

# Proyecto de Electrónica Digital III

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES  
CÁTEDRA DE ELECTRÓNICA DIGITAL III



UNIVERSIDAD  
NACIONAL DE CORDOBA

Alumnos :

Justin Lafay, Agustín Pallardó, Franco Saporito

Control de Motor con puente H, comandos UART y ADC

Profesor :

Ing. Julio Sánchez  
Ing. Fernando Gallardo

Noviembre de 2023

# Objetivo

El objetivo de nuestro proyecto es desarrollar un controlador para un motor eléctrico, utilizando un puente H. En nuestro caso, emplearemos el microcontrolador LPC1769. Contará con las siguientes funciones:

- Control de la tensión de entrada del motor mediante **2 PWM** (uno para cada sentido).
- Control de la dirección de rotación del motor mediante un **puente H**.
- Modificación del ciclo de trabajo del PWM (para cambiar la velocidad) de dos maneras diferentes:
  - A través del **ADC**, con una entrada de voltaje continuo. En este caso, la dirección de rotación se hace por interrupción.
  - Mediante la **conexión UART**, con el envío de valores de velocidad y dirección de rotación.
- Almacenamiento de los valores de ADC en modo burst en un registro de memoria mediante **DMA**.

## Selección de pines

### Control del motor

Para darle el sentido de giro y la velocidad al motor, se han seleccionado los pines 4 y 5 del puerto 2 para controlar la parte inferior del puente H como llave, y los pines 2 y 3 del puerto 2 para controlar la parte superior del puente H, suministrando con señales PWM para darle la velocidad correspondiente. La selección fue por cercanía.

### Interrupciones

Se han utilizado 3 de los 4 pines dedicados para interrupciones externas, por ende, se han empleado los pines 10, 11 y 12 del puerto 2.

### Luces por GPIO

Para controlar las luces, se eligieron los pines 6, 7 y 8 del puerto 2, siendo estos elegidos por cercanía a los otros pines más utilizados, ya que la mayor parte de la placa puede emplear esta función.

### UART

Se ha empleado el módulo UART0 para la comunicación, siendo que la placa tiene más módulos UART disponibles, los pines utilizados son el 2 y el 3 del puerto 0, que permiten este modo de operación.

### ADC

Se utilizó solamente un canal del módulo ADC en nuestro programa y en configuración burst, por lo que hemos decidido utilizar el primero, correspondiente al pin 23 del puerto 0.

## Sensado de velocidad

Para lograr esta operación, hemos empleado la función Capture del TIMER0. Elegimos el pin 26 del puerto 1 que disponía de esta funcionalidad.

## Desarrollo

### Estructura general

El código se basa en una variable que centraliza toda la información necesaria para controlar el motor llamada "flags"; esta tiene 8 bits, y utilizamos los primeros 6 para controlar el motor :

Function	Error UART	Recepcion lista UART	Flag de motor frenado	ADC(0)/UART(1)	Parada de emergencia	Sentido giro motor
Bits nº	6	5	4	3	2	1

Cada elemento (ADC, PWM, Timer, UART,...) luego se inicializa y se utiliza en una función dedicada. El main, por su parte, tiene un bucle while(1) que hace funcionar el motor con su función correspondiente 'girar()'.

### Interrupciones

Se utilizan 3 interrupciones externas:

- INT0 para cambiar entre ADC y UART
- INT1 para cambiar la dirección de rotación del modo ADC
- INT2 para activar la parada de emergencia

### Timer y SysTick

El timer se usa para medir la velocidad del motor en modo capture, contando la diferencia entre pulsos. El SysTick nos permite hacer el blink de la parada de emergencia. Cuando se entra en la interrupción y la flag de parada de emergencia se encuentra activa, apaga los transistores y hace parpadear el led rojo.

### Control de tensión con PWM

El PWM permite obtener en la salida una señal cuadrada, cuyo valor promedio es variable a través de su "duty cycle". Utilizamos 2 PWM para cada sentido de rotación del motor. Cada PWM está conectado a un transistor; las señales de alto/bajo nivel permiten cambiar el estado del transistor, que controla el paso o no de la corriente al motor para suministrar energía. El valor promedio de la señal cuadrada nos permite controlar la tensión continua de entrada del motor.

En nuestro caso, utilizamos 2 salidas del PWM1. El PWM se configura un poco como un temporizador. Se selecciona como reloj de entrada, y debe ser alto para evitar una variación demasiado grande de la corriente en el motor ( $f > 50 \text{ kHz}$ ). Hemos elegido un período de aproximadamente 3,5 microsegundos ( $= 290 \text{ kHz}$ ). Para calcular los valores que se deben colocar en el temporizador del PWM1, utilizamos la siguiente fórmula:

$$T = \frac{1}{PCLK} \cdot (PR + 1) \cdot (TMR + 1)$$
$$T = \frac{1}{25 \cdot 10^9} \cdot (20 + 1) \cdot (4095 + 1) = 3,44 \cdot 10^{-6} \text{ s} \approx 290 \text{ kHz}$$

Elegimos un valor de 4095 como entrada porque es la resolución (12 bits) del ADC. Luego, se pueden modificar los MR3 (para PWM1.3) y MR4 (para PWM1.4) para ajustar el 'duty cycle' de las dos salidas del PWM1.

## Gestión de la velocidad y dirección de giro

El control de la velocidad del motor se realiza mediante 2 medios (UART o ADC). Es posible alternar entre los 2 medios con una interrupción (EINT0) en flanco descendente, que modifica el valor del bit n.º 3 en el registro 'flags'.

### Velocidad con ADC

El ADC está configurado en modo burst. En la función de interrupción del ADC, calculamos un promedio de las últimas 10 valores para evitar que las perturbaciones en la entrada modifiquen bruscamente la salida PWM. Dado que la salida del PWM está en 4095, podemos utilizar directamente el valor del registro ADC. También fijamos un valor igual a 0 si es menor a 300 e igual al máximo (4095) si es mayor a 3800. En el caso de que se cambie a modo UART, se deshabilita la interrupción de ADC.

### Velocidad y dirección de giro con UART

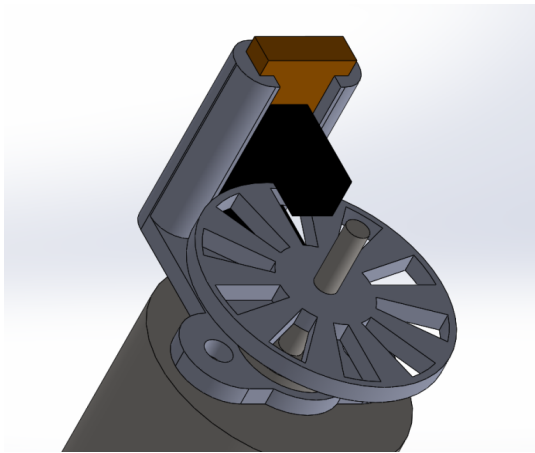
La gestión de la velocidad y la dirección del motor se realiza a través de 8 paquetes UART de 8 bits cada uno (16 bits en total). El mensaje contendrá 3 identificadores de posición, "." para indicar el inicio, "," para indicar la separación, y ";" para indicar el final del mensaje. Si alguno de estos llega en un lugar que no le corresponde, no se admitirá el mensaje. Entre el "." y "," va el sentido de giro, que puede ser 1 o 0, y entre ";" y ";" va el valor de velocidad, que va entre 0 y 4095. Para poder realizar el cambio de sentido de giro, primero se tendrá que frenar el motor, y luego se asigna el nuevo sentido.

### Configuración del UART

L'UART está configurado mediante el driver. Para recibir datos, este pasa por varias etapas:

- Recepción de un byte
- Almacenar el valor recibido en el búfer Rx
- Si es el final de la transmisión, se verifica el sentido de rotación del motor y se utiliza la función 'acomodar', que permite probar la correcta recepción de los bytes UART.

## Medición de la velocidad



Para medir la velocidad, utilizamos un motor de veleta. Un detector infrarrojo nos proporciona una frecuencia que podemos utilizar como entrada para el motor. Utilizamos el modo de captura del temporizador para calcular la velocidad promedio del motor. La señal de entrada desencadena una captura en el pin 1.26, que almacena los dos últimos valores recibidos. Con la diferencia de tiempo entre estos dos valores, podemos obtener un periodo de tiempo y, por lo tanto, la velocidad del motor.

## Función de giro

La función 'rotation' del motor permite modificar la velocidad y el sentido de rotación del motor. Toma como entrada 2 valores: velocidad y sentido de rotación. Esta función se coloca dentro del bucle while(1) del programa principal para realizar una prueba constante del valor del ADC o del UART. Cuando este valor alcanza una velocidad inferior a 300, el motor entra en modo 'frenando' (el flag de motor frenando se enciende), lo que significa que las 2 salidas PWM se establecen en 0.

## Configuración del DMA

El DMA nos permite transferir datos sin interrumpir al procesador principal. Es útil en nuestro caso porque de lo contrario el procesador podría ser interrumpido en su gestión del motor. Su objetivo será recuperar los datos del ADC en modo burst y almacenarlos en una zona de memoria. El procesador solo tendrá que leer el valor del registro de memoria para obtener los valores necesarios. Lo configuramos usando el driver.

## Configuración del channel

Usamos el canal 0. Lo más importante en esta configuración es que está en modo TRANSFERTYPE\_P2M (de dispositivo a memoria). No colocamos una dirección de origen, sino un dispositivo de origen (en nuestro caso, "GPDMA\_CONN\_ADC").

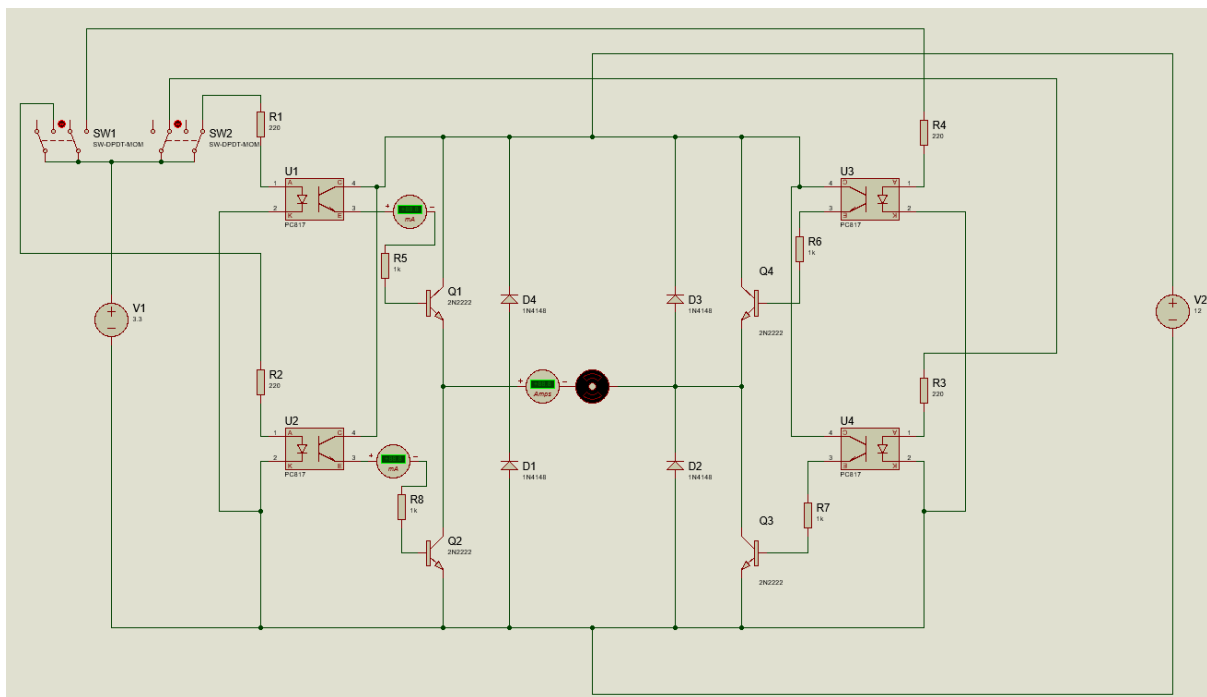
## Configuración de GPIO

Dentro del programa se ha dado lugar a 3 LEDs como luces de testigo, las cuáles indican estado; si el LED rojo se encuentra encendido, significa que el motor está detenido, si el LED verde se encuentra encendido, significa que el motor estará girando en sentido horario, y si es azul, será sentido antihorario. También notificarán si hubo una parada de emergencia, parpadeando el led rojo, si hubo un cambio de sentido, parpadeando el LED

azul o verde, y si hubo un cambio de modo de control, haciendo una secuencia de escalera con los 3 colores.

Como se optó por utilizar un LED RGB de cátodo común, la lógica de control será inversa, es decir, si se desea encender un LED, el pin correspondiente se deberá establecer en bajo, y caso contrario si se desea apagarlo, el pin se debe poner en alto. Al usar esta configuración, cuando el pin se encuentra en alto, tendrá una tensión muy similar a la que se encuentre el cátodo, por ende, la diferencia de voltaje será muy pequeña, lo suficiente para hacer que no llegue a la tensión de encendido del led. Cuando se desea encender el led, se pone la compuerta en 0, que básicamente la conecta a masa, en donde el limitante de corriente que tendremos será la resistencia que le corresponde a cada color.

## Montaje de la placa



Estructura del puente H, utilizamos optoacopladores para controlarlo para poder proteger a la placa en caso de fallas.

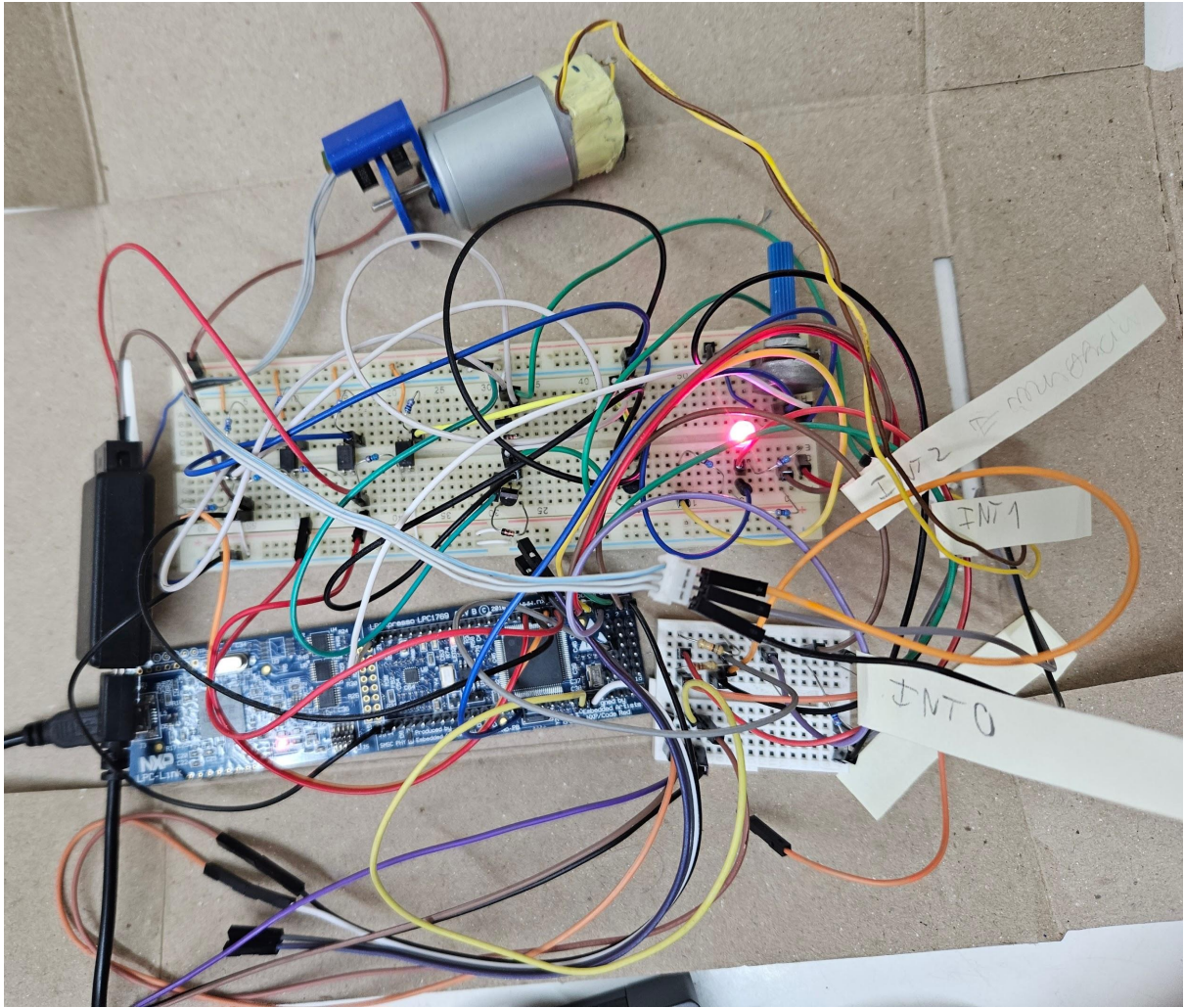
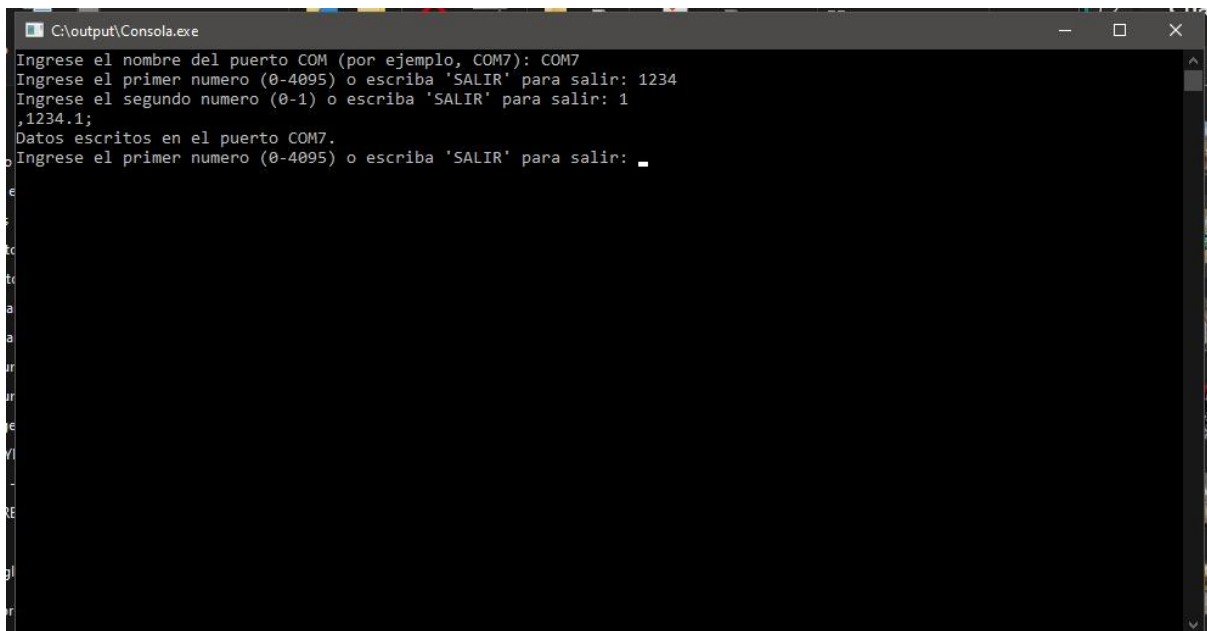


Foto de la placa con breadboard.



# Programa UART

Para facilitar la accesibilidad a la comunicación UART entre el dispositivo emisor y el proyecto, hemos diseñado un programa en C++, el cual envía datos de velocidad y sentido de giro ya formateados, listos para reconocerlos en la placa. Al iniciar el programa, el mismo solicita informar el puerto en que se encuentra conectado el módulo UART, en nuestro caso, utilizamos un módulo con un chip CP2102, que por lo general, se encontraba en el puerto COM7. Luego de especificar eso, el programa solicita un valor de velocidad entre 0 y 4095, y luego solicita un sentido de giro que puede ser 1 o 0. Si el número ingresado se encuentra fuera de sus límites, o si no es un número, el programa vuelve a solicitar el ingreso de los datos. Una vez ingresados ambos datos, se creará un string con un formato “.X,YYYY;”, en donde el “.” indica el comienzo de la cadena, la “X” el sentido de giro, la “,” el separador entre los datos, “YYYY” (que puede ser de 1 a 4 dígitos) la velocidad, y “;” para indicar el fin de la cadena. También se empleó una palabra clave “SALIR”, que se puede ingresar en cualquier momento del programa, y como su nombre lo indica, sirve para salir del programa.



```
C:\output\Console.exe
Ingrese el nombre del puerto COM (por ejemplo, COM7): COM7
Ingrese el primer numero (0-4095) o escriba 'SALIR' para salir: 1234
Ingrese el segundo numero (0-1) o escriba 'SALIR' para salir: 1
.1234.1;
Datos escritos en el puerto COM7.
Ingrese el primer numero (0-4095) o escriba 'SALIR' para salir: _
```

Esta ha sido una forma muy eficiente de transmitir los datos sin tener que realizarlo a mano y cometer posibles errores, aunque esto no afectaría en absoluto ya que el código del microcontrolador posee un algoritmo para detectar que estos datos se encuentren en orden.



# Conclusiones

Este trabajo, por más simple que parezca, fue un desafío para nosotros, debido a que precisamos constancia en la lectura del manual y hoja de datos del microcontrolador para saber qué registros modificar, qué registros observar, y cómo implementar funciones nuevas al programa en desarrollo. Una forma de trabajar que nos permitió avanzar de forma rápida fue el testeo de módulos de manera individual o en conjunto, como por ejemplo, utilizar ADC con PWM, y UART con GPIO. También nos facilitó el testeo el uso de un analizador lógico, ya que con él pudimos comprobar que los mensajes por UART eran los esperados, cómo tratarlos en cada caso, y también nos facilitó la configuración del módulo PWM. Hemos concluido que la configuración del módulo DMA fue una de la que más tiempo nos demandó para hacerla funcionar correctamente. Lo más importante de todo este desarrollo es el conocimiento de esta arquitectura que nos queda para la programación dispositivos similares y poder encarar proyectos mucho más complejos con menor dificultad.