# Universidad Nacional de Córdoba

## Facultad de Ciencias Exactas, Físicas y Naturales

## Electrónica digital 3

***Trabajo Práctico Final:***

***Brazo robótico configurable***

*Profesor* : Ing. Martin Ayarde

*Integrantes*:

- ✓ Santana, Ignacio
- ✓ Viccini, Patricio

---

# Descripción del proyecto

El proyecto consiste en un brazo robótico de 4 grados de libertad hecho con una impresora 3D y motorizado por 4 motores paso a paso nema 17. En la punta del brazo se encuentra una pinza, movida mediante un servo motor, que es capaz de recoger objetos. El movimiento será comandado por una aplicación de PC y se comunicará con la placa a través del puerto serie, mediante el módulo UART.

# Requerimientos del proyecto planteados inicialmente

- Debe permitir al usuario mover el brazo mediante una aplicación de PC
- Debe poder guardar una serie de movimientos para luego reproducirlos de forma repetitiva
- Con la base completamente apoyada sobre una superficie plana horizontal(ejemplo una mesa), la estructura debe poder sostener su propio peso en cualquier posición posible
- Debe tener al menos 5 modos de funcionamiento
- Debe poder agarrar, levantar y soltar objetos
- Un modo de funcionamiento debe ser el que permite la configuración de las velocidades de los steppers
- Un modo de funcionamiento debe ser el libre manejo del brazo, con la garra controlada desde la aplicación de PC
- Un modo de funcionamiento debe ser el libre manejo del brazo, con la garra controlada desde un potenciómetro
- Un modo de funcionamiento debe ser el grabado de los movimientos a repetir
- Un modo de funcionamiento debe ser el estar constantemente repitiendo los movimientos guardados
- El control de la garra mediante el potenciómetro debe reaccionar en menos de 1 segundo
- Al comenzar a grabar, debe detenerse el movimiento del brazo
- Al comenzar a grabar, debe guardar esa posición del brazo como posición de referencia relativa
- Al terminar de grabar, debe retornar a la posición inicial de referencia y guardar este movimiento como último de la serie
- Al ejecutar los movimientos guardados, debe inhabilitar los controles de movimiento
- La app debe comunicarse con la placa mediante puerto serie
- La app debe permitir cambiar el modo de funcionamiento
- La app debe mostrar el modo de funcionamiento actual del sistema

---

- La app debe tener unas instrucciones donde se expliquen los controles
- La app debe permitir controlar el movimiento mediante el teclado y también mediante el click del mouse

## Especificaciones

A continuación se tienen las distintas especificaciones para los distintos bloques funcionales del proyecto:

*Control de los motores paso a paso*

*Imagen 1: Motor Paso a Paso Nema 17*

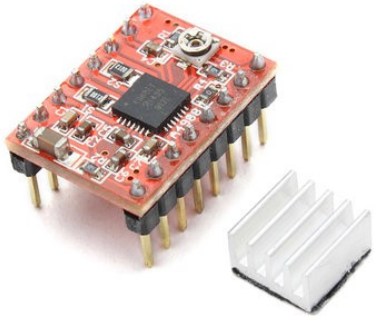

*Imagen 2: Driver A4988 para motores Paso a Paso*

Se desarrolló una biblioteca basada en CMSIS y pensada para los drivers del mismo funcionamiento que el A4988 llamada *stepper_motors,* que mediante los timers y los pines GPIO controla de cada stepper cosas como los pines asignados, la velocidad, el microstepping seleccionado, el sentido de giro, el timer asignado y la ejecución de un paso. Los pasos del motor con esta biblioteca son controlados mediante el match 0 del timer asignado al motor, el cual se corresponde con el periodo entre cada paso(o micro paso), cuando el match ocurre se produce una interrupción y dentro de la rutina de servicio de la misma, si el stepper está en un estado en el cual debe hacer un paso, se ejecuta el paso poniendo un 1 a la salida del pin de step asignado al motor y luego un 0. Un mismo timer puede ser compartido por más de un stepper, pero estos pasan a compartir velocidades, ya que las mismas son propias del timer y no de cada motor.
En el programa principal estos motores se inicializan y configuran. Luego cuando llega un comando por UART, si esté indicase que algún stepper debe comenzar a girar en alguna dirección o detenerse, se modifica el estado del stepper para que la acción sea ejecutada al producirse las interrupciones de su timer asignado.
Para el control de cada stepper se utiliza un circuito que maneja la corriente y la tensión necesaria para los steppers (**anexo 1 figura 2**).

---

*Control del servo*

Se escribió una biblioteca basada en CMSIS llamada *servo_motors,* que mediante el módulo PWM controla el servo cosas como el canal PWM asignado, el ciclo completo, el valor del duty_cycle, los límites superior e inferior del duty_cycle, el periodo de cada tic(esto impacta directamente en la resolución de la onda, a menor periodo mayor resolución) y el estado.
El ángulo de giro del servo se controla según la onda que recibe, la misma es una PWM donde el valor del duty_cycle se traduce en el ángulo en el que el servo detendrá su giro. Los límites de dicho duty cycle y el periodo de la onda(el valor a colocar en el MR0 del módulo PWM) depende de cada servo(**anexo 1 figura 3**).

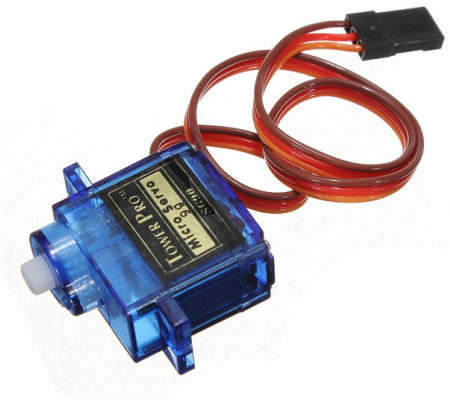


*Imagen 3: Servo motor utilizado*

El servo que fue utilizado funciona con niveles de 5V, por lo que fue necesario un *level shifter* para la señal PWM generada por la LPC, este circuito cuenta con cuatro resistencias y dos transistores BJT(**anexo 1 figuras 4 y 5**).
En el programa principal este motor se inicializa y configura. Según el modo de funcionamiento seleccionado, el servo se comanda desde UART mediante la aplicación de PC o según el valor leído por el módulo ADC conectado a un potenciómetro.
Desde la app de PC: cuando llega un comando por UART, si esté indicase que debe cerrar la garra o abrir la garra, se activa el SysTick y se cambia el estado del servo para que comience a moverse. Dentro de la rutina de servicio del SysTick, según si el estado es abriendo o cerrando, por cada interrupción se avanza una unidad hacia el valor límite que se busca. Esto se realiza para que el movimiento del servo sea más fluido y lento, evitando los picos de corriente que se dan si se cambia el duty_cycle de un límite al otro de golpe. Al alcanzar el límite deseado el estado cambia a cerrado o a abierto y se desactiva el SysTick.
Desde el potenciómetro: constantemente se realizan conversiones en el modo *burst* del ADC con un rate bastante bajo, disminuyendo así el ruido de la lectura, y al final de cada una se produce una interrupción. Dentro de la rutina de servicio del ADC se convierte el valor leído del potenciómetro en un valor de duty_cycle válido y la PWM que va al servo pasa a tener este nuevo duty_cycle.

*Modulo UART*

Para el proyecto, se optó por utilizar la comunicación UART para intercambiar datos entre la LPC1769 y la computadora, debido a que el microcontrolador cuenta con 4 módulos UART,

---

y ademas contabamos con el módulo PL2303, el cual convierte las señales de comunicación serie a señales usb, lo cual simplifica mucho la comunicación. Se utilizó el módulo UART0 de la placa lpc, y se lo configuro por defecto:

- Baud Rate de 9600 Baudios
- Sin bit de paridad
- 1 bit de Stop
- Ancho de datos a transmitir 8 bits
- Nivel de disparo de FIFO de 1 byte

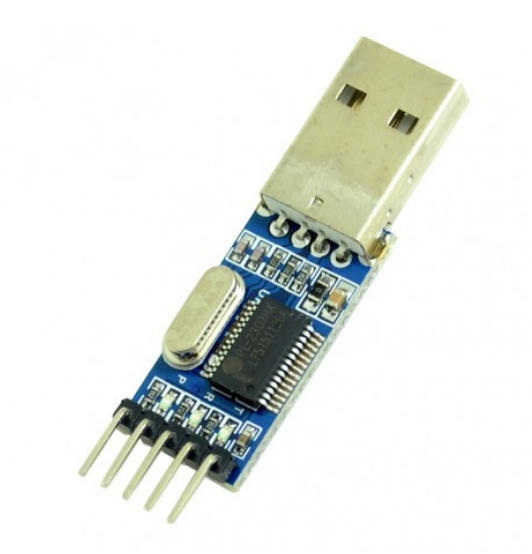

*Imagen 4: Módulo PL2303*

La configuración del puerto serial del COM4 fue configurado de la misma manera, a través de la aplicación de escritorio ,para que la comunicación pueda interpretarse de manera correcta por ambos dispositivos.
También se configuró el módulo UART0 para que cada vez que llegue un dato al buffer de recepción FIFO, se levante una interrupción que ejecute el comando enviado por la computadora. Los comandos pueden ser de tres tipos: Comando de acción, Comando de configuración, Comando de cambio de estado.

- *Comando de acción:* Estos puede comenzar los movimientos de los 4 motores paso a paso, tanto en forma horaria o antihoraria, detenerse o abrir/cerrar la garra del brazo, es decir mover hacia un lado o hacia otro el servo.
- *Comando de configuración:* Este comando es aquel que va a configurar las velocidades de los motores, y se ejecutan solo bajo modo configuración. Este comando consta de dos informaciones : el primer carácter es el motor a configurar y el segundo carácter es la velocidad.
- *Comando de cambio de estado:* Estos comandos simplemente cambian los estados de funcionamiento del brazo.

**Aplicación de PC**

Fue desarrollada en C# a través del IDE *Visual Studio 2019* y se comunica con el brazo robótico mediante el puerto serie. Permite cambiar de modo de funcionamiento, configurar velocidades y controlar el movimiento del brazo robótico.
Cuenta con una ventana de *help* en la que se explican los controles para un fácil aprendizaje de los mismos.
Los botones en pantalla reaccionan según el estado y el modo de funcionamiento.
Si hay problemas al conectarse con el puerto serie lanza una alerta.
En los modos que lo permiten, los motores se mueven mientras el control correspondiente a ese movimiento se mantenga presionado y se detiene al soltar, ya sea la tecla asignada o el click del mouse sobre el botón en pantalla.

---

# Modos de operación del brazo robótico

***Configuración***

El brazo permanece quieto mientras y se habilita la modificación de las velocidades de los motores. En nuestro caso hay 3 velocidades que son configurables, velocidad del motor de la *Muñeca* y del motor de el *Codo*, velocidad del motor de el *Hombro* y velocidad del motor de la *Base*(**anexo 1 figura 6**). Los niveles de velocidades posibles van del 1 al 10, siendo el 10 la velocidad máxima.
Por cada velocidad que se quiere configurar se reciben 2 comandos, el primero indica cual motor va a cambiar su velocidad y el siguiente indica el nivel de velocidad seleccionado.

***Manual***

El control de los motores responde a los comandos enviados desde la app de PC e interpretados por la rutina de interrupción del UART. El ADC se encuentra desactivado y el servo solo se controla a través de comandos por UART.
Periódicamente los timers asignados a los motores interrumpen y según el estado de los motores producen pasos en algún sentido o ninguno manteniéndose detenidos

***Manual+Pote***

Los steppers se controlan igual que en el modo Manual pero el servo es controlado por el potenciómetro. En este modo es donde el ADC está activado y el comando UART que corresponde al servo está desactivado.

***Grabación***

Al pasar a este modo se habilita el timer 1 y se comienza a llevar un registro de los cambios de estado de cada motor y cuanto tiempo ocurre entre cada cambio (esto leyendo el TC del timer 1), de esta forma se guarda un *reg stamp* por cada cambio marcando la secuencia de los movimientos del brazo. También se lleva registro de cuántos pasos y en qué sentido se aleja cada motor de la posición en la que se comenzó a grabar. Al pasar al modo de *Ejecución* se graban cómo últimos movimientos los que hacen volver al brazo a su posición inicial, realizando estos movimientos también sin necesidad de recibir más comandos.
En el sentido del control del movimiento es idéntico al modo *Manual* solo que además ocurre todo el proceso de grabación anteriormente descrito.
Al pasar de este modo a otro que no sea *Ejecución,* se borran los movimientos grabados.

***Ejecución***
Se mantiene reproduciendo los comandos guardados, esperando entre comando y comando el valor de tiempo del último reg stamp leído. La espera se lleva acabo usando el

---

match 0 del timer 1, cada vez que esté interrumpe se carga el valor siguiente y se ejecuta el comando que sigue.
En este modo los comandos dirigidos a los motores del brazo están inhabilitados, por lo tanto el movimiento solo depende de la secuencia grabada.
Al salir de este modo se borra la secuencia que fue ejecutada.

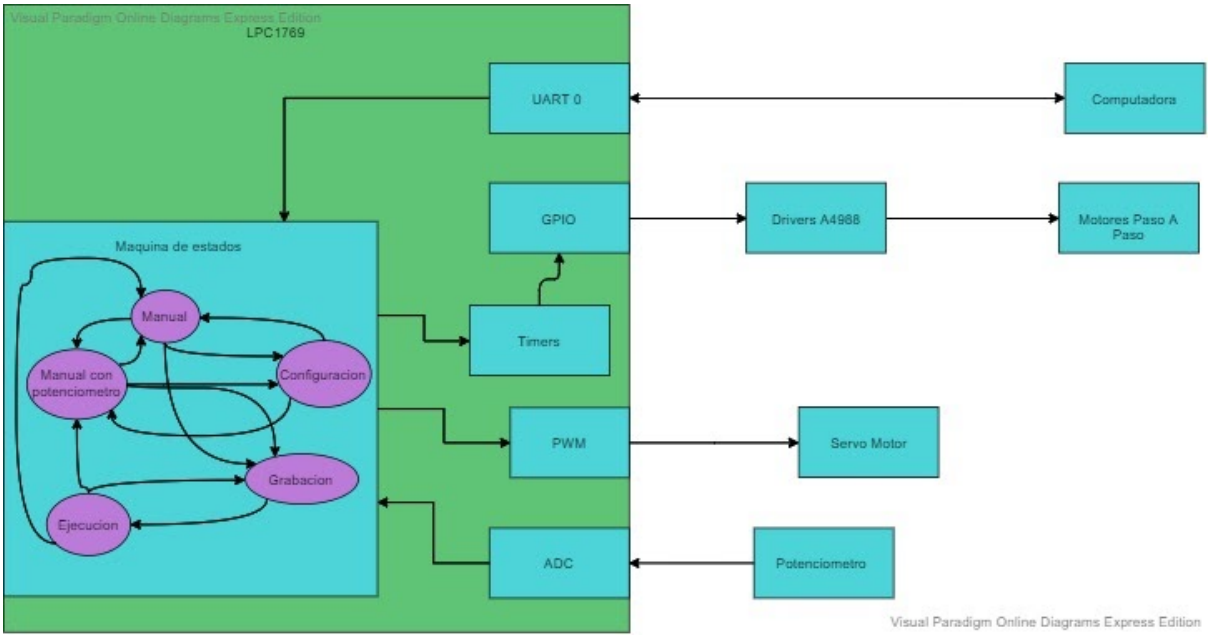


*Imagen 5 : Diagrama en bloques del proyecto*

## Componentes utilizados para el proyecto

- 1 Placa LPC1769
- 1 Bridge USB a UART PL2303
- 4 stepper motor driver A4988
- 4 stepper motor Nema 17
- 1 servo motor SG90
- 1 potenciómetro de 10K ohm
- 2 transistores BJT 2N2222A
- 1 resistencia de 10K ohm
- 1 resistencia de 4,7K ohm
- 2 resistencias de 1M ohm
- 1 capacitor electrolítico de 100 µF
- Protoboard
- Cables
- Partes plásticas del brazo robótico
- Tornillos

---

# Cálculos Realizados para el proyecto

*Cálculo de los transistores para el level-shifter*

Cuando esté en alto la entrada queremos que la corriente de salida de la LPC sea baja, ponemos por ejemplo 3$\mu$A

$$I_{b1} = \frac{V_{OH} - 0.7V}{R_{b1}} = 3\mu A$$

$$R_{b1} = 866\ K\Omega \simeq 1M\Omega$$

Luego $I_{b1}$ serán 2.6 $\mu$A. Sabiendo como dato que $\beta$ = 200 para el 2N2222, tenemos :

$$R_{c1} = \frac{5V - 0.35V}{I_{b1} * \beta} = 8,2\ K\Omega \simeq 10K\Omega$$

En el cálculo anterior se despreció la corriente de base del segundo transistor, ya que en cualquier estado (1 o 0 lógico) esta es despreciable. Entonces cuando esta en uno la entrada del level shifter, la base del transistor tiene una tensión aproximada a 0, por lo que el segundo transistor está en estado de CORTE y a la salida hay 5V.
Cuando la entrada está en bajo o 0, el primer transistor está cortado, porque en la base del segundo hay 5V. Para mantener bajo el consumo de corriente, hacemos que $I_{b2}$ = 5 $\mu$A.

$$I_{b2} = \frac{5V - 0.7V}{R_{b2}} = 5\mu A$$

$$R_{b2} = 860\ K\Omega \simeq 1M\Omega$$

Calculamos la resistencia de colector para que sature a la salida:

$$R_{c2} = \frac{5V - 0.35V}{I_{b2} * \beta} = 5K\Omega \simeq 4,7K\Omega$$

Entonces la salida será 0 ( en realidad es una tensión de saturación del transistor 2, pero esta es muy baja comparada con 5V)

*Cálculo de prescaler para Timer*

En nuestro caso, al usar la librería CMSIS, se setea por defecto que los timers tienen un PCLK = CCLK/4=25MHz. La librería se pensó para que el que la usara además de nosotros debiera ingresar el valor de match n en $\mu$s, por lo tanto los contadores del timer deben incrementarse cada 1 $\mu$s, por lo que los valores de prescaler son seteados en :

$$PR = 25Mhz * 1\mu s = 25$$

Esto se realiza para cada uno de los timers usados por los motores paso a paso

---

*Cálculo de frecuencia del ADC*

Nuevamente en el caso del ADC, la librería CMSIS setea por defecto PCLK = CCLK/4=25MHz. Se muestrea a una frecuencia bien baja, de manera que se filtren de algún modo las altas frecuencias que afectan al potenciómetro. Si bien este no sería un correcto modo de filtrado, ya que se debería usar un filtro activo, se redujo significativamente el ruido cuando redujimos a este valor la frecuencia. Se usó el modo burst del ADC ya que no es necesaria una conversión tan rápida, por lo que el ADC convertirá un valor cada un tiempo prolongado. Como estamos en modo burst, seteamos el prescaler interno del ADC en el máximo valor de 256

$$f_{ADC} = \frac{25Mhz}{65*256} = 1503Hz$$

Esa será la frecuencia a la que funcionara el ADC.

## Conclusión

Como meta principal se buscaba demostrar y reforzar las habilidades adquiridas en la materia *Electronica Digital 3*, consideramos que con este proyecto se consiguió este objetivo al aplicar los conocimientos adquiridos para el desarrollo del brazo robótico. A su vez, se pudo aprender temas que iban más allá de lo que la materia comprendió durante su cursado, como por ejemplo el manejo del *modulo PWM*, el desarrollo de una app de escritorio o las cuestiones mecánicas del dispositivo.
Se logró un dispositivo que integra distintos módulos de la *LPC1769*, así como también consigue controlar, de la manera en la que los requerimientos describen, el hardware externo a la placa y su estado interno.
Según el estado actual del proyecto, pensamos que se pueden seguir desarrollando mejoras sobre él mismo e incluso agregar nuevos sistemas que lo complementen. Por ejemplo, un sistema de control realimentado de la posición, un tipo comunicación vía Bluetooth con su respectiva aplicación de celular, nuevos modos de funcionamiento, más parámetros configurables, la fabricación de una PCB para el montaje del circuito, etc. Esto nos refleja que se consiguió un dispositivo en el que seguir trabajando y actualizando para seguir aprendiendo y creciendo como futuros ingenieros.
Por último, se manifiesta que el trabajo fue una experiencia fructífera desde lo académico y lo profesional y que pudo incentivar nuestra curiosidad por los temas comprendidos.