pensado para aplicaciones de baja velocidad y no críticas, manteniendo bajo el coste, utiliza tres hilos (incluida alimentación). Se puede implementar utilizando un puerto serie del microcontrolador y un 'transceiver'. Actualmente, está en fase de estandarización por la ISO (pero aún así es utilizado ampliamente en vehículos).
El bus CAN es un bus de comunicaciones ampliamente utilizado en los vehículos y en la industria en general. Es un bus multimaster, orientado a mensajes (no a direccionamiento), asíncrono y con una topología de bus diferencial. Es un bus robusto, con una buena gestión de errores y un arbitraje basado en prioridades. Desde el punto de vista de la implementación, en el bus CAN debemos considerar el controlador de bus y el 'transceiver'. En nuestro caso, partimos del hecho de que el LPC1768 incorpora, ya integrado, el controlador. Como 'transceiver', utilizaremos el integrado MCP2551 de Microchip, con la ventaja de que nuestra plataforma de trabajo, el PIC School, trae un módulo con este componente ya montado tal y como se muestra en la figura 7.

Antes de abordar esta segunda práctica los alumnos ya han recibido las nociones básicas de los buses LIN y CAN en las clases de teoría (principales características, aspectos físicos, esquema de comunicación, tramas de los mensajes, etc ...), con lo cual están en condiciones de afrontar esta práctica que se organiza en dos sesiones, pero con un objetivo global.
El objetivo global de la práctica es construir un *'gateway'* LIN-CAN, cuyo esquema podemos ver en la figura 8. En dicho 'gateway' el microcontrolador actuará de maestro del bus LIN, comunicándose con una tarjeta LIN (que dispone de una serie de pulsadores junto con un potenciómetro) y un LED LIN tri-color. Además, el microcontrolador gestionará el medidor de distancias de aparcamiento de la primera práctica (sustituyendo el 'buzzer' por un LED). Este 'gateway' enviará el estado (color e intensidad) del LED LIN y la distancia que mide el sensor de ultrasonidos por el bus CAN. Todos estos

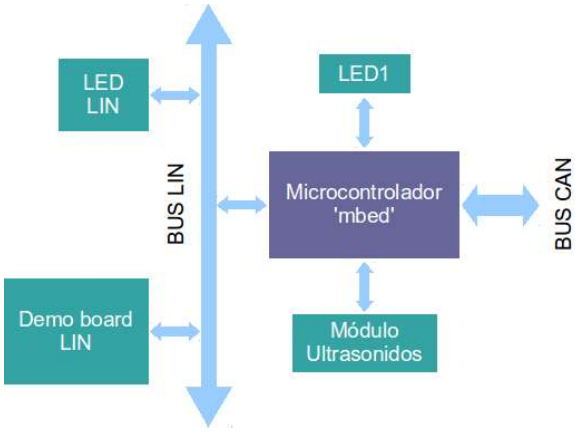

Figura 8. Diagrama de bloques del *'gateway'* LIN-CAN a realizar en la práctica 2.

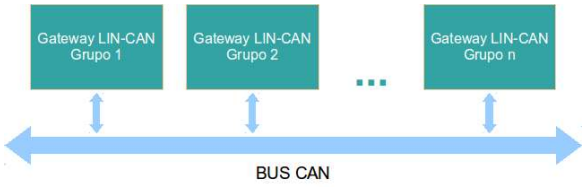

Figura 9. Esquema global de la conexión a través de bus CAN de los 'gateways' desarrollados por cada grupo de alumnos.

módulos compartirán el bus CAN existente en el laboratorio (ver figura 9).

La práctica se desarrolla en dos sesiones de tres horas cada una. En la primera sesión se aborda la parte correspondiente al bus LIN. Para acometer esta parte los alumnos cuentan con:

- *Librería LIN*: Es una librería, adaptada para esta asignatura de Arduino a 'mbed', que permite utilizar uno de los puertos serie del 'mbed' (ver figura 1) como interfaz LIN. Esto es necesario, ya que el comienzo del protocolo LIN es diferente al RS232, aunque luego la transmisión de los bytes que forman la trama LIN si sigue el protocolo RS232.
- *Transceiver LIN*: Es un dispositivo necesario para conectar el puerto serie del microcontrolador al bus LIN. En nuestro caso utilizamos el circuito integrado de Microchip MCP2004 [8].
- *Placa 'PICKit 28-Pin LIN Demo Board'*: Es una placa para desarrollar prototipos LIN del fabricante Microchip [9]. Incluye los pulsadores, un potenciómetro, un 'transceiver' LIN y un microcontrolador (de la famila PIC). Mediante mensajes LIN podemos encuestar el estado de los pulsadores y del potenciómetro. En el 'firmware' por defecto realiza funciones de esclavo LIN. Se muestra en la figura 10.
- *'Automotive Ambient Lighting Module Reference Design'*: Es un LED tricolor de Microchip, con interfaz esclavo LIN. Incluye un 'transceiver' LIN y un microcontrolador PIC. Con el 'firmware' que trae es posible

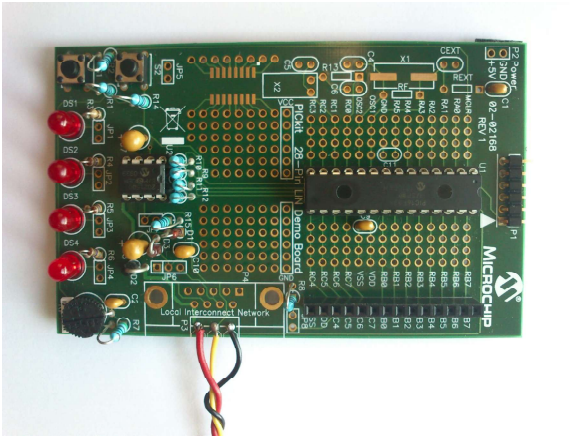

Figura 10. Placa de demostración LIN de Microchip utilizada en la Práctica 2.

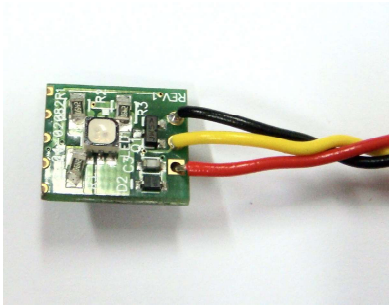

Figura 11. LED LIN de Microchip utilizado en la Práctica 2.

configurar el color y la intensidad mediante un mensaje LIN [10]. En la figura 11 lo tenemos representado.

Con estos componentes los alumnos deben desarrollar el software para que el 'mbed' realice las funciones de master en el bus LIN. La tarea LIN master realizará, de forma periódica, una encuesta a la placa 'Demo Board LIN' y actuará en consecuencia sobre el LED LIN. Uno de los pulsadores nos permitirá cambiar el color del LED (dentro de la secuencia predefinida, a elegir por los alumnos), mientras que el potenciómetro nos cambiará la intensidad del LED LIN.

La segunda sesión de la práctica consiste en publicar el estado de nuestros sensores ('Demo Board LIN' o estado de la selección del LED LIN y sensor de distancias) en el bus CAN y recoger el estado de los sensores del grupo elegido (que realizaremos mediante el segundo pulsador de la 'Demo Board LIN') para actuar sobre nuestro LED LIN y el LED del 'mbed', indicado en la figura 8. Para esta segunda parte, únicamente deberemos conectar a uno de los interfaces CAN del 'mbed' el 'transceiver' CAN del PIC School, mostrado en la figura 7.
Con el fin de que los alumnos tengan un nodo de referencia en el bus CAN, en todo momento se encuentra disponible el 'nodo del profesor' que podemos ver en la figura 12 y tiene todo el sistema implementado.

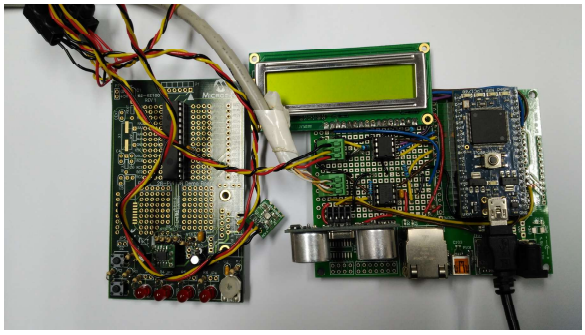

Figura 12. Placa con un nodo completo de la práctica para que los alumnos puedan verificar sus desarrollos CAN.

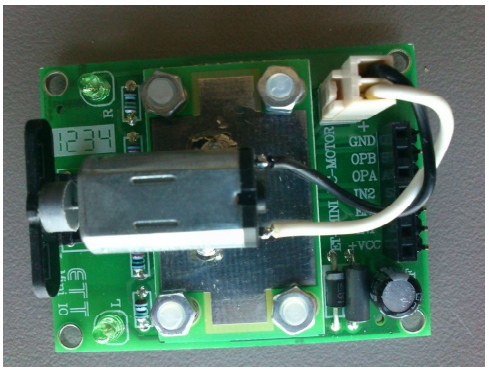

Figura 13. Placa con el motor de DC y sensor óptico de velocidad utilizado en la práctica 3.

II-C. Práctica 3. Control de un motor de DC

Los motores de DC son utilizados en multitud de aplicaciones, también en el sector de los vehículos. Sus aplicaciones incluyen bombas (de combustible, refrigerante), ventiladores en general y servomecanismos. En muchas ocasiones es necesario controlar la velocidad de rotación del motor de forma precisa. La tercera práctica planteada para esta asignatura aborda este problema.

La práctica propone a los alumnos el estudio del control de velocidad de un pequeño motor de DC. Para ello se utiliza la placa mostrada en la figura 13. Dicha placa, además del motor, incluye un sensor óptico de velocidad y el doble puente H (L293) que actúa como driver del motor.

Los alumnos deben realizar un 'driver' para el sensor óptico de velocidad, utilizando para ello interrupciones y temporizadores. Se les aconseja realizar una clase de C++ que encapsule todo el 'driver'. A continuación, se les pide que implementen un lazo de realimentación y comprueben la respuesta. El resultado es, como era de esperar, un error en la velocidad, no alcanzando el 'setpoint', por tanto, se les pide que para mejorar la respuesta utilicen un control PI o PID. Para ello, deben diseñar el controlador (con Matlab o Scilab, por ejemplo) e implementarlo en el software, tarea bastante sencilla ya que existe una clase de C++ que implementa un

PID en el 'mbed'.

II-D. Práctica 4. Lector de códigos de error OBD-II

En la Electrónica de Vehículos actual el diagnóstico juega una papel fundamental y los alumnos deben trabajar con él. Por ello, en la última práctica los alumnos deben realizar un lector de códigos de error OBD-II (*On Board Diagnostic*). La norma OBD-II [11] es la norma estandarizada para obtener información sobre el estado de los componentes electrónicos a bordo del vehículo relacionados con las emisiones. Es obligatorio para todos los vehículos a partir, aproximadamente, del año 2000. La información ofrecida por OBD-II no es tan amplia ni detallada como la que ofrecen los sistemas propios de cada fabricante, pero éstos son propietarios y en la mayoría de los casos, confidenciales.

En Europa, el estándar considera principalmente dos posibles protocolos físicos para conectar el lector de códigos al sistema de diagnóstico: KWP-2000 (línea K) o CAN. De hecho, a partir de 2008, muchos fabricantes utilizan CAN para OBD-II, según la norma ISO-15765 (*Road vehicles - Diagnostics on Controller Area Networks (CAN)*).

Uno de los aspectos más importantes de OBD-II lo constituyen los servicios disponibles, los cuales nos ofrecen la información accesible, estos servicios se encuentran descritos en la norma ISO-15031 (*Road vehicles - Communication between vehicle and external equipment for emissions-related diagnostics*). En dicha norma se definen 9 servicios (del $01 al $09). Podemos encontrar información sobre dichos servicios en [12], además de en el documento de la norma ISO-15031.

Los servicios que tienen interés para esta practica son:

- *Servicio $01 - Obtener parámetros actuales del sistema de propulsión*: Este servicio nos permite obtener información acerca de parámetros de tiempo real (PIDs), además de qué parámetros están soportados. La comunicación debe comenzar pidiendo el parámetro (PID) $00 del servicio $01, lo cual nos dará información de que otros PIDs están disponibles.
- *Servicio $03 - Obtener códigos de error relacionados con las emisiones (DTC)*: Los DTC (*'Diagnostic Trouble Codes'*) son los códigos de error confirmados y almacenados en la Unidad de Control (ECU). Este servicio nos permite obtenerlos. Deberemos previamente realizar una petición al servicio $01, PID $02, para saber cuantos DTCs hay almacenados.
- *Servicio $04 - Borrar DTCs almacenados*: Este servicio permite borrar los códigos de error almacenados y apagar el indicador de avería (MIL).

El objetivo de la práctica consiste en realizar un lector de DTCs y, de forma opcional, obtener algunos parámetros de tiempo real. Es decir, desarrollar un software para el 'mbed' que, conectado a una ECU, trabaje con los tres servicios OBD-II comentados anteriormente.

Para realizar esta práctica, el hardware de la herramienta de diagnóstico no es un problema, ya que es fácil de construir utilizando nuestro 'mbed', el LCD, los pulsadores y el 'transceiver' CAN del PIC School. Sin embargo, para realizar está práctica de forma satisfactoria necesitamos una ECU por puesto de trabajo. En estos casos lo habitual es trabajar con emuladores de ECU, que permiten simular códigos de error y parámetros de tiempo real, no obstante, la necesidad de tener un simulador por puesto de trabajo nos hacía complicado asumir el gasto.

Figura 14. Placa de Microchip utilizado como interfaz CAN con el PC.

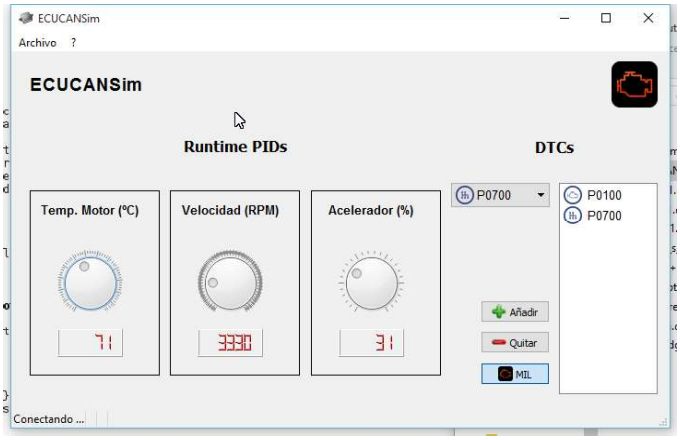

Figura 15. Panel de control de *ECUCANSim*, el simulador OBD-II desarrollado para la práctica 4.

Como solución, hemos utilizado un interfaz CAN para PC de Microchip (bastante económico) [13], que podemos ver en la figura 14. Con ello, hemos resuelto los problemas de conectividad entre el CAN bus y el PC.

Pero lógicamente, además, necesitamos el emulador en sí. Para ello, hemos desarrollado para la asignatura un programa que hace esas funciones de emulación y que hemos llamado *'ECUCANSim'*. Tal y como podemos ver en la figura 15, *ECUCANSim* nos ofrece tres PIDs del servicio $01, varios DTCs almacenados (que podemos seleccionar de la lista), la posibilidad de borrarlos mediante la herramienta de diagnóstico y activar la luz de avería (MIL). Además, el simulador funciona bajo Windows o Linux, proporcionándonos mayor flexibilidad.

III. Conclusiones y trabajo futuro

Después de varios años los resultados son bastante buenos, a pesar de las limitaciones de diseñar unas practicas para un

asignatura tan específica con un presupuesto muy limitado. Los alumnos adquieren destrezas básicas en el manejo de buses en vehículos, nociones básicas sobre interacción con sensores y actuadores utilizando un sistema operativo en tiempo real, así como una aproximación a los sistemas de diagnostico.
En un futuro queremos sustituir el entorno software de 'mbed' por una implementación del sistema OSEK/VDX [14]. En particular, queremos trabajar con la versión de ERIKA Enterprise [15], que ofrece una implementación de OSEK/VDX certificada y de libre uso, además, soporta plataformas como Arduino y otras basadas en ARM.

AGRADECIMIENTOS

El presente trabajo ha sido parcialmente financiado por la Junta de Extremadura y FEDER a través de la ayuda GR15101.

REFERENCIAS

[1] R. Bosch, *Automotive Electrics and Automotive Electronics*, 5ªEd.Springer Vieweg, 2014. ISBN: 978-3-658-01783-5.
[2] W.B. Ribbens, *Understanding Automotive Electronics*, 7ªEd. Butterworth Heinemann, 2013. ISBN: 978-0080970974.
[3] Página web oficial del consorcio *AUTOSAR*. http://www.autosar.org.
[4] Página web oficial del sistema *MBED*. https://developer.mbed.org/.
[5] Descripción del *MBED LPC1768*. https://developer.mbed.org/platforms/mbed-LPC1768/.
[6] Descripción en Wikipedia del *BUS LIN*. https://en.wikipedia.org/wiki/Local_Interconnect_Network.
[7] Descripción en Wikipedia del *BUS CAN*. https://en.wikipedia.org/wiki/CAN_bus.
[8] Descripción y hoja de características del *MCP2004* de Microchip. http://www.microchip.com/MCP2004.
[9] Información de la placa *'PICkit 28-Pin LIN Demo Board'* de Microchip. http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=DM164130-3.
[10] Información del *'Automotive Ambient Lighting Module Reference Design'* de Microchip. http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=APGRD004.
[11] Descripción en Wikipedia de *OBD-II*. https://en.wikipedia.org/wiki/On-board_diagnostics.
[12] Descripción en Wikipedia de los servicios *OBD-II*. https://en.wikipedia.org/wiki/OBD-II_PIDs.
[13] Información de la placa *'MCP2515 CAN Bus Monitor Demo Board'* de Microchip. http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=MCP2515DM-BM.
[14] Página del portal *OSEK/VDX*. http://www.osek-vdx.org.
[15] Página principal de *ERIKA Enterprise*. http://erika.tuxfamily.org/drupal/.