Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Implementación de un entorno de comunicación Bluetooth basado en el módulo HC-06

Autor:

Mucientes San José, David

Tutor(es):

Pérez Turiel, Javier

Ingeniería de Sistemas y Automática

Valladolid, Julio 2021.

AGRADECIMIENTOS

En primer lugar, me gustaría quedar constancia que este trabajo y todo lo que conlleva, ha sido posible gracias al apoyo incondicional de mi familia, mis padres y a mi hermana que siempre me han apoyado y han creído en mí. De ellos he aprendido que con constancia y trabajo los resultados llegan por sí solos.

Especial mención a mi madre, que aun en todos los malos momentos siempre he sabido darme el mejor consejo para poder continuar y alcanzar el objetivo final.

Agradecer a mi tío Mauro su apoyo y sus consejos, siendo un gran soporte para poder estudiar durante estos años.

A Raquel, que me ha acompañado en esta etapa de mi vida, confiado y dándome ánimos en todas las circunstancias.

Por último, agradecer a mi tutor Javier y a la encargada del ITAP Ana su gran ayuda para la realización del trabajado, solucionándome las dudas y los problemas que me han surgido

RESUMEN

El objetivo de este trabajo es la implementación de un entorno de comunicación basado en el módulo Bluetooth HC-06. Con su utilización, buscaremos conectar la interfaz desarrollada en el PC y dos Microcontroladores, Arduino y el Microcontrolador de la familia C2000 LAUNCHXL-F28069M vía Bluetooth.

La realización de este proyecto conlleva conocer el funcionamiento de este tipo de dispositivos, funcionando juntamente con los microcontroladores. Para ello se hace indispensable la compresión de los métodos de programación necesarios para su uso final.

Para el control y el manejo de los datos se ha creado una aplicación grafica con la que interacción entre usuarios y dispositivos es más intuitiva. Con ella vamos a poder realizar la recogida de los datos que nos van a ser útiles. A partir de este punto ya podremos establecer la comunicación entre el equipo principal y los microcontroladores.

Como objetivo final del estudio se busca concluir si el módulo utilizado, es adecuado para establecer una conexión fiable y que pueda ser útil de cara a desarrollar prototipos que se asemejen al diseño final de comercialización. Con la realización de estas fases de diseño, se reducen errores y se puede producir mejoras en el producto final que no se valoraban en un principio.

PALABRAS CLAVE

Microcontrolador, Arduino, Texas Instruments, Bluetooth, Visual Studio, Interfaz

INDICE DE CONTENIDO

INDICE DE TABLAS

INDICE DE ILUSTRACIONES

CAPITULO 1: INTRODUCCION Y OBJETIVOS

1.1. INTRODUCCION

Con la elaboración de este trabajo de fin de grado (TFG) se pretende desarrollar una comunicación inalámbrica estable entre una interfaz que se ejecuta en un ordenador y un microcontrolador, vía Bluetooth.

La comunicación Bluetooth se realiza entre el ordenador y el módulo Bluetooth, el HC-06. Los datos, por tanto, se intercambian entre el ordenador y el microcontrolador. El microcontrolador se comunica con el módulo HC-06 mediante una comunicación serie a través de los hilos TX y RX.

En la primera parte, a fin de conocer y poder tener un desarrollo más cercano al objetivo final el microcontrolador utilizado es el Arduino UNO, con ello podemos comenzar el desarrollo del programa, y poder tener una perspectiva de las posibilidades reales del proyecto, así como de los problemas que puedan surgir.

Como segunda parte, el microcontrolador usado el LAUNCHXL-F28069M de Texas Instruments. Se escoge este microcontrolador ya que es el que actualmente se encuentra en el prototipo de Robhand. Este es el que controla los actuadores del exoesqueleto, lo que posibilita el movimiento de la mano del paciente.

Para poder llevar el control de la comunicación, se va a crear una interfaz de usuario para que, desde el ordenador, se puedan tanto, enviar consignas con las posiciones deseadas de cada actuador, como la de conocer la posición actual de los mismos.

Con la realización del presente trabajo se pretende entender y analizar la comunicación serial con Arduino UNO R3. En concreto sus especificaciones, funcionamiento y el tipo de programación que se debe utilizar para su desarrollo. Comprender el uso del módulo bluetooth HC-06. Estudio del desarrollo a través de circuitos impresos de Texas Instruments, en concreto de la familia C2000 el dispositivo LAUNCHXL-F28069M. Aprender los fundamentos de la programación en C#, para el desarrollo de interfaces a través de Visual Studio, con su aplicación de Windows Forms.

Se busca determinar, si el módulo comercial HC-06 resulta adecuado para el fin que se pretende obtener del desarrollo del trabajo.

1.2. OBJETIVOS

El objetivo principal del TFG es eliminar la comunicación cableada por USB entre el PC y TMS320F28069M, reemplazándola por la comunicación BLE y que pueda ser de utilidad dentro de la plataforma Robhand.

Para lograr el objetivo final, se han realizado 2 etapas, la primera consistirá en completar la comunicación utilizando la plataforma Arduino y la segunda con la placa de Texas Instruments LAUNCHXL-F28069M.

Se necesita diseñar un programa en Visual Studio para que el usuario pueda controlar la comunicación. Esto será, iniciar la comunicación con el módulo bluetooth, poder definir los datos a enviar y solicitar los datos al microcontrolador.

Durante la primera etapa el objetivo es completar la comunicación entre le ordenador y el Arduino utilizando el módulo HC-06. Para ello será necesario configurar en Arduino que puertos serie (TX y RX) se va a usar para poder enviar y recibir datos a través del módulo bluetooth. Necesitaremos poder comprobar que los datos recibidos en Arduino son correctos, y además recoger los datos para poder enviarlos de vuelta al ordenador. Para ello será necesario que configuremos otra comunicación a través del monitor serial.

En la segunda parte, el objetivo será sustituir el Arduino por el LAUNCHXL-F28069M. Esto conlleva la creación de un nuevo programa para la placa de desarrollo de Texas, pero reutilizar el programa creado de Visual Studio. De nuevo, tendremos que configurar el dispositivo y realizar las conexiones para que se comunique vía serial (TX y RX) con el módulo bluetooth. Deberemos poder leer datos para poder enviarlos al ordenador, para ello usaremos el monitor serial del propio IDE de Texas o de otro programa externo.

CAPITULO 2: ESTADO DEL ARTE

2.1. Robhand

La plataforma Robhand es un proyecto que nace en el Instituto de las Tecnologías Avanzadas de la Producción (ITAP) de la Universidad de Valladolid en colaboración con la empresa CyL Imas D de Salamanca, y dos centros asistenciales, el Hospital Clínico Universitario de la Universidad de Valladolid y la Corporación de Rehabilitación Club Leones Cruz del Sur de Chile. Se encuentra financiado parcialmente por el Centro para el Desarrollo Tecnológico Industrial (CDTI) [1]

Este entorno creado se fundamenta en un dispositivo robótico para rehabilitación de mano, mediante terapias activas y pasivas, cuyo objetivo principal es ayudar a las personas con discapacidad neuromotora.

Está diseñado para interactuar con personas que muestran una limitación funcional de la mano. Este dispositivo no busca sustituir la mano, sino ayudar al paciente a recuperar la capacidad de realizar movimientos de apertura y cierre de la mano.

Para poder llevar a cabo este proyecto es necesario el desarrollo de numerosos elementos complementarios que, unidos, nos permiten alcanzar el objetivo final. Estos son el exoesqueleto, que es la estructura creada para que el paciente pueda reposar su mano, actuadores como los servomotores que van a realizar los movimientos de la mano. Estos van a estar controlados por un microcontrolador, el cual se va a comunicar con el entorno software creado para el paciente.

Durante el desarrollo del proyecto, se han creado varios prototipos para ir cumpliendo con las fases de evolución del proyecto. Actualmente se continua con su progreso y mejora.

Ilustración 1: Ultima versión del prototipo Robhand

2.2. Visual Studio

Visual Studio es un Entorno de Desarrollo Integrado (IDE) creado y desarrollado por Microsoft. Reúne una gran cantidad de herramientas que nos permiten crear multitud de aplicaciones de escritorio, aplicaciones móviles, aplicaciones web o servicios web. Ofrece una gran cantidad de opciones que permite a todo tipo de personas, ya sean grandes desarrolladores o alguien con poca experiencia crear aplicaciones de escritorio muy completas.

Está basado en BASIC (Beginner's All-purpose Symbolic Instruction Code), un lenguaje de programación de alto nivel, que puede ser tanto interpretado como compilado, no estructurado, y de fácil aprendizaje, Visual Basic .NET es un lenguaje de programación orientado a objetos que cuenta con los beneficios que le brinda .NET Framework, el modelo de programación diseñado para simplificar la programación de aplicaciones en un entorno sumamente distribuido. [2]

Para el objetivo descrito en cuanto a la interfaz de usuario para la comunicación vamos a utilizar la opción de Windows Forms que nos ofrece Visual Studio.

Windows Forms es un marco de interfaz de usuario para compilar aplicaciones de escritorio de Windows. Es una de las vías más provechosas de crear aplicaciones de escritorio basadas en el diseñador visual proporcionado en Visual Studio. [3] Con ello podemos realizar aplicaciones muy agradables visualmente y amigables con el usuario final, al tener el mismo aspecto que la propia interfaz de Windows.

2.3. Microcontroladores

Un microcontrolador es un circuito integrado digital que debido a su capacidad de programación puede usarse para numerosas aplicaciones. Está formado por una unidad central de procesamiento (CPU), unidades de memoria (RAM y ROM) y periféricos (unidades de entrada y de salida). Todas estas partes están interconectadas entre sí. Como vemos tiene características similares a las de un ordenador personal estándar. Su función es la de automatizar procesos y procesar información.

Al tener todo el hardware integrado dentro del mismo chip, para poder usarlo debemos especificar el funcionamiento de sus elementos a través del software, con el programa que el microcontrolador requiera. Dentro de la memoria se guardan los programas que se desarrollan y las CPU va a ser la encargada de procesas paso a paso las instrucciones del programa a ejecutar.

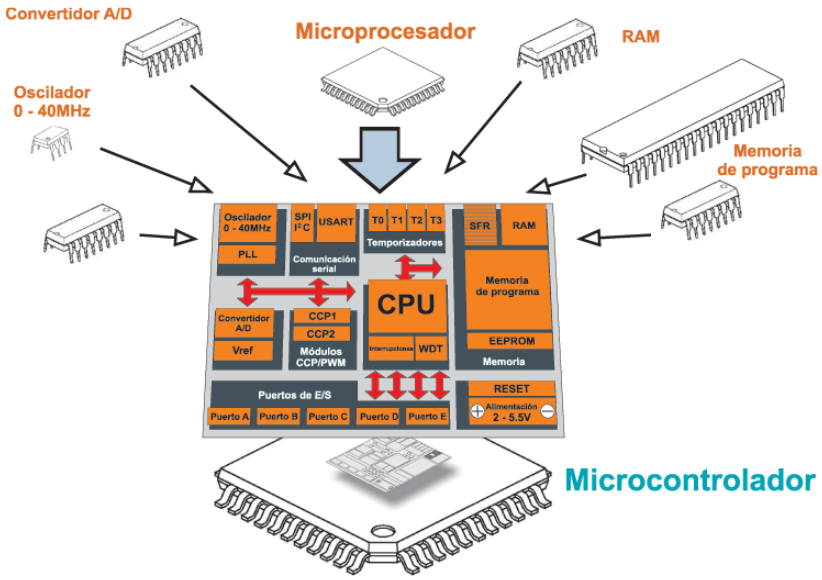

Ilustración 2: Esquema de un Microcontrolador [4]

Las ventajas del uso de microcontroladores son su bajo coste y su fácil integración en la mayoría de los circuitos y proyectos de electrónica. Su capacidad para almacenar y ejecutar programas únicos los hace muy versátiles para infinidad de aplicaciones.

Visto que es un microcontrolador y las ventajas que nos ofrecen, los microcontroladores utilizados para la realización del trabajo han sido:

2.3.1. Arduino UNO

Arduino es una plataforma de hardware de código abierto. Es una placa de circuito impreso que integra un microcontrolador ATMEL. Dependiendo de la placa podemos tener un tipo un tipo de microcontrolador u otro, Arduino se basa en los ATMEGA168, ATMEGA328 y ATMEGA1280. La placa cuenta con una serie de puertos y conexiones que le permite comunicarse con otros dispositivos, como puede ser el propio PC, otras placas, módulos o sensores. Existen diversas placas Arduino, cada una con unas características determinadas que las hacen más adecuadas depende el uso que se requiera de ellas. En cualquier caso, todas ellas comparten una serie de elementos comunes que son diferencia respecto al resto de plataformas para el diseño electrónico.

Entre el resto de las plataformas posibles, que podríamos elegir para el desarrollo del proyecto donde las herramientas que se nos prestan son más complicadas a la hora de programar, Arduino es la que más sencillo y simplificado nos permite trabajar, debido a su amplia compatibilidad con muchos sensores y a una interfaz sencilla.

Las principales ventajas frente a otras plataformas son:

- **Multi-plataforma:** el software de Arduino funciona en los 3 sistemas operativos más usados en la actualidad, Windows, Linux y Mac-OS.
- **Programación sencilla:** en el aprendizaje se resulta muy sencillo con el ambiente de programación que ofrece Arduino a los usuarios.
- **Accesible:** son unas placas baratas y fáciles de adquirir en numerosas tiendas, ofreciéndonos una buena potencia para diferentes el desarrollo de aplicaciones
- **Software y hardware ampliable y de código abierto:** tanto el hardware como el software son de licencia libre, por lo que cualquier programador o desarrollador puede modificar tanto hacer una versión similar o modificar las librerías propias para adaptarlas al uso que le quieran dar.

Una vez conocidas las características que se han descrito, vamos a usar para este proyecto, la placa Arduino UNO.

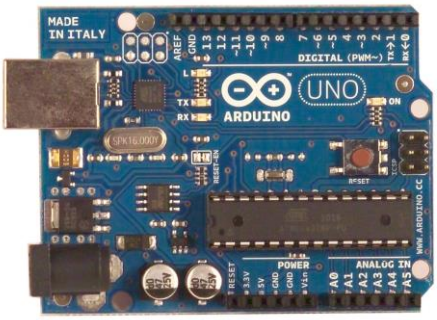

Ilustración 3: Arduino UNO

Las principales características de la placa son [5]:

Microprocesador	Atmega 238P
Voltaje de Operación	5 V
Voltaje de Entrada (recomendado)	7-12 V
Voltaje de Entrada (limite)	6-20 V
Velocidad de reloj	16 MHz
Pines digitales de E/S	14 (6 con salida PWM)
Pines digitales PWM de E/S	6
Entradas analógicas	6
Corriente DC por pin de E/S	20mA
Corriente DC para 3.3V pin	50mA
Memoria de Programa (Flash)	32 Kb
Memoria de Datos (SRAM)	2Kb
Memoria Auxiliar (EEPROM)	1Kb
Largo	68.6mm
Ancho	53.4mm
Peso	25g

Tabla 1: Características Arduino UNO

Las principales partes que componen la placa son las que se muestran en la figura siguiente:

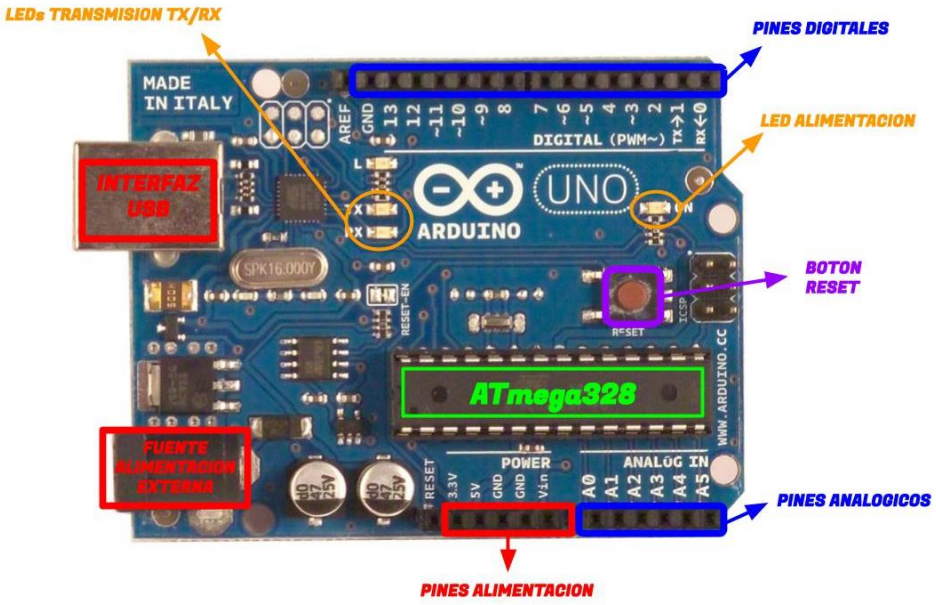

Ilustración 4: Ubicación de los componentes en la placa Arduino UNO

2.3.1.1. Pines de alimentación

Entre estos pines se encuentran:

2.3.1.1.1. Pin de 3.3V

Desde este pin obtenemos 3.3V para cualquier dispositivo que necesite de esta tensión y tiene una corriente máxima de 50mA. Es generada por el chip FTDI que se encuentra integrado en la placa

2.3.1.1.2. Pin de 5V

Este pin saca 5V del regulador de la placa. Dicho regulador es necesario puesto que puede ser alimentado con distintos voltajes

2.3.1.1.3. Pin de Tierra

Masa del circuito para los pines, es decir la tensión de referencia 0V.

2.3.1.1.4. Pin de Vin

Es el voltaje de entrada cuando se usa una fuente de alimentación externa (no tiene en cuenta la conexión USB). Se puede proporcionar voltaje a la placa a través de este pin, o en caso de que se esté utilizando una fuente de alimentación externa tomar el valor que está siendo suministrado.

2.3.1.2. Pines digitales de entrada y salida

En estos pines conectaremos los diferentes sensores o actuadores. Desde podremos recoger datos o enviar señales a los actuadores. De entre todos los pines de los que dispone la placa, tenemos 14 pines digitales que pueden utilizarse como entrada o salida con las funciones pinMode(), digitalWrite() y digitalRead(). Operan a 5V. Estos reciben como máximo 40mA y disponen de una resistencia pull-up (la cual por defecto se encuentra desconectada) de 20-50kOh. Hay numerosos pines que se encuentran reservados para usos específicos [6]

- Serie: 0(RX) y 1(TX). Utilizados para recibir (RX) y trasmitir (TX) datos serie. Están directamente conectados a los pines serie del microcontrolador. Utilizando estos pines podremos conectarnos con otras placas.
- Interrupciones externas: 2 y 3. Estos pines pueden ser configurados para activar interrupciones.
- PWM: 3, 5, 6, 9, 10 y 11. Proporcionan una salida de 8 bits en modo PWM.
- SPI: 10-13. Estos pines soportan la librería de comunicación de dispositivos SPI.
- LED: 13. Este pin está conectado con un led de la placa. Es completamente configurable, cuando se le asigne un valor HIGH se encenderá, en cambio en LOW estará apagado.

2.3.1.3. Pines Analógicos

Esta placa contiene 6 pines de entrada analógicos. Los elementos que se conecten aquí suelen tener mayor precisión que los digitales pero su uso requiere de una lógica levemente mayor. Más adelante se comentará el uso de un termistor analógico.

2.3.1.4. Conector USB

De entre los diferentes tipos de conectores USB que existen, el Arduino UNO utiliza el tipo B hembra. La placa se puede alimentar directamente desde este puerto cuando lo conectamos el ordenador sin necesidad de utilizar otra fuente de alimentación externa. Es a través de este bus, por donde cargamos le programa desarrollado en el software de Arduino.

2.3.1.5. Botón de Reset

Utilizando este botón podemos reiniciar la ejecución del código del microcontrolador

2.3.1.6. Microcontrolador ATmega328

El microcontrolador es el elemento más importante de la placa. Es donde se instalará y ejecutará el código que se haya diseñado. Ha sido creado por la compañía Atmel, tiene un voltaje operativo de 5V, aunque se recomienda como entrada de 7-12V con un límite de 20V. Contiene 14 pines digitales de entrada y salida, 6 pines analógicos que están conectados directamente a los pines de

la placa Arduino comentados anteriormente. Dispone de 32KB de memoria flash (de los cuales 512 bytes son utilizados por el bootloader). En la memoria flash se instalará el programa a ejecutar. El bootloader será el encargado de preparar el microcontrolador para que pueda ejecutar nuestro programa. También tiene una memoria EEPROM de 1KB que puede ser leída o escrita con la librería EEPROM. En la parte de procesamiento dispone de un reloj de 16Mhz y 2KB de memoria RAM. [6]

2.3.1.7. Fuente de alimentación externa

Para alimentar a la placa también podemos usar el conector Jack de 3.5mm con corriente continua entre 7 y 12V

2.3.2. TEXAS INSTRUMENTS LAUNCHXL-F28069M

Es una placa de circuito impreso desarrollada por la empresa Texas Instruments. Este microcontrolador pertenece a la familia C2000 de TI. Los procesadores de esta familia están dedicados al control digital de la señal (DSC). Este dispositivo está diseñado para realizar las operaciones más generales que se usan cuando se trabaja con sensores digitales. Su objetivo es completar el proceso en el menor tiempo posible para que se pueda formar la salida con ese nuevo valor en tiempo real. Este tipo de procesadores es el indicado para sistemas en los que se requiere de control avanzado de tiempo real. Este nos permite el control de motores, almacenamiento de datos y sistemas de control digital con un coste mínimo. TMS320F28069M

Dentro de la familia de los C2000, tenemos 3 grupos de DSC's. Los llamados C24x que utilizan 16 bit, los C28x que usan 32 bit y el grupo que incluye memoria flash que son los F28xx.

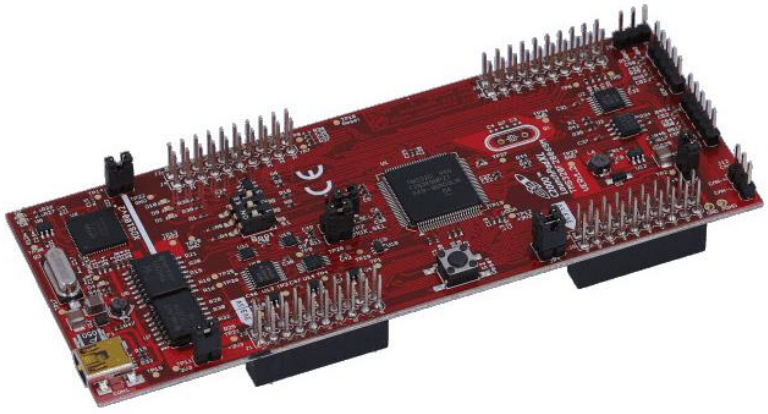

Ilustración 5: LAUNCHXL-F28069M

Las características de la placa de desarrollo son las que se reflejan en la siguiente tabla: [7]

Microprocesador	TMS320F28069MPZT
Velocidad	90MHz
Memoria Flash	256kB
RAM	96kB
EEPROM	N/A
TIMERS	3x32 bit
Comunicación Serial	2SPI, i2c, 2 UARTs, CAN
Canales ADC	12-bit 16 canales
BoosterPacks Pins	2x40
Largo	131mm
Ancho	51mm
Peso	40g

Tabla 2: Características LAUNCHXL-F28069M

Los principales componentes que forman el circuito son las que se muestran en la figura siguiente:

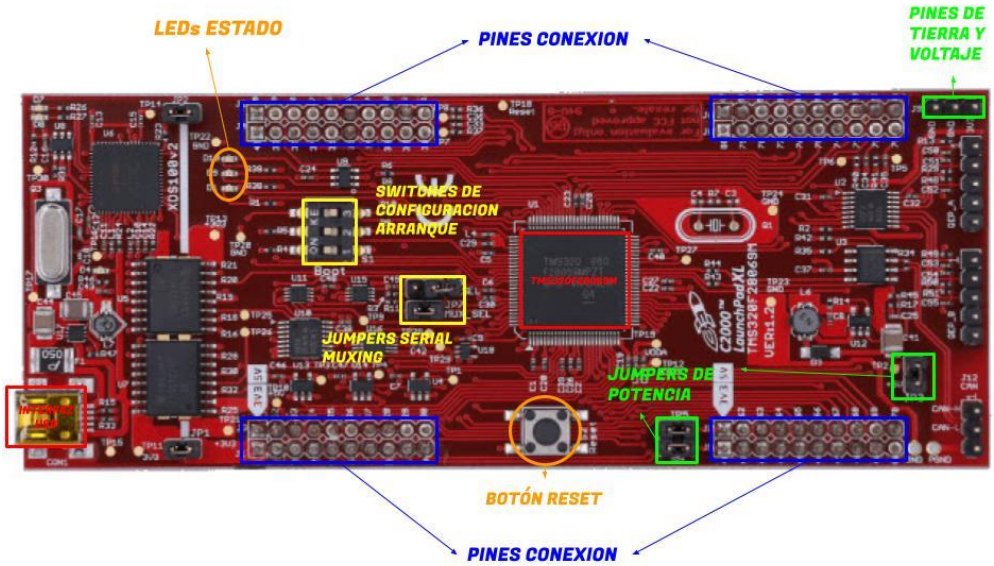

Ilustración 6: Ubicación de los diferentes componentes en la placa LAUNCHXL-F28069M

2.3.2.1. LEDs de Estado

Disponemos de dos LEDs de usuario que podemos programar y controlar, y de otro LED que nos indica el estado de encendido o apagado de la misma.

2.3.2.2. Pines de conexión

La placa dispone de un total de 40 pines de conexión, los cuales se encargan de controlar las distintas partes y funciones que nos brinda. Dentro del manual del propio microcontrolador se nos presentan las diferentes opciones en las que podemos configurar estos pines, así como para los tipos de interfaces que están implementados.

2.3.2.3. Pines de Tierra y voltaje

El circuito impreso dispone de 3 pines de alimentación principales, a los cuales podremos conectar los módulos a tierra, a 3.3V y a 5V. Estos pines son de conexión directa, por lo que no tendremos que activar ninguna opción vía software para que habilitarlos.

2.3.2.4. Botón Reset

Utilizando este botón podemos reiniciar la ejecución del código del microcontrolador

2.3.2.5. Jumpers de Potencia

Esta placa dispone de 5 Jumpers diferentes, que podemos habilitar, para poder elegir la configuración por donde pasa la energía. [7]

JUMPER	OPCION DE ENERGIA
JP1	Habilita 3.3 V desde USB (deshabilita el aislamiento)
JP2	Habilitar GND desde USB (deshabilita el aislamiento)
JP3	Habilite el conmutador de 5 V (desconectado el suministro de 3,3 V del dispositivo de destino)
JP4	Conecta la CPU de destino 3.3 V al segundo conjunto de BoosterPack encabezados
JP5	Conecta la MCU de 5 V de destino al segundo conjunto de encabezados BoosterPack

Tabla 3: Opciones de los Jumpers de energía

2.3.2.6. Interfaz USB

Sirve a su vez tanto de alimentación para la propia placa, como de puerto para la conexión e intercambio de datos con el ordenador, para poder crear y cargar los programas. Este incluye el método de depuración XDS100v2 de alta velocidad.

2.3.2.7. Switches de configuración de arranque

Se incluye en el microcontrolador una ROM de arranque, para que se realicen unas comprobaciones de inicio del dispositivo. Se incluye en la placa un Switch de arranque, con diferentes configuraciones posibles. Este Switch está formado por tres pines que combinando entre sus estados H(ON) y L(OFF) tendremos la posibilidad de escoger la configuración que más se ajuste a nuestras necesidades, esta se encuentra recogida en la tabla 4.

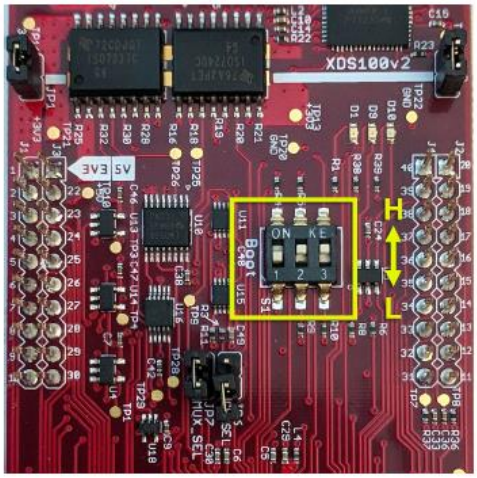

Ilustración 7: Opciones del Switch de arranque [7]

Modo Boot	S1-Switch 1 (GPIO34) H = Pulled to 1 L = Pulled to 0	S1-Switch 2 (GPIO37 / TDO) H = Pulled to 1 L = Pulled to 0	S1-Switch 3 (TRSTn) H = XDS100v2 L = Tied to 0
Modo Emulación	L	H	H
IO Paralelo	L	L	L
SCI	H	L	L
Wait	L	H	L
GetMode	H	H	L

Tabla 4: Opciones de configuración para el arranque [7]

2.3.2.8. Jumpers Serial Muxing

El propio circuito tiene integrado un adaptador USB a UART integrado, además contiene dos periféricos SCI, mientras que el Launchpad tiene tres lugares donde estos deben estar enrutados. Con todo esto, la placa incluye un multiplexor de conectividad serie para poder realizar esta configuración de SCI. En la tabla siguiente vemos sus funciones:

MUX_SEL (JP7)	CH_SEL (JP6)	FUNCION
ON	ON	USB / UART desactivado; J1.3 y J1.4 - GPIO28 y GPIO29; J7.3 y J7.4 - GPIO15 y GPIO58
ON	OFF	USB/UART – GPIO28 y GPIO29, J1.3 and J1.4 – Hi-Z; J7.3 y J7.4 – GPIO15 y GPIO58
OFF	ON	USB/UART – GPIO15 and GPIO58; FAULT/OCTW – GPIO28 and GPIO29; J7.3 and J7.4 – Hi-Z
OFF	OFF	USB/UART – GPIO15 y GPIO58; FAULT/OCTW – GPIO28 y GPIO29; J7.3 y J7.4 – Hi-Z

Tabla 5: Opciones de conectividad serial [7]

CAPITULO 3: METODOLOGIA

3.1. TECNOLOGIA BLUETOOTH

Bluetooth es un estándar desarrollado con la finalidad de lograr la comunicación de datos de corto alcance. Las principales características de esta tecnología son el bajo coste, consumo y complejidad. Hoy en día se encuentra integrada en la mayoría de los dispositivos cotidianos, con el fin de sustituir las conexiones cableadas sin perder seguridad ni fiabilidad.

En 2010, se introdujo la versión Bluetooth Low Energy, la que es la versión 4.0 de Bluetooth. Se pretende mejorar con esta versión algunos de los aspectos que tiene el Bluetooth Clásico. La mejora principal se encuentra en el consumo energético (de ahí su nombre)

Se utiliza como especificación industrial en redes WPAN o Wireless Personal Area Network, lo que permite la transmisión de datos en la banda de 2,4GHz

3.1.1. Historia

La tecnología bluetooth tiene su origen en la compañía Ericsson Mobile Communications, la cual inició una investigación en 1997 para lograr una interfaz de radio de baja potencia y bajo coste que pudiera ser usada entre teléfonos móviles y tarjetas de PC, a fin de eliminar los cables entre dispositivos. Viendo la gran posibilidad que sería la creación de un estándar de comunicaciones inalámbricas, contactaron con Intel para que se uniese al proyecto.

Un año más tarde, se crea el grupo de interés especial o SIG, formado por cinco grandes compañías (Ericsson, Intel, Nokia, IBM y Toshiba), oficializando como Bluetooth, el nombre de este nuevo estándar.

3.1.2. Tipos de Bluetooth y versiones [8]

Todos los dispositivos Bluetooth están formados por dos piezas imprescindibles, el primero la radio que se encarga de transmitir la señal, y el segundo la CPU que se ocupa de procesar las señales que le llegan.

Podemos clasificar, en base a la potencia de transmisión, en cuatro clases a los dispositivos Bluetooth

- **Clase 1:** conectividad hasta 100m y potencia de 100mW
- **Clase 2:** rango conectividad hasta 20m y potencia de 2.5mW
- **Clase 3:** alcance hasta 1m y potencia de 1mW
- **Clase 4:** rango máximo de 50cm y potencia de 0.5mW

A lo largo del desarrollo de la tecnología se han pasado por numerosas versiones en las que se han mejora e incluido nuevos aspectos, para dotar al estándar de mejores especificaciones.

Todas las versiones de los diferentes estándares Bluetooth, tiene capacidad de retrocompatibilidad, por lo que el ultimo estándar cubre todas las versiones anteriores.

Versión del Estándar	Fecha Adopción	Nota de la Versión
1.0–1.0a-1.0b-1-1	Julio 1999 – Febrero 2001	Progreso desde la primera versión borrador hasta la incorporada en el IEEE 802.15.1
1.2	Noviembre 2003	Añadidas mejoras para la transmisión de voz
2.0+EDR	Noviembre 2004	Gracias al EDR se aumenta la velocidad de flujo de datos a 3Mbps
2.1+EDR	Julio 2007	Añade opción de emparejamiento simple seguro
3.0+HS	Abril 2009	Incluye nuevo canal de transmisión para incrementar la velocidad de flujo de datos a 10Mbps
4.0	Junio 2010	Incluyen estándar de BLE y ATT y GATT en lugar de EDR
4.1	Diciembre 2013	Actualización de la arquitectura para incluir en el futuro nuevas funcionalidades
5.0	Mitad 2016	Versión orientada para el Internet de las Cosas (IOT) incluye SAM para prevenir y detectar interferencias. Doble de ancho y 4 veces de alcance más que la versión anterior
5.1	Enero 2019	Añade funcionalidad para conocer la ubicación a los que estén conectados (No tan precisa como GPS)
5.2	Enero 2020	Mejoras en BLE. Nuevo perfil EATT. Permite enviar audio sincronizado a varios dispositivos. Aumenta seguridad de la conexión

Tabla 6: Detalle de las versiones de Bluetooth

3.2. MODULO BLUETOOTH HC-06

El módulo HC-06 contiene la tecnología Bluetooth SPP (protocolo de puerto serie), la cual está diseñada para establecer conexiones inalámbricas a través del puerto serial del dispositivo al que lo conectemos y con ello poder hacerle capaz de comunicarse mediante Bluetooth.

La ventaja que nos puede dar el utilizar este módulo, es que la tecnología Bluetooth se encentra de forma nativa en la mayoría de los dispositivos que usamos día a día como el PC, Smartphones y Tablet. Su uso es, además, independiente del sistema operativo, por lo que su compatibilidad la hace completa con cualquier plataforma en la que queramos desarrollar nuestro trabajo o proyecto.

Ilustración 8: Modulo Bluetooth HC-06

El módulo utilizado para este trabajo es de la compañía DSD TECH. Las características de este módulo son:

Bluetooth	V2.0+EDR
Voltaje de trabajo	3,3-6 V
Rango profundidad modulación	2Mbps - 3Mbps.
Corriente de Emparejamiento	30~40 mA
Corriente en Comunicación	8mA
Rango temperatura almacenamiento	-40ºC ~ +85ºC
Rango de temperatura de trabajo	-25ºC ~ +75ºC
Potencia de emisión	3dBm

Tabla 7: Características módulo HC-06

El módulo HC-06 dispone de 4 pines para su conexión y de un led que parpadea si no se encuentra emparejad y que se queda fijo una vez que si lo está.

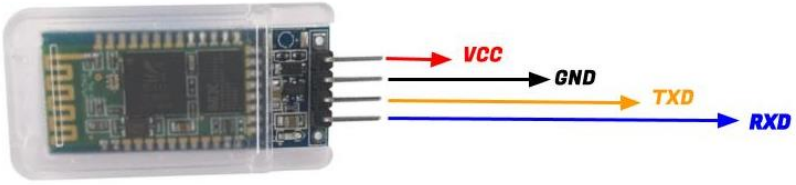

Ilustración 9: Pines de conexión modulo HC-06

- **VCC**: pin de alimentación del dispositivo
- **GN:** pin de conexión a tierra
- **TXD**: pin de transmisión de datos
- **RXD**: pin de recepción de datos

De fabrica viene configurado para actuar como esclavo, es decir, se mantiene a la espera de recibir la orden por parte del maestro. Además, dispone de un reducido número de instrucciones. Su conexión es sencilla a través de comandos AT con un puerto serie.

Por defecto los parámetros son:

- **BaudRate**: 9600N81
- **Contraseña**: 1234

Lista de comandos AT: [9]

COMANDO	INFORMACION	RESPUESTA
AT	Prueba de comunicación (cada segundo)	OK
AT+NAMEname	Resetear nombre del Bluetooth	OKname
AT+PINxxxx	Cambiar PIN vinculación, xxxx deben ser 4 números	OKsetPIN
AT+BAUDX	Configuración velocidad (Baudrate) (ver tabla siguiente)	OKX
AT+PN	Eliminar verificación de paridad	OK NONE
AT+PO	Establecer comprobación de paridad impar	OK ODD
AT+PE	Establecer comprobación de paridad par	OK EVEN

Tabla 8: Lista de comando para configurar el módulo HC-06

COMANDO	INFORMACION	RESPUESTA
AT+BAUD1	Velocidad (baudrate) de 1200	OK1200
AT+BAUD2	Velocidad (baudrate) de 2400	OK2400
AT+BAUD3	Velocidad (baudrate) de 4800	OK4800
AT+BAUD4	Velocidad (baudrate) de 9600 (por defecto)	OK9600
AT+BAUD5	Velocidad (baudrate) de 19200	OK19200
AT+BAUD6	Velocidad (baudrate) de 38400	OK38400
AT+BAUD7	Velocidad (baudrate) de 57600	OK57600
AT+BAUD8	Velocidad (baudrate) de 115200	OK115200

Tabla 9: Comandos para configurar la velocidad de transmisión del módulo HC-06

3.3. DISEÑO DEL PROTOCOLO DE COMUNICACION

Para poder realizar un correcto de diseño del protocolo de comunicación tendremos que conocer, como se comunican los dispositivos con los que vamos a trabajar.

3.3.1. Protocolo serial Asíncrono (UART)

La comunicación entre el módulo Bluetooth y los microcontroladores va a ser cableada, por lo que el protocolo que va a regir esta comunicación va a ser el protocolo serial asíncrono. Vamos a ver en que consiste:

El protocolo serial asíncrono es un protocolo de comunicación común en el mundo de la electrónica, utilizado para comunicar información entre varios dispositivos. Este protocolo se implementa con los receptores trasmisores asíncronos universales UART. Cuando un dispositivo utiliza este protocolo serial, su UART transmite en la línea “TX” y recibe datos por la línea “RX”

Para poder conectar dos dispositivos usando este método, uno de los dos envía un carácter por su línea TX y el otro la recibe por su línea RX y viceversa. Para realizar la conexión correcta de dos dispositivos se sigue el esquema de la figura 10.

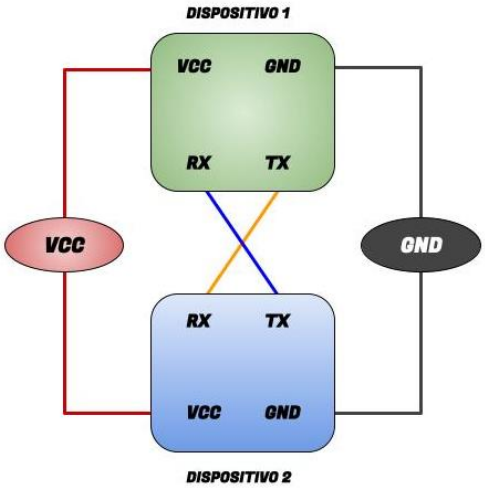

Ilustración 10: Esquema de conexión de dos dispositivos para su comunicación serial asíncrona

Lo más importante a destacar y tener en cuenta es que el pin TX del dispositivo 1 está conectado el pin RX del dispositivo 2. De modo que el dispositivo 1 envía datos por el pin TX mientras que el dispositivo 2 los recibe por el pin RX.

En UART, solo uno de los dispositivos puede tener el control de la línea en un determinado momento. Si más de un dispositivo transmite a la línea RX de otro dispositivo, se produce un error llamado “contención de bus”

3.3.2. Diseño y reglas del protocolo

Además de los dispositivos principales de la comunicación vamos a disponer de entidades que son los elementos que en un equipo realizan parte de las funciones correspondientes al nivel de arquitectura al que pertenecen.

Las entidades van a proporcionar los servicios de comunicación a los usuarios, están deben de permitir iniciar la comunicación, finalizarla y poder enviar y recibir mensajes. Van a trabajar en el nivel de enlace.

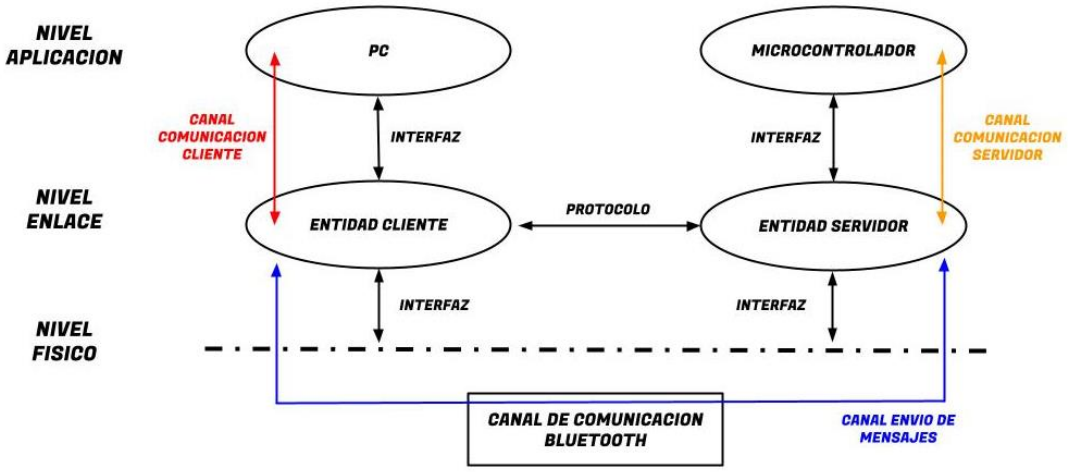

Ilustración 11: Esquema de comunicación entre los elementos

La comunicación va a ser de tipo cliente-servidor. De este modo tendremos una comunicación bidireccional entre los dispositivos. A nivel de enlace la interacción entre las entidades va a seguir la regla de parada y espera.

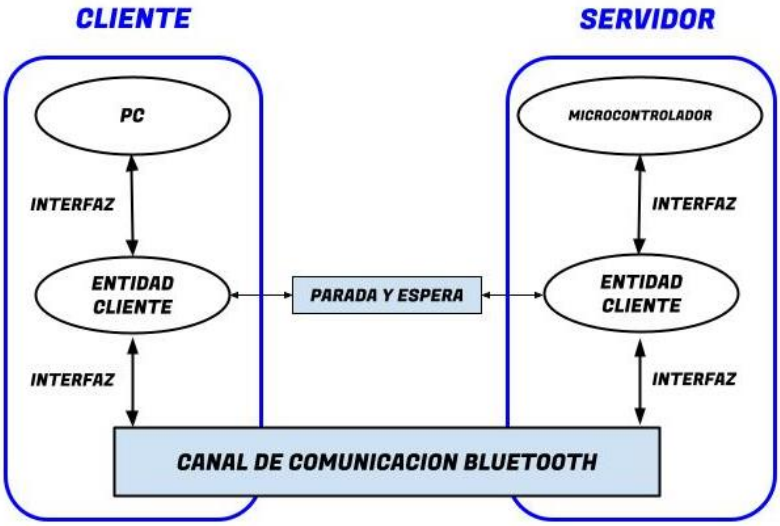

Ilustración 12: Esquema de comunicación completo con los protocolos implementados

El cliente es el que inicia y dirige la comunicación, espera y recibe las respuestas del servidor. En nuestro caso el dispositivo que actúa como cliente es el PC, y los servidores van a ser los microcontroladores. Utilizando esta arquitectura podríamos conectarnos a ambos servidores al mismo tiempo, pero por el uso final que lo vamos a dar, el cliente solo se va a conectar a uno de ellos durante la comunicación. Para interactuar con el usuario final se va a disponer de una interfaz gráfica.

Las reglas que va a definir el protocolo de comunicación son las siguientes:

- Por cada petición realizada por parte del PC se va a esperar la respuesta por parte del microcontrolador conectado. Una vez que se envía la petición por parte del PC, que es el encargado de iniciar y dirigir la comunicación, no se envía la siguiente hasta que no se recibe la respuesta de confirmación.
- En el que caso de que se produzca un error, debido a perdida de información o alguna interferencia en el dispositivo bluetooth durante la comunicación, se producirán 3 reintentos de envío del mensaje. Si tras el 3 envío no se logra enviar el mensaje correctamente pondremos fin a la comunicación.
- Para el envío y recepción de los mensajes vamos a suponer que el usuario puede introducir a través de la interfaz valores incorrectos, pero que no serán admitidos, es decir, se notificará por pantalla que no admitiremos texto, valores negativos, ni valores que superen el rango máximo del vástago del actuador que será de 0-30mm.
- Del lado de los microcontroladores, vamos a suponer que siempre se introducen los valores correctos, es decir valores numéricos entre 0-30mm
- Vamos a trabajar con la premisa de que los mensajes de petición-respuesta y su contenido siempre es correcto, es decir, los casos admitidos como error van a ser aquellos que se produzcan por algún condicionante externo como interferencias. En cuyo caso al no recibir el mensaje correctamente por alguna de las dos partes, se reenviará de nuevo el último mensaje enviado, ya sea de petición o de respuesta. De igual modo ser realizará la comprobación en ambos dispositivos de que tanto los mensajes como su contenido son los adecuados y correspondientes.

Durante la recepción y el envío de los mensajes se pueden dar varios eventos diferentes para tener en cuenta:

1. Envío de petición con respuesta correcta, completando la comunicación

   Se realiza una petición por parte del PC hacia el microcontrolador, el mensaje es recibido correctamente en el microcontrolador. Se procesa el mensaje y se genera la respuesta correspondiente a la petición recibida, que tras analizarla el cliente es la esperada.

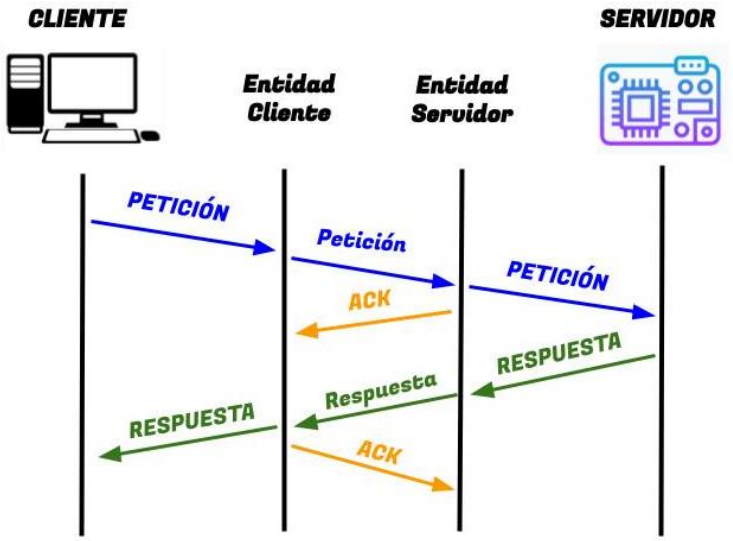

Ilustración 13: Esquema de envío de petición con respuesta correcta, completando la comunicación

2. **Envío de petición con respuesta de error, llegando al número máximo de intentos de reenvío, poniendo fin a la comunicación**

Se inicia una petición por parte del PC. El mensaje recibido no es correcto debido a que se ha producido una pérdida de información durante la transmisión. El servidor envía mensaje de error al cliente para notificarlo. El cliente al recibir el error reenvía la petición de nuevo. Se produce la misma secuencia 3 veces por lo que se pone fin a la comunicación sin lograr completar el envío.

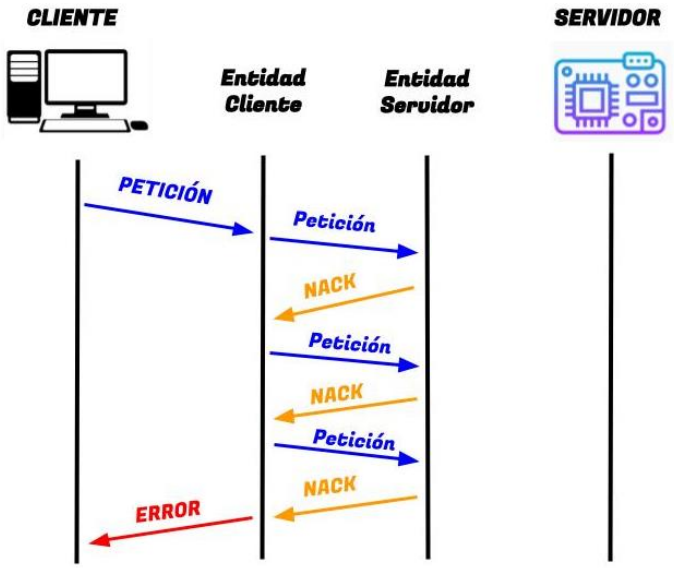

Ilustración 14: Esquema de envío de petición con respuesta de error, llegando al número máximo de intentos de reenvío, poniendo fin a la comunicación

3. **Envío de petición con respuesta, pero esta es errónea, llegando al número máximo de intentos de reenvío, poniendo fin a la comunicación**

Se inicia una petición por parte del PC. El mensaje recibido en servidor es correcto por lo que se envía la respuesta. La respuesta recibida por el cliente es errónea por lo que se notifica el error. El servidor al recibir el mensaje de error procede al reenvío de la respuesta a la petición primera. Se produce la misma secuencia 3 veces por lo que se pone fin a la comunicación sin lograr completar el envío de la información.

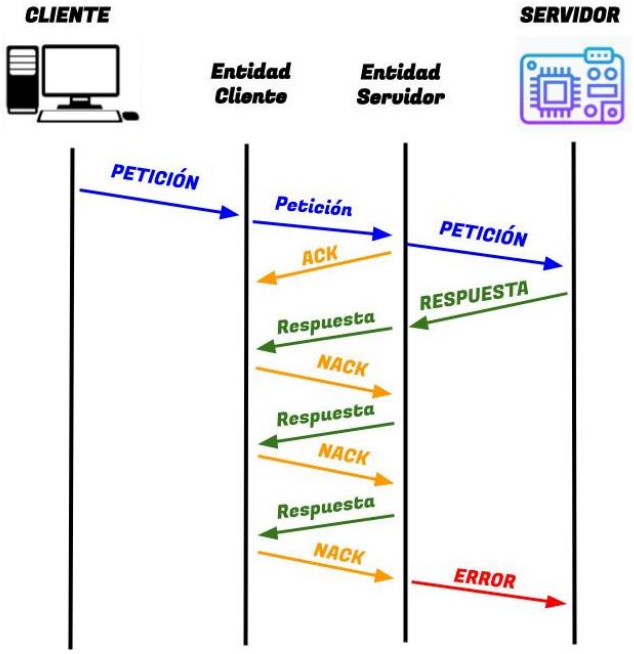

Ilustración 15: Esquema de envío de petición con respuesta, pero esta es errónea, llegando al número máximo de intentos de reenvío, poniendo fin a la comunicación

4. **Envío de petición con respuesta de error, sin llegar al número máximo de intentos de reenvío, completando la comunicación**

Se inicia una petición por parte del PC. El mensaje recibido no es correcto debido a que se ha producido una pérdida de información durante la transmisión. El servidor envía mensaje de error al cliente para notificarlo. El cliente al recibir el error reenvía la petición de nuevo. Se produce la misma secuencia 2 veces, pero al tercer intento se logra enviar la respuesta por parte del servidor, por lo que finalmente se logra completar la comunicación.

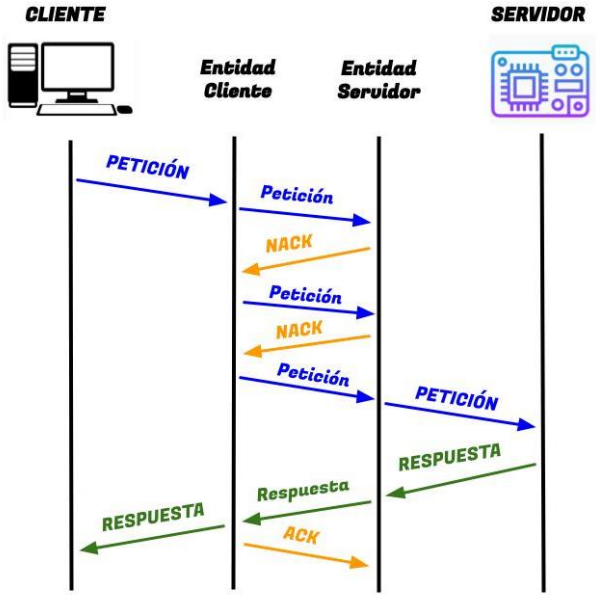

Ilustración 16: Esquema de envío de petición con respuesta de error, sin llegar al número máximo de intentos de reenvío, completando la comunicación

5. **Envío de petición con respuesta, pero esta es errónea, sin llegar al número máximo de intentos de reenvío, completando la comunicación**
   Se inicia la comunicación con la petición por parte del cliente, el servidor procesa el mensaje recibido y manda la respuesta. El cliente detecta un error y lo notifica, al recibir el mensaje de error el servidor reenvía la respuesta que recibe correctamente el cliente por lo que se logra completar la comunicación.

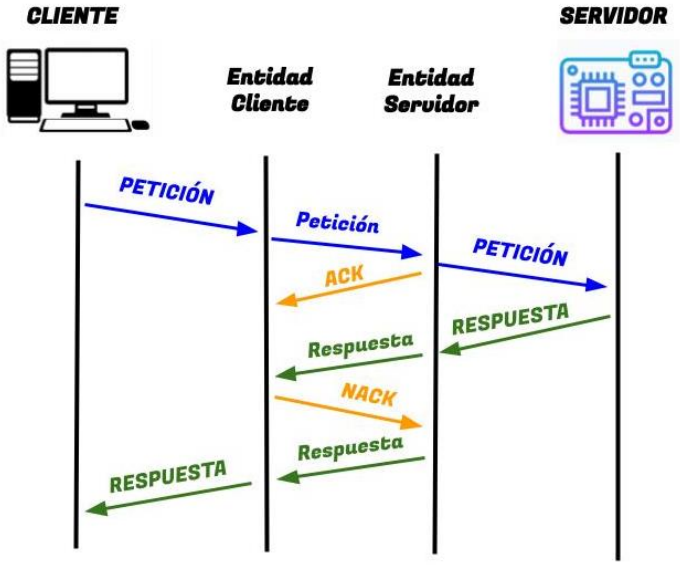

Ilustración 17: Esquema de envío de petición con respuesta, pero esta es errónea, sin llegar al número máximo de intentos de reenvío, completando la comunicación

3.3.3. Formato de las tramas intercambiados

Los diferentes tipos de mensajes que se van a intercambiar van a seguir una estructura fija compuesta por bytes. Estos bytes van a ser identificativos únicamente para nuestro protocolo.

El formato de la descripción de los campos que componen las tramas se describe en la siguiente tabla:

CAMPO	FORMATO	DESCRIPCION
CABECERA	BYTE	Indica comienzo de trama. Byte fijo: **1B**
TAMAÑO DE TRAMA	BYTE	Indica la longitud de la trama en bytes: 6+Tamaño dato
COD_INST	BYTE	Código de instrucción
VAR_ID	BYTE	Etiqueta identificativa para el tipo de dato
DATO	BYTE	Datos a enviar
COD_ERROR	BYTE	Código de Error
CHECK_SUM	BYTE	Valor de la suma de todos los bits
FIN	BYTE	Indica finalización de trama. Byte fijo: **1D**

Tabla 10: Campos que componen las tramas

Vamos a definir que posibles valores pueden tener los distintos campos mencionados:

3.3.3.1. COD_INST

El código de instrucción (COD INST) va a ser diferente, dependiendo del tipo de mensaje que utilice en la comunicación. Para ello se ha asignado un valor para cada una de las tramas, los cuales se presentan en la siguiente tabla:

TRAMA	CODIGO INSTRUCCIÓN	DESCRIPCION
Petición Lectura	0x01	Solicitud para enviar datos de Microcontrolador→PC
Respuesta Lectura	0x11	Respuesta a Petición Lectura
Petición Escritura	0x02	Solicitud para enviar datos de PC→Microcontrolador
Respuesta Escritura	0x22	Respuesta a Petición Escritura
Error	0xFF	Informe de fallo

Tabla 11: Definición del campo COD_INST

3.3.3.2. VAR_ID

El campo de las variables (VAR_ID) se compone de cinco tipos de variables dependiendo del tipo de dato (posición del actuador) Este se asocia a cada actuador. Es el byte que marca el comienzo de la transmisión del dato de posición del actuador. Las distintas variables se presentan en la siguiente tabla:

VARIABLE	VAR_ID	DESCRIPCION
TODOS	0Xff	Se envían todos los datos posibles
Posición Actuador 1	0x00	Posición actuador 1
Posición Actuador 2	0x01	Posición actuador 2
Posición Actuador 3	0x02	Posición actuador 3
Posición Actuador 4	0x03	Posición actuador 4
Posición Actuador 5	0x04	Posición actuador 5

Tabla 12: Variables del campo VAR_ID

3.3.3.3. ERROR

Como código de Error, vamos a tener 2 estados distinguiendo cuando alguno de los campos de las tramas es incorrecto y otro para cuando no hay error, como vemos en la tabla siguiente:

COD_ERROR	DESCRIPCION
0x00	No hay error
0xF0	Trama invalida

Tabla 13: Códigos de Error

3.3.3.4. CHECKSUM

El Checksum o también llamado suma de verificación es una función sencilla que se utiliza para garantizar la integridad de una serie de datos. Con ello se pretende encontrar cambios accidentales en una secuencia de datos y con ellos poder protegerlos. La forma de utilizarlo correctamente es utilizarlo dentro del mensaje como un campo más en el que se sumen todos los valores anteriores a él (a excepción del campo de cabecera), y realizar este cálculo tanto desde el lado de la transmisión como de la recepción.

3.3.4. Transmisión de datos de PC a Microcontrolador

Se quiere enviar los datos de posición de todos los actuadores del PC al Microcontrolador (igual para Arduino que para Texas). Para ello, introducimos los datos de posición al ordenador (mediante la aplicación de Visual Studio) y damos al botón enviar.

A nivel interno, lo que sucede en el proceso es lo siguiente:

- PC manda una petición de escritura al Microcontrolador con los datos de posición que hemos introducido por pantalla
- El Microcontrolador recibe los datos y verifica que los datos recibidos son correctos. Posteriormente, manda un mensaje de respuesta de escritura al Arduino (si los datos han sido recibidos correctamente) o un mensaje de error.

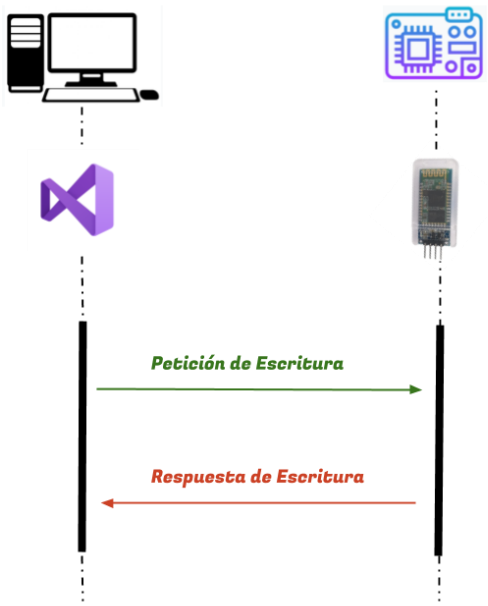

Ilustración 18: Transmisión de datos de PC a microcontrolador

3.3.4.1. PETICION DE ESCRITURA

Una vez que a través de la interfaz de usuario se han introducido los datos a enviar, y el usuario así lo ha indicado, se procese a enviar hacia el microcontrolador el mensaje Petición de Escritura.

Los campos que componen la trama son:

CABECERA	TAMAÑO	COD_INST	VAR_ID (1)	DATO (1)	...
BYTE	BYTE	BYTE	BYTE	4BYTE	...
1B	1F	02	XX	XX	...

...	VAR_ID (5)	DATO (5)	COD_ERROR	CS	FIN
...	BYTE	4BYTE	BYTE	BYTE	BYTE
...	XX	XX	00	XX	1D

Tabla 14: Formato de la trama Petición de Escritura

La trama petición de escritura está formada por un total de 31 campos (byte). Al igual que el resto de las tramas los bytes de inicio (cabecera) y de fin son compartidos, el resto de los campos son únicos para este tipo de mensaje.

Los campos DATO (X), que como vemos reflejado en la tabla está formado por 4 bytes, corresponde al valor descompuesto en 4 bytes del valor (en float32) que ha introducido el usuario. Los valores se explicarán más adelante en los apartados obtenido como se logra descomponer un valor *float* a su equivalente array de 4 bytes.

VAR_ID (ACTUADOR 1)	DATO (1)	DATO (2)	DATO (3)	DATO (4)
BYTE	BYTE	BYTE	BYTE	BYTE
00	XX	XX	XX	XX

Tabla 15: Parte del mensaje correspondiente a cada actuador

3.3.4.2. RESPUESTA DE ESCRITURA

Una vez el microcontrolador ha recibido y comprobado que los campos del mensaje Petición de Escritura son los esperados, este procede a crear y enviar la respuesta al PC de que los datos.

Los campos que componen la trama son:

CABECERA	TAMAÑO	COD_INST	VAR_ID (1)	...	VAR_ID (5)	COD_ERROR	CS	FIN
BYTE	BYTE	BYTE	BYTE	...	BYTE	BYTE	BYTE	BYTE
1B	0B	22	XX	...	XX	00	XX	1D

Tabla 16: Formato de la trama Respuesta de Escritura

3.3.5. Transmisión de datos de Microcontrolador a PC

El ordenador desea conocer los datos de posición del Microcontrolador (igual para Arduino y Texas) Cuando el microcontrolador recibe esta petición de lectura de datos, enviará dichos datos al pc (estos datos se introducirán mediante el ordenador, en concreto, desde el monitor serie de la IDE de Arduino o desde el monitor serie que nos proporciona de Texas, Code Composer Studio).

Internamente, lo que pasa es lo siguiente:

- PC manda una petición de lectura
- El Microcontrolador recibe la petición de lectura. Si el mensaje se ha recibido correctamente, el Microcontrolador manda un mensaje de respuesta de lectura al PC con los datos de posición.

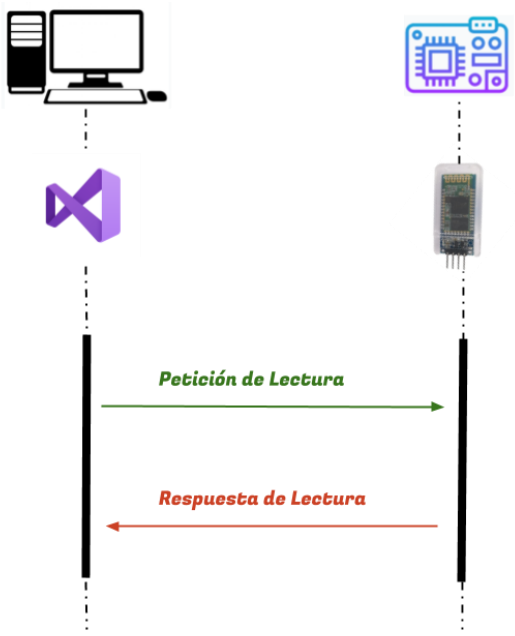

Ilustración 19: Transmisión de datos de Microcontrolador a PC

3.3.5.1. PETICION DE LECTURA

Cuando el usuario decide solicitar los datos al microcontrolador se produce la Petición de Lectura.

Los campos que componen la trama son:

CABECERA	TAMAÑO	COD_INST	VAR_ID (1)	...	VAR_ID (5)	COD_ERROR	CS	FIN
BYTE	BYTE	BYTE	BYTE	...	BYTE	BYTE	BYTE	BYTE
1B	0B	01	XX	...	XX	00	XX	1D

Tabla 17: Formato de la trama Petición de Lectura

3.3.5.2. RESPUESTA DE LECTURA

Cuando el microcontrolador recibe el mensaje de Petición de Lectura, comprueba que estos son los adecuados y procede a crear el mensaje de respuesta.

Los campos que componen la trama son:

CABECERA	TAMAÑO	COD_INST	VAR_ID (1)	DATO (1)	...
BYTE	BYTE	BYTE	BYTE	4BYTE	...
1B	1F	11	XX	XX	...

...	VAR_ID (5)	DATO (5)	COD_ERROR	CS	FIN
...	BYTE	4BYTE	BYTE	BYTE	BYTE
...	XX	XX	00	XX	1D

Tabla 18: Formato de la trama Respuesta de Lectura

Como vemos, los campo DATO (X), estará formado por los 4 bytes resultado de descomponer el valor *float* de cada actuador que se ha recogido desde el monitor serial del microcontrolador. El proceso exacto de cómo se convierte el valor *float* a un array de 4 bytes, se detalla más adelante.

Con lo que el formato del mensaje, usando como ejemplo al valor de pulgar quedaría:

VAR_ID (PULGAR)	DATO (1)	DATO (1)	DATO (1)	DATO (1)
BYTE	BYTE	BYTE	BYTE	BYTE
00	XX	XX	XX	XX

Tabla 19: Desglose de los campos para enviar las variables de los actuadores

3.3.6. Transmisión de datos errónea

Cuando algunos de los campos de los mensajes recibidos no es el esperado, o bien cuando el mensaje que se espera recibir no es el deseado, se genera el mensaje de error, el cual se reenviara el número de veces establecido para notificarlo. Está formado por los siguientes campos:

CABECERA	TAMAÑO	COD_INST	COD_ERROR	CS	FIN
BYTE	BYTE	BYTE	BYTE	BYTE	BYTE
1B	06	FF	F0	XX	1D

Tabla 20: Formato de la trama Error

CAPITULO 4: DESARROLLO

4.1. Creación de la interfaz en Visual Studio

En este apartado se describe de forma funcional cómo se comporta la aplicación y como se ejecuta el programa, pasando por todas las funciones encargadas de la correcta ejecución del programa.

4.1.1. Lenguaje de programación en Visual Studio

Visual Studio utiliza un lenguaje de programación basado en C#. Este lenguaje fue desarrollado e impulsado por Microsoft, mezcla las características más básicas del lenguaje C++ simplificándolos al estilo Java y ofreciendo un entorno de trabajo. C# forma parte la plataforma .NET, las principales ventajas de usarlo es que unifica los modelos de programación, simplifica el desarrollo del programa, ofrecer una gran interoperabilidad con código existente y simplifica la instalación y administración de las aplicaciones siendo extensible. [10]

4.1.2. Diagrama de flujo

El funcionamiento general de la aplicación se describe en la ilustración 20:

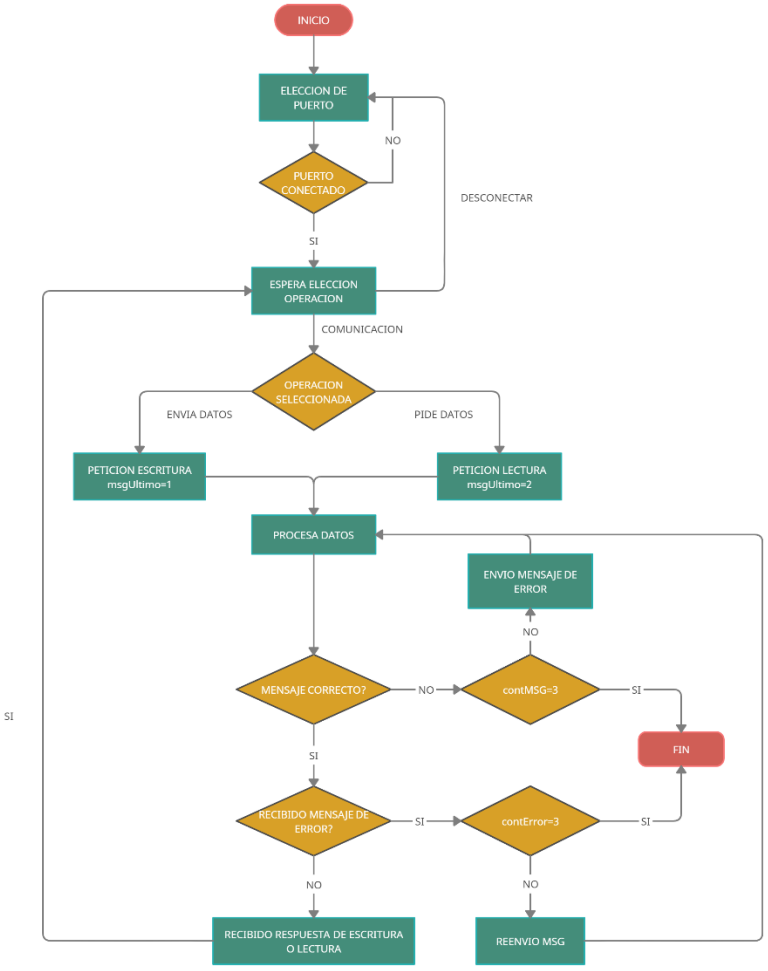

Ilustración 20: Diagrama de flujo de la aplicación en Visual Studio

La aplicación en su inicio se encuentra a la espera de que se seleccione el puerto al que conectarse para poder iniciar la comunicación. Una vez el usuario seleccione el puerto y la velocidad de transmisión, se procede a intentar la conexión con el módulo bluetooth.

Una vez establecida la conexión, la aplicación queda a la espera de que se seleccione la operación. En el caso de quiere enviar datos se pasará al estado de Petición de Escritura, donde se recogen los datos que el usuario introduce en la aplicación para crear el mensaje y enviarlo hacia el microcontrolador. Sin embargo, si se elige la opción de pedir datos, el programa envía el mensaje de petición de lectura automáticamente.

Si se recibe el mensaje de Respuesta de Escritura, la ejecución del programa quedara a elección del usuario, le cual puede realizar una petición nueva, o finalizar la conexión.

Si en vez de recibir alguno de los mensajes de respuesta, se recibe un mensaje de Error, esto significara que el mensaje que hemos enviado al microcontrolador es incorrecto o se ha producido una pérdida de información, en ese caso se procederá con el reenvío del mensaje, hasta un máximo de 3 veces, llegado a ese máximo, se pondrá fin a la ejecución del programa.

Cuando el mensaje que se recibe es correcto, pero alguno de los campos que lo componen no lo es, o el mensaje recibido no es el esperado, se generara el un mensaje de error que se enviara al microcontrolador, para notificar el error.

4.1.3. Interfaz desarrollada

A fin de utilizar la misma aplicación para ambos microcontroladores, se ha diseñado la interfaz, de forma compatible para ambas plataformas, es decir, no vamos a tener que seleccionar con que microcontrolador vamos a trabajar sino el puerto COM al que tenemos conectado el módulo bluetooth.

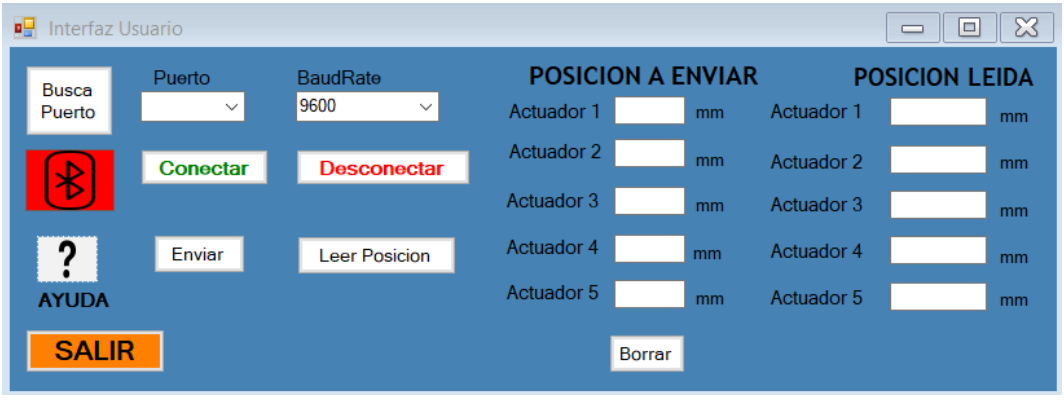

Ilustración 21: Interfaz de usuario desarrollada en Visual Studio

Una vez vista el aspecto general de la aplicación creada vamos a explicar sus funciones:

4.1.3.1. Busca Puertos

El botón busca puertos, está asociado a la función que busca los puertos disponibles dentro del PC. Esta hace un chequeo tanto de los puertos USB como de los bluetooth a los que se puede conectar. Se encuentra asociada al desplegable de *Puerto*, y colocara en él, los puertos que haya obtenido tras la búsqueda. Una vez cargados en el menú desplegable ya podremos seleccionar el puerto al cual queremos establecer conexión.

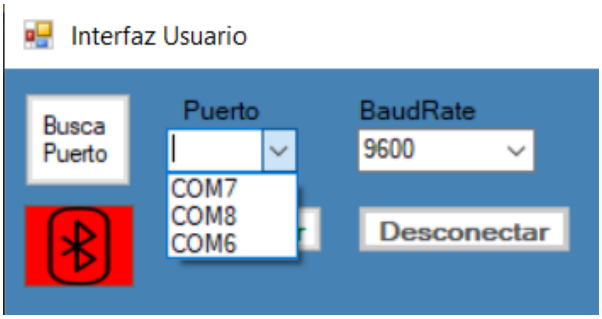

Ilustración 22: Ejemplo de ejecución de la opción busca puertos

4.1.3.2. Baudrate

En este desplegable podremos elegir la velocidad del puerto. Por defecto este preseleccionado el valor más típico 9600, pero también se la da la opción de configurarlo con 115200 o de incluso poder añadir más. Para los requisitos de la aplicación y las características del módulo, usamos 9600.

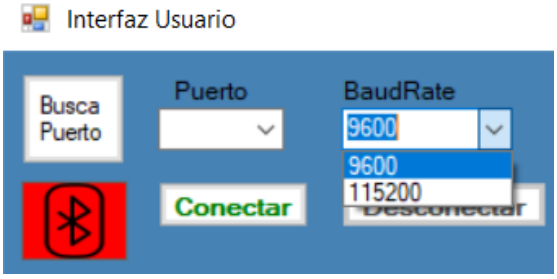

Ilustración 23: Opciones del desplegable Baudrate

4.1.3.3. Conexión/desconexión

Estos botones son los que nos permiten el control de la comunicación.

El botón de conectar va a llamar a la rutina que crea el puerto según los datos recogidos en desplegables de *Puerto* y *Baudrate.*

En el caso de que alguno de los que se encontrase vacío sin escoger ningún valor por el usuario se nos notificaría por pantalla (Ilustración 24) al pulsar el botón de conectar.

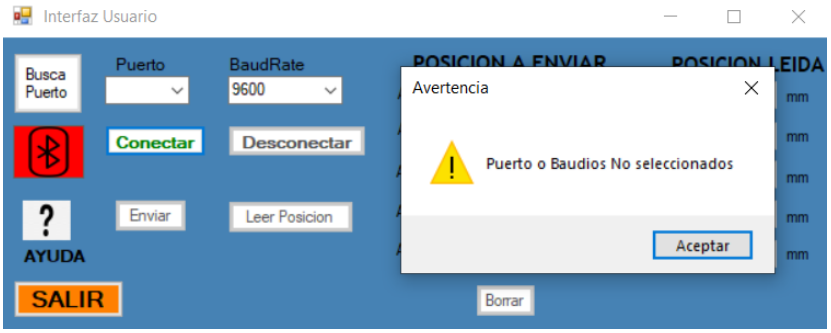

Ilustración 24: Notificación de opción de conexión no seleccionada

Teniendo el nombre y la velocidad del puerto tendremos que configurar el resto de los campos, según las reglas del protocolo. Este va a ser del tipo 8N1, es decir, 8 bits de datos, sin paridad y con un bit de parada.

Después se ejecutará la opción de abrir el puerto para establecer conexión, si no se consigue abrir el puerto se muestras el mensaje de "Error al Abrir el puerto" y podremos tras ello volver a intentar establecer conexión. Cuando se logre establecer conexión, el icono del bluetooth de la interfaz cambiara a color verde, para indicarnos que la conexión se ha realizado correctamente.

La función del botón desconectar es cerrar la comunicación con el módulo bluetooth. Además, reinicia la interfaz al estado inicial limpiando los datos recibidos o enviados de sus casillas correspondientes y liberando los recursos usados por el sistema. Llegados a este punto podríamos volver a establecer conexión y empezar con una nueva comunicación.

4.1.3.4. Enviar

El botón enviar se encuentra asociado con varias funciones complementarias. A través de él, vamos a recoger los datos que el usuario ha introducido previamente en los cuadros correspondiente a la zona de *Posición a Enviar.*

En primer lugar, se comprueban la integridad de los datos que ha introducido el usuario a través de una función externa, esta comprueba que los valores que ha introducido el usuario se encuentran el rango permitido por los vástagos de los motores, que es 0-30mm. Además de esto, en los propios cuadros de texto se han limitado los caracteres que se puede introducir por teclado, permitiendo solo valores numéricos y el signo '.' para cuando se requiera de introducir valores decimales. Si los valores de los actuadores exceden los márgenes establecidos, se notifica con un mensaje (Ilustraciones 25 y 26) y automáticamente se procede al borrado de dicho valor en su campo, para que se pueda volver a introducir uno correcto (Ilustración 27)

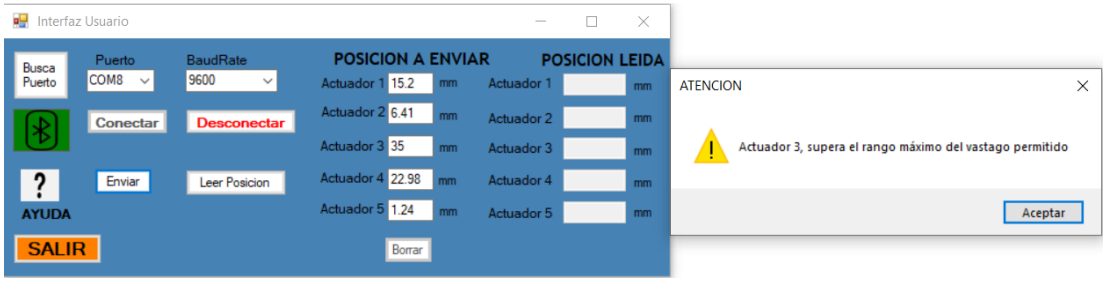

Ilustración 25: Notificación de rango máximo superado

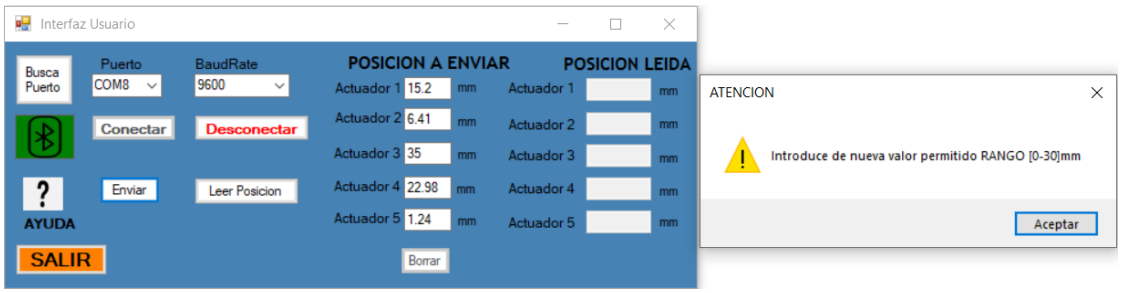

Ilustración 26: Notificación con el rango permitido

Ilustración 27: Limpieza del campo erróneo

Una vez los datos introducidos son correctos, se proceder a iniciar el envío del mensaje *Petición de Escritura*, su funcionamiento lo veremos en el siguiente apartado más en profundidad.

4.1.3.5. Leer posición

Es el botón asociado a la rutina que realiza la Petición de Lectura al microcontrolador. La creación y funcionamiento de este mensaje lo veremos en el siguiente apartado.

4.1.3.6. Ayuda

Se ha creado un cuadro de ayuda para que el usuario pueda obtener información sobre las funciones que realizan los diferentes botones que tiene disponibles en la interfaz.

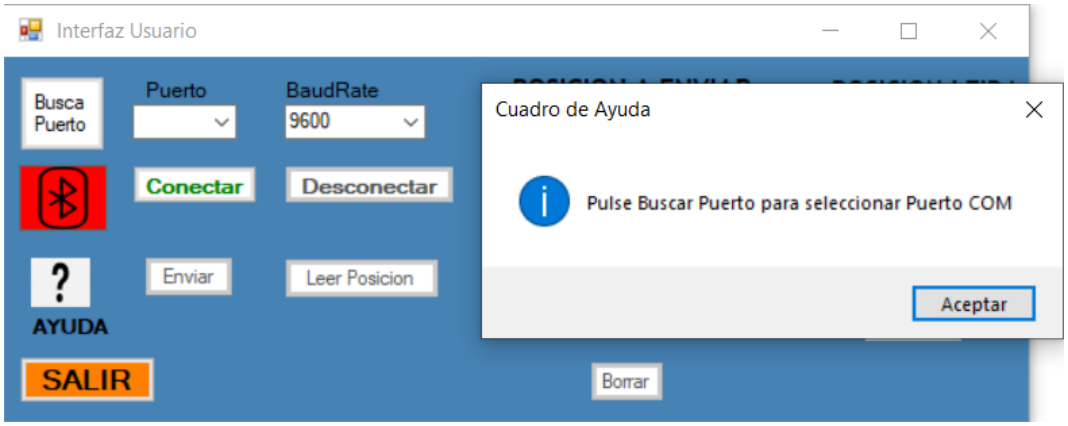

Ilustración 28: Cuadro de ayuda asociado al botón de Busca Puerto

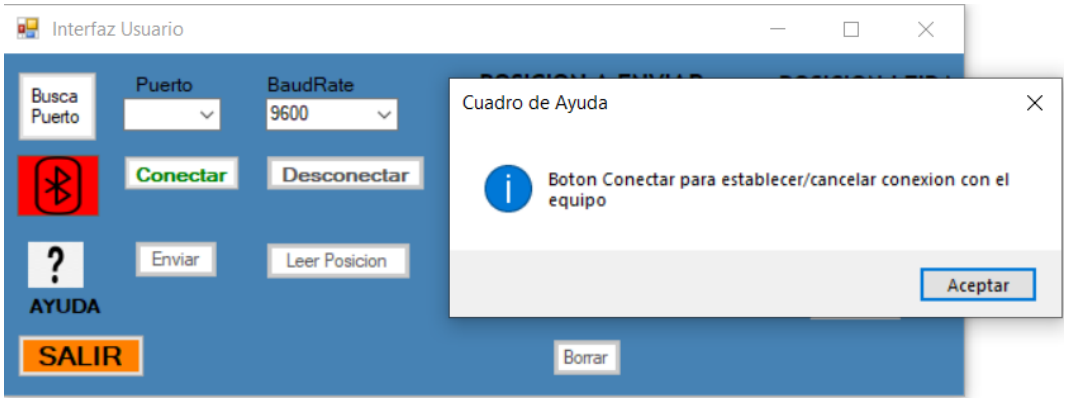

Ilustración 29: Cuadro de ayuda asociado al botón Conectar

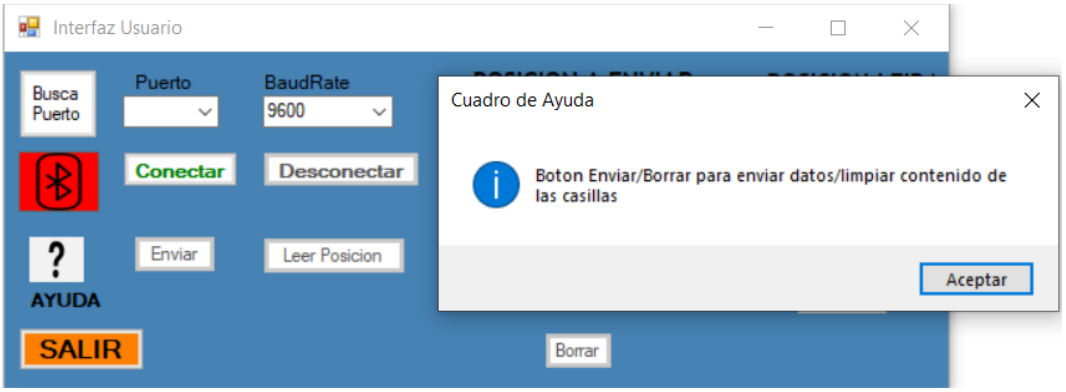

Ilustración 30: Cuadro de ayuda asociado al botón Enviar

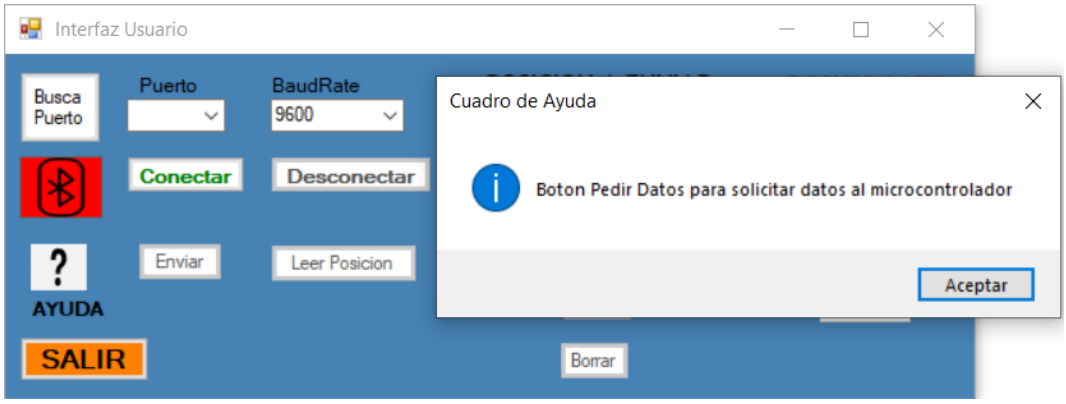

Ilustración 31: Cuadro de ayuda asociado al botón Leer Posición

4.1.3.7. Salir

Asociado al botón Salir, rutina encargada de finalizar la ejecución del programa.

4.1.4. Estructura y creación de los mensajes intercambiados

Para la creación de las tramas, vamos a disponer de unos campos fijos, que van a ser los campos de las tramas de los mensajes. Estos son los mismo descritos en el punto 3.3.3. El tipo de dato utilizado para estos datos es el byte, ya que es la unidad más pequeña y útil que podemos utilizar, y que nos va a permitir no tener problemas de compatibilidad en ambas plataformas.

Una vez dispuestos dichos campos, es hora de crear los mensajes que vamos a enviar. Para ello se han creado 3 funciones independientes, una para cada mensaje, en cada una de ellas se define un vector de tamaño fijo, que los mensajes van a disponer siempre del mismo tamaño y estructura.

4.1.4.1. Petición de Escritura

Se creado una función específica para el mensaje de petición de escritura. Esta función es de tipo Void y se invocara desde la función asociada al botón de Enviar, después de que la ejecución de la función que comprueba los datos introducidos finaliza.

Necesitamos crear un matriz de tipo byte, de tamaño correspondiente a petición de escritura, la cual vamos a rellenar con los campos requeridos, mediante la siguiente línea de código:

```
int tPETESC = 31;
byte[] mPetEsc = new byte[tPETESC];
```

Ahora ya podemos rellenar los datos del vector, según los campos descritos en la Tabla 14, del punto 3.3.4.1, empezaremos por el byte menos significativo para que el dato a enviar sea recibido correctamente.

Una vez tenemos el vector completamente relleno podemos pasar a realizar el envío del mensaje. Nos ayudamos de la función propia *serialBT.Write* la cual escribe un número especificado de bytes en el puerto serie utilizando los datos de un búfer.

Como parámetros de esta función pasamos *mPetEsc*, que es la matriz de bytes que queremos escribir en el puerto serie, 0 que corresponde al desplazamiento en bytes de base cero del parámetro buffer donde comienzan a copiarse los datos para el puerto y por último *mPetEsc.Length* es el número de bytes que se van a escribir. Como resultado tenemos la siguiente línea:

```
serialBT.Write(mPetEsc, 0, mPetEsc.Length);
```

Para finalizar mostramos un mensaje para confirmar el envío de la petición, y establecemos la variable que controla el último mensaje enviado a 1.

4.1.4.2. Petición de Lectura

Para la creación del mensaje de petición de escritura se ha creado una función específica de tipo Void.

Se ejecutará la función petición de escritura cuando el usuario solicite los datos al microcontrolador. Esto será cuando se active el botón *Pedir Datos.*

Este mensaje al no requerir datos por parte del usuario podemos crearlo de igual modo que el mensaje petición de escritura, es decir primero creando una matriz y después rellenando según los campos descritos en la Tabla 17.

Una vez cerrado el mensaje, lo enviamos por el puerto serie con la función *serialBT.Write,* de igual modo que lo descrito para el mensaje de petición de escritura.

Después de que se envíe mostramos un mensaje por pantalla para indicar que se ha completado la transmisión y pondremos la variable que controla el último mensaje enviado a 2.

4.1.4.3. Mensaje de Error

La trama de error podrá surgir en numerosos puntos durante la ejecución del programa. Es por ello por lo que se ha creado una función que se ejecutara cuando detecte alguno de los errores contemplados durante el diseño. Los campos que componen el mensaje son los descritos en la

Cuando se ejecute un mensaje de error, se notificará en la interfaz y enviara el mensaje a los microcontroladores para notificarles el Error.

Añadiremos una variable para contar el número de veces que se envía el mensaje de error y en el caso de que se produzca el máximo permitido pueda finalizar el programa, llamando a la función de Desconexión que será la encargada de llevar la interfaz al estado inicial, finalizar la comunicación y cerrar el programa.

4.1.5. Funciones complementarias utilizadas

A fin de realizar un programa más compacto y reutilizable, se han creado varias funciones complementarias a las principales que serían las de los mensajes creados, las principales son:

4.1.5.1. Recepción de datos

La recepción de los datos por parte del puerto serial necesitamos dicha indicación por parte del puerto que hemos creado con el objeto SerialPort. En nuestro programa se ha agregado un *SerialDataReceivedEventHandler* a *DataReceived* para leer todos los datos disponibles que se reciben por el puerto que se seleccione (en nuestro caso el COM8, que es el correspondiente a dispositivo bluetooth HC-06).

El evento creado para la recepción de los datos es: *SerialBT_DataReceived*

El primer elemento por configurar es el número de bytes en el buffer interno antes de que se produzca el evento. Esto lo hacemos a la hora de crear y configurar el puerto y es aspecto muy importante ya que ajustando bien este parámetro nos vamos a asegurar de esperar una cantidad suficiente de datos para poder tratarlos, ya que puede ser que no se envíen con la suficiente rapidez desde el microcontrolador y recibamos solo parte del mensaje. Como el tamaño de mensaje mínimo que vamos a utilizar en la comunicación es 6, se ha configurado este parámetro con ese valor.

```
serialBT.ReceivedBytesThreshold = 6;
```

Para poder recibir los datos vamos a checkear si hay datos disponibles para leer en el puerto. En primer lugar, vamos a leer el número de bytes que tenemos disponibles con la función *Serial.BytesToRead,* y después crearemos una matriz del tamaño leído para almacenar los datos.

Con los datos guardados vamos a comprobar que el mensaje que hemos recibido es el correcto, para ello vamos a comparar que el tamaño de los datos recibidos coincide con el campo del tamaño, si es así, podemos administrar el mensaje recibido y comprobar que todos y cada uno de los campos son los esperados, con la función *administraMSGrecibido,* la cual se explicará más adelante.

Para los casos en los cuales estas dos variables no coinciden, se crea de nuevo una matriz auxiliar para terminar de leer los datos que se encuentran en el buffer. Después combinamos ambas matrices para formar el mensaje completo y procedes a administrarlo y comprobar los datos.

Con este esquema de funcionamiento nos vamos a asegurar de que en caso de que se produzca un error por perdida de información, sea por algún problema de conexión entre los dispositivos y no por un retraso en el envío de las tramas, que haga que el mensaje no llegue dividido en partes, pero en realidad es correcto.

4.1.5.2. Administrar mensaje recibido

Función de tipo void, cuyos argumentos son el tamaño de la matriz de tipo int, y la propia matriz de bytes donde se encuentran los datos.

La utilidad de esta función es discriminar por tamaño los mensajes recibidos para saber qué tipo de petición hemos recibido y con ello continuar con la ejecución del programa.

Se recorre la matriz de datos recibida y se compara uno a uno con el formato de dicho mensaje, en el caso de ser correcto se continúa avanzado por el bucle, si hay algún campo que no se corresponde se llama a la función CodigoError, que es la que enviara el mensaje de error.

Si el mensaje que hemos recibido es el de Respuesta de lectura, necesitamos pasarlo a tipo *float*, para disponer del valor real de cada actuador y para poder presentarlo en la interfaz al usuario. Para ello usamos la función propia de C#, *BitConverter.ToSingle*, esta función devuelve un numero de punto flotante de precisión sencilla convertido a partir de cuatro bytes en la posición especificada de una matriz de bytes. Los parámetros de esta función son la matriz de bytes y la posición inicial de dicha matriz a convertir. Además, para acotar el valor a 2 decimales usamos la función propia *Math.Round* y después realizaremos un casteo a *float* para poder almacenar la variable. Como resultado general de la conversión general, para cada actuador tendremos:

```
float fAct1R = (float)Math.Round(BitConverter.ToSingle(aAct1R, 0),2);
```

Al usar un objeto de tipo *textbox* para mostrar los valores recibidos, necesitamos tener los valores a mostrar en tipo *string*, por lo que nos hace falta realizar una última transformación con la función propia *Convert.ToString,* la cual convierte el valor del número de punto flotante de precisión simple especificado en la representación de cadena equivalente.

Para añadir elementos al cuadro de texto, necesitamos crear un delegado con la rutina *this.Invoke*, este será el encargado de encapsular el método que hace que se agregue un elemento al cuadro de la lista. En él se ejecuta el subproceso que encuentre dentro de sus parámetros. Al trabajar los cuadros de texto, con variables de tipo string, tendremos que realizar otra conversión de *float* a *string*. La expresión general para poder imprimir los datos en los cuadros de texto es:

```
sAct1 = Convert.ToString(fAct1R);
this.Invoke(new EventHandler(MostrarDatos));
```

Además de esto necesitamos hacer otro evento para imprimir en el cuadro de texto, este es *TextBoxBase.AppendText(String)* siendo el *String* el texto que hemos convertido previamente y asociaremos a cada cuadro de texto su equivalente para cada actuador.

4.1.5.3. CompruebaDatos

Esta función consta de tres partes. Lo primero que hace esta función es recoger el texto que se ha introducido en los cuadros de texto de la interfaz, y lo convierte a través de la función *float.Parse* a una variable de tipo *float*. Para ello se usa la siguiente línea de código:

```
fAct1 =
float.Parse(txtAct1Env.Text.Trim(),CultureInfo.InvariantCulture.NumberFormat);
```

Como segundo paso, vamos a comprobar que los valores introducidos se encuentran dentro del rango del actuador. En caso contrario, se notificará al usuario para que vuelva a introducirlos.

Después de tener las variables de cada actuador en tipo *float* necesitamos convertirlas a una matriz de 4 bytes para poder introducir cada valor en el campo correspondiente de cada mensaje. Para ello utilizamos la función propia, *BitConverter.GetBytes().* Previamente debemos crear una variable tipo *byte[]* donde almacenar los valores que nos devuelve la función.

```
aAct1 = BitConverter.GetBytes(fAct1);
```

4.1.5.4. ReenvioMSG

Función encargada de enviar el último mensaje hacia el microcontrolador en el caso de producirse un error. Este reenvío se produce un máximo de 3 veces, si se supera este límite se pondrá fin a la comunicación.

4.1.5.5. CalculaChecksum

Como se ha explicado anteriormente, para poder calcular el *ckecksum* necesitamos realizar la suma de todos los elementos que componen las tramas a excepción del byte de inicio y fin.

Para realizar dicha suma se ha creado una función de tipo byte, por lo cual nos retorna una variable de tipo byte y como parámetros tiene una variable de tipo *int*, y una matriz también de tipo byte. Las cuales son el tamaño del mensaje y el propio mensaje

Dentro de la función se discrimina por el tamaño, el mensaje de petición de escritura y el de respuesta de lectura, ya que en ese caso se realiza la suma de los bytes fijos del mensaje.

Para calcular el *checksum*, tendremos que sumar sus elementos y realizar su complemento a dos, para obtener el valor correcto.

4.2. Implementación mediante Arduino

4.2.1. Lenguaje de programación en Arduino

El equipo de desarrollo de Arduino creo un lenguaje de programación propio para poder programar este tipo de tarjetas basado en Wiring. Este en un framework de código abierto para la programación de microcontroladores. Wiring está escrito en C/C++. La principal ventaja de usar este framework para su programación es la gran cantidad de microcontroladores que soporta, el caso que nos interesa a nosotros también el Atmega que es el usado por Arduino. [11]

4.2.2. Diagrama de flujo

El funcionamiento desde el punto de vista del microcontrolador es la de servidor, el cual se encuentra a la espera de resolver las peticiones por parte del cliente.

El funcionamiento general de la aplicación se describe en la ilustración 32:

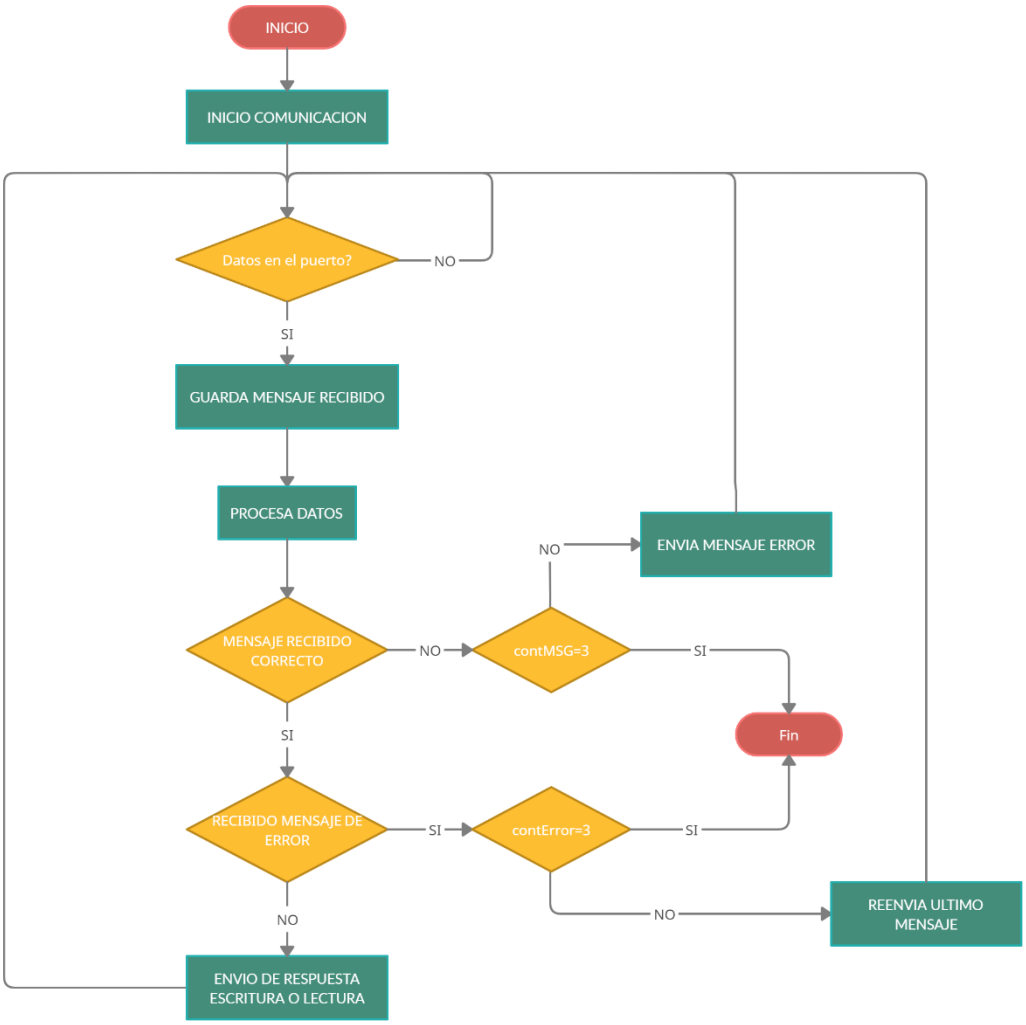

Ilustración 32: Diagrama de Flujo del funcionamiento de Arduino

En el estado inicial el Arduino inicia los dos puertos de comunicación, una para su comunicación serial y otro para la vía bluetooth. A partir de ese momento, queda a la espera de recibir datos por su puerto. En el momento que se detectan datos entrantes en el puerto, se almacenan en una variable para después poder mandarlos a la función que se encarga de comprobar la integridad de los datos recibidos. En primer lugar, se hace una discriminación por tamaño, si el mensaje recibido no corresponde con el tamaño que se tiene predefinido o alguno de los campos no son los correctos se generara el mensaje de error para enviar a la aplicación.

Si la trama recibida es alguna de las contempladas (Petición Escritura, Petición Lectura o Mensaje de Error), se pasará a analizar que los campos del mensaje sean los correctos.

Si recibimos un mensaje de error, esto significara que el último mensaje enviado es incorrecto, por lo que tendremos que proceder al reenvío del último mensaje, esta operación se realizara un máximo de 3 veces antes de poner fin a la comunicación.

Si el mensaje recibido es alguna de las peticiones, se comenzará con su rutina. Para el mensaje de Petición de Escritura, se desencriptará el mensaje el mensaje y se almacenaran los datos de los actuadores en sus variables correspondientes además de mostrarlas por el propio monitor serial de Arduino. Después se devolverá hacia la aplicación la trama de Respuesta de Escritura para notificar que el mensaje se ha recibido correctamente.

Por el contrario, si recibimos una petición de lectura, tendremos que solicitar los datos a través del monitor serial, almacenarlos y generar el mensaje de Respuesta de Lectura para enviar los datos a la interfaz de usuario en el PC.

4.2.3. Librería SoftwareSerial [12]

4.2.3.1. Descripción

El propio Hardware de Arduino tiene incorporado los pines 0 y 1, para la comunicación serie, pero estos pines también van al conector USB donde se cargas los sketches, por lo que es delicado usarlos. No se puede realizar transmisiones simultáneas por el mismo pin.

Esta librería lo que permite es la comunicación en serie por otros pines digitales de Arduino, vía software para realizar la misma funcionalidad. Con ella, es posible crear múltiples puertos serie de software con velocidades de hasta 115200 bps.

4.2.3.2. Sintaxis y parámetros

Para poder utilizar la librería deberemos incluir la cabecera:

#include <SoftwareSerial.h>

Para llamar la función tendremos que realizar la siguiente declaración:

SoftwareSerial NombreT(Rx,Tx);

Con:

- NombreT: el que deseemos asignar al puerto serie
- Rx: pin digital para Recibir datos
- Tx: pin digital para Transmitir datos

4.2.3.3. Métodos

METODO	DESCRIPCION
begin()	Abre el puerto serie, fija su configuración y velocidad de comunicación
available()	Indica el número de bytes/caracteres disponibles para lectura
read()	Lee un carácter disponible y lo borra del buffer
peek()	Lee un carácter disponible, pero queda disponible en el buffer (no borra)
write()	Envía datos binarios al puerto serie
flush()	Espera a que la transmisión de datos de salida serie termine
print()	Imprime una representación textual de un dato
println()	Imprime una representación textual de un dato junto con su saldo de línea; o solo este ultimo
overflow()	Prueba para ver si se ha producido un desbordamiento del búfer en serie del software
listen()	Habilita el puerto serie del software seleccionado para escuchar. Solo un puerto serie de software puede escuchar a la vez; Los datos que lleguen a otros puertos serán descartados. Cualquier dato ya recibido se descarta durante la llamada a listen() (a menos que la instancia dada ya este escuchando). Devuelve un bool.
isListening()	Pruebas para ver si el puerto serie del software solicitado está escuchando activamente. Devuelve bool.
end()	Desactiva la comunicación. Para reactivar usar begin() nuevamente

Tabla 21: Métodos disponibles con SoftwareSerial.h

4.2.3.4. Limitaciones

Esta librería tiene las siguientes limitaciones conocidas:

- Si utiliza varios puertos de serie de software, solo uno puede recibir datos a la vez
- No todos los pines del Arduino Mega admiten interrupciones, por lo que solo se puede usar los siguientes pines para RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- No todos los pines del Arduino Leonardo admiten interrupciones, por lo que solo se puede usar lo siguiente para RX: 8,9,10,11,14 (MISO), 15(SCK), 16(MOSI)
- En Arduino o Genuino 101, la velocidad máxima de RX actual es 57600 bps
- En Arduino o Genuino 101 RX no funciona en el Pin 13

4.2.3.5. Advertencias

- Siempre es mejor opción utilizar hardware serial, ya que tiene mucho mejor rendimiento
- *AltSoftSerial* puede transmitir y recibir simultáneamente. Consume un temporizador de 16 bits, deshabilita algunos pines PWM
- *SoftwareSerial* puede tener varias instancias en casi cualquier pin, pero solo 1 puede estar activo a la vez. Tampoco se puede transmitir y recibir al mismo tiempo.

4.2.4. Esquema de conexión

Para poder conectar el módulo HC-06 al Arduino debemos de conectar los pines del módulo a +5V y GND respectivamente. Para conectar los RX y TX, como hemos visto durante la explicación del protocolo serial asíncrono, será de forma cruzada entre Arduino y el módulo. El propio modulo en la placa tiene asignados unos pines específicos TX y RX, pero estos al necesitar de la comunicación vía USB con el PC, no podemos usarlo. Para ello se hace indispensable el uso de la librería SoftwareSerial, que nos permite crear un puerto serial virtual. Con ello podemos escoger cualquier pin Digital de la placa para poder crearlo, y a través de la definición de la librería en el programa seleccionarlos. En mi caso he elegido los pines 11 y 10. El pin 11 (cable naranja Ilustración 34) será el que conectamos al RX del módulo y el pin 10 (cable azul Ilustración 34) al TX.

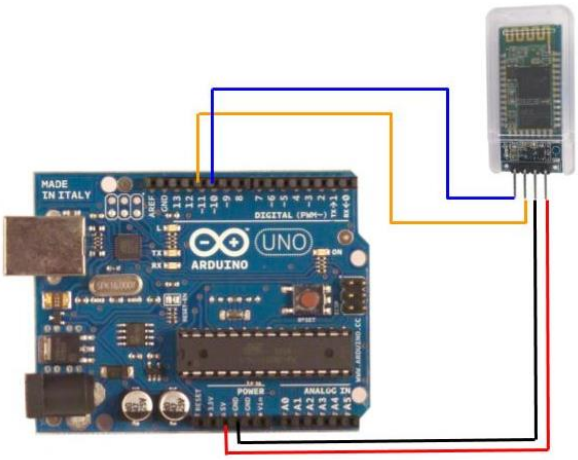

Ilustración 33: Esquema de conexión entre Arduino y HC-06

4.2.5. Funciones implementadas

El primer paso del desarrollo del programa con Arduino es definir la librería software serial, la cual iniciaremos los pines de TX/RX a los 10 y 11 del Arduino.

Los bytes que componen los mensajes descritos en puntos anteriores los definiré al comienzo del programa, y fuera de todos los bucles principales, para que puedan ser usados por todas las partes del sketch. Además, esto nos permitirá poder reutilizar el programa o poder modificar los valores de las tramas de una manera más segura, en definitiva, aumentar la compatibilidad del programa.

4.2.5.1. Union en C

El principal problema que nos vamos a encontrar durante la realización del programa es como convertir de un valor de tipo float a un array de 4 bytes que podamos añadir al mensaje que necesitamos enviar por el puerto serial de Arduino. Para ello tenemos en C un tipo de dato especial llamado unión. Estas uniones nos permiten utilizar de una forma eficiente la misma ubicación de memoria.

Para definir la union, tenemos que usar su declaración propia unión, del mismo modo que definiríamos una estructura. Dentro de la unión vamos a definir los miembros que componen la unión:

```
union datos {
        float fval;
        byte  b[4];
} act1, act2, act3, act4, act5;
```

Una vez definida, tenemos una variable de tipo datos la cual puede almacenar una variable de tipo *float* o una cadena de bytes. Esto significa que

una sola variable, la misma ubicación de memoria, se puede usar para almacenar múltiples tipos de datos. [13]

Para usar cualquier miembro de la unión, se utiliza el operador de acceso (.), por ejemplo, *act1.fval*

4.2.5.2. Setup

En esta parte del programa vamos a inicializar la comunicación tanto con el monitor serie de Arduino como con el módulo bluetooth. Para ello nos vamos a ayudar del método *begin()* descrito en la tabla 21. Configuraremos la velocidad de transmisión a 9600.

4.2.5.3. Loop

Es la parte principal del programa, en ella el microcontrolador va a estar ejecutando el código continuamente, es por ello que aquí es donde vamos a realizar la comprobación de cuando tenemos datos disponibles en el puerto. Para ello utilizamos el método *availbale()* para comprobar cuando y cuantos datos hay en el puerto. Aprovechando esta última característica vamos a poder crear el matriz donde almacenar los datos recibidos con el tamaño que nos devuelve dicha función. Una vez tenemos leído el mensaje.

4.2.5.4. Respuesta Escritura

Este mensaje se ejecuta como mensaje de confirmación de que de que la instrucción de Petición de Escritura se ha recibido correctamente. Una vez creado la matriz asignamos a cada campo su valor correspondiente y después procedemos a enviarlo gracias a la función *write()*. Cuando se envía el mensaje completo es importante establecer la variable que nos permite identificar el último mensaje enviado a 1, para en caso de que se produzca error, poder saber qué mensaje hay que reenviar.

4.2.5.5. Respuesta Lectura

Es función va a ser la encargada de pedir los datos al usuario a través del monitor serial de Arduino y enviarlos hacia el PC. Para ello tendremos que invocar en primer lugar a la función auxiliar que pide datos.

Una vez se tiene los datos, el primer paso para la creación de la instrucción completa, es convertir el valor tipo *float* que hemos recogido a su equivalente array de 4 bytes para poder enviarlo encapsulado en el mensaje. Para ello hacemos uso de nuevo de la *Union*, a través de la siguiente instrucción:

```
act1.fval = fAct1R;
```

El envío se realiza de manera análoga a la trama de Respuesta de Escritura. En este caso pondremos la variable de último mensaje enviado a 2.

4.2.5.6. Mensaje Error

Esta instrucción es la más sencilla de todas de construir ya que todos sus campos son fijos y constantes siempre. En se va a llevar el control del estado de error. Cada vez que se envíe, el contador de envío de mensajes de error aumentará, y una vez llegue al máximo establecido, será el encargado de ejecutar la rutina para finalizar la comunicación.

4.2.5.7. Procesa Datos

Función encargada de procesar el mensaje recibido. Una vez sabemos el número de datos que tenemos en el puerto, se va a crear una matriz de tamaño el leído donde vamos a almacenar el mensaje. Si el tamaño no coincide con los establecidos se descarta el mensaje y se notifica el error. Cuando el tamaño corresponde a las peticiones, ya sea de lectura o de escritura o del mensaje de error, se pasará después a la comprobación de que cada campo del mensaje es el adecuado, en caso de no serlo se notificara el error.

En el caso de Petición de Lectura y Error, cuyos campos son fijos y no incluyen ningún dato para ser almacenado y procesado, una vez se llegue al último byte del mensaje y se compruebe se invocará a la función correspondiente para que se genere la respuesta a la petición solicitada.

Sin embargo, cuando recibimos una Petición de Escritura, que sí que contiene datos para almacenar tendremos que recoger dichos datos, los cuales vendrán en tipo *byte* y convertirlos a su equivalente *float*, para poder guardarlos. Para ello asociaremos cada byte a su matriz correspondiente en *Big-endian* [14]

```
aAct1[0] = bLeido[7];

aAct1[1] = bLeido[6];

aAct1[2] = bLeido[5];

aAct1[3] = bLeido[4];
```

Después para convertirlo a tipo *float* usaremos un puntero a memoria, con la siguiente expresión:

```
float* fp = (float*)aAct1;

fAct1 = *fp;
```

Después de terminar de recoger todos los datos de los actuadores y comprobados que el resto de los campos del mensaje son correctos, mostraremos los datos al usuario a través del monitor serial, e invocaremos a la función Respuesta de Escritura para devolver la confirmación de recepción de los datos.

4.2.5.8. Pide Datos

A través de esta función auxiliar vamos a solicitar los datos de los actuadores, a modo de simulación de los que sería realizar una lectura directa al sensor de los actuadores del prototipo. De forma análoga para los 5 actuadores, se solicitará al usuario la introducción de la posición.

4.2.5.9. Reenvío Mensaje

Esta función simplemente controla a través de la variable de último mensaje enviado, a qué tipo de trama tiene que invocar para realizar el reenvío del mensaje, cuando se ha producido un error en el envío último.

4.2.5.10. Calcula Checksum

Función de tipo byte, que tiene como argumentos una variable de *int* que es el tamaño del vector, y otra de tipo *byte[]* que es la matriz de bytes que se ha recibido por serial.

La función dependiendo del tamaño de los datos recibidos, va a sumar esos campos. En el caso de que el mensaje sea el de Respuesta Lectura o Petición Escritura. En este caso se sumarán los bytes fijos del mensaje para comprobar que es el correcto

Para el resto de los casos se sumarán todos los bytes del mensaje a excepción de los bytes de inicio y de fin.

Esta función también nos va a servir para comprobar la integridad del dato que recibimos. Para ello dentro de la comprobación de cada mensaje llamaremos a esta función.

4.3. Implementación mediante Texas Instruments

4.3.1. Lenguaje de programación Texas Instruments

El lenguaje básico de programación en los microcontroladores de Texas Instruments, el lenguaje C. La ventaja principal que nos proporciona este tipo de lenguaje para la programación de este tipo de microcontrolador es que además de disponer estructuras típicas de los lenguajes de alto nivel, también las tiene a nivel bajo que es donde nos interesa llegar en la programación. Con esto y gracias al compilador conseguimos mezclar el código ensamblado con código C o acceder directamente a la memoria de este.

4.3.2. Diagrama de flujo

El modo de funcionamiento con el microcontrolador de Texas Instruments, es a grandes rasgos similar al de Arduino pero con alguna pequeña variación.

El funcionamiento general de la aplicación se describe en la ilustración siguiente:

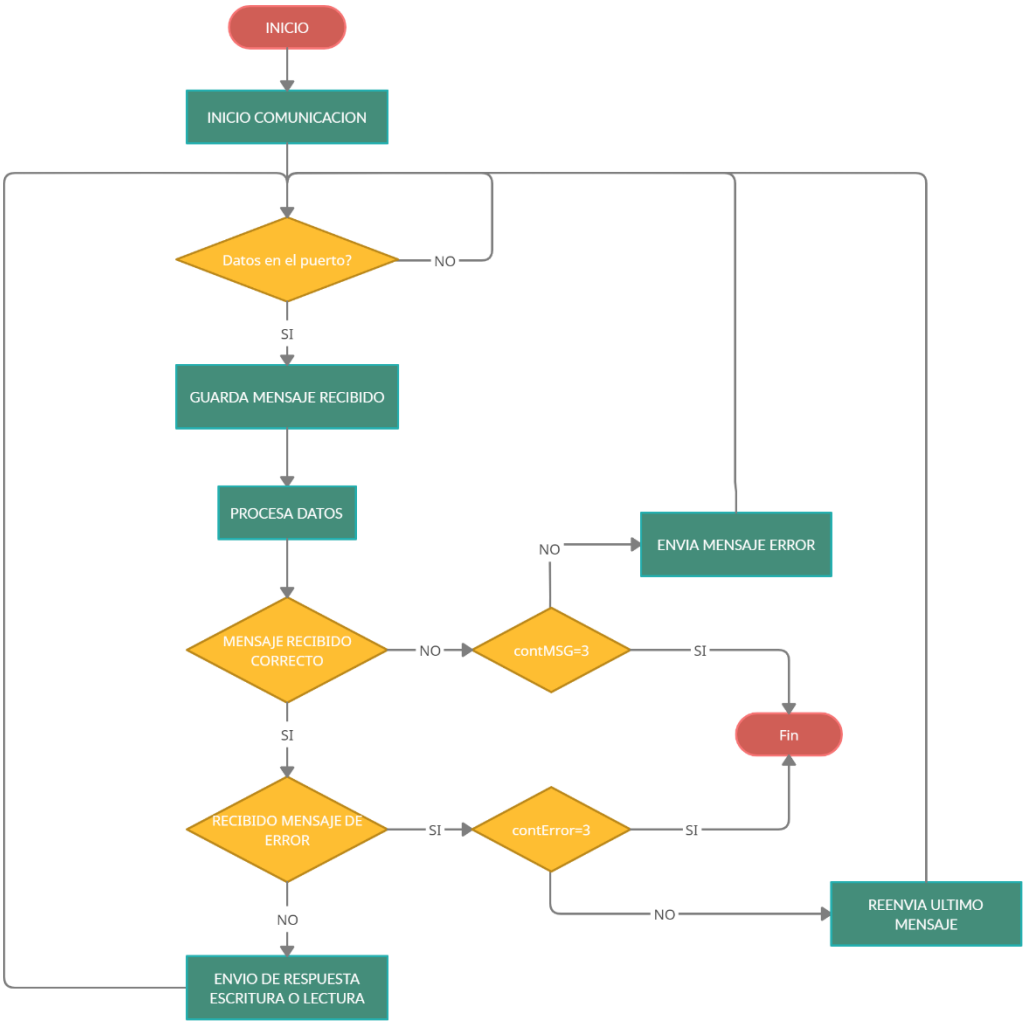

Ilustración 34: Diagrama de flujo aplicación en F28069M

El funcionamiento de la aplicación en F28069M es similar a la utilizada en el otro microcontrolador, pero utilizando sus funciones y programación propia.

En el inicio de la ejecución del programa tenemos que inicializar ciertos recursos del sistema que son necesarios para el correcto funcionamiento de la placa, esto lo conseguimos con *InitSysCtrl().* Para poder utilizar correctamente los pines y los GPIO de la placa tenemos que en primer lugar inicializar la función donde se incluyen todos los pines con initGpio() y además de este el de cada puerto SCI especifico que vamos a utilizar, estos son InitSciaGpio() y InitScibGpio(). En ellos tenemos que definir cuáles de los pines disponibles que tiene la placa vamos a usar, así como realizar su configuración la cual incluye activar su resistencia pull-up y establecer que pin es el transmisor y cual el receptor.

Los otros dos necesarios son las funciones que nos van a permitir disponer de los dispositivos PIE que viene con la placa *InitPieCtrl()* y *InitPieVectTable().*

La configuración de cada puerto SCI-A y SCI-B, se realiza con las funciones *iniciaSCIa()* y *iniciaSCIb().* En estas funciones se realiza la configuración del puerto (byte de datos, parada, arranque, velocidad de transmisión…)

El programa se encuentra a la espera de recibir datos por el puerto Bluetooth, para poder avanzar al siguiente estado. Una vez recibe datos se continua con su procesamiento para determinar qué tipo de mensaje se ha recibido. Si el mensaje no es correcto se genera la rutina de error, siempre y cuando el contador de envío de mensajes de error no llegue al máximo, momento en el cual se pondría fin a la comunicación.

Si la trama recibida es correcta, pero es un mensaje de error, se pasará a la subrutina de reenvío de mensaje, ya que esto será indicativo que ha habido algún error en el último mensaje enviado.

Cuando se han realizado todas las comprobaciones y el mensaje recibido es finalmente el esperado se genera la respuesta a la petición recibida, y el programa se queda en el punto inicial, esperando a recibir nuevos datos por el puerto para poder nuevamente procesarlos.

4.3.3. Instalación de Control Suite

Para poder utilizar será necesario instalar también ControlSuite: https://www.ti.com/tool/CONTROLSUITE

Además de control suite deberemos instalar C2000Ware que es el software más reciente disponible para poder utilizar nuestra placa de desarrollo.

Como base para el desarrollo del programa se ha utilizado el programa de ejemplo dado con este software que es *sci_echoback.* Este utiliza la configuración inicial para una comunicación serial básica.

4.3.4. Esquema de conexión

Para poder establecer comunicación entre el PC y Visual Studio es necesario configurar dos puertos, uno para comunicarnos vía serial con el PC y crear otro virtual por el que enviamos los datos vía bluetooth a la interfaz en el PC.

Para la comunicación vía USB, usaremos la predeterminada que viene cargada con la placa que por defecto es pines 28 y 29 que son a los que están asociados el cable USB, esto lo podemos comprobar en el documento de introducción de la placa:

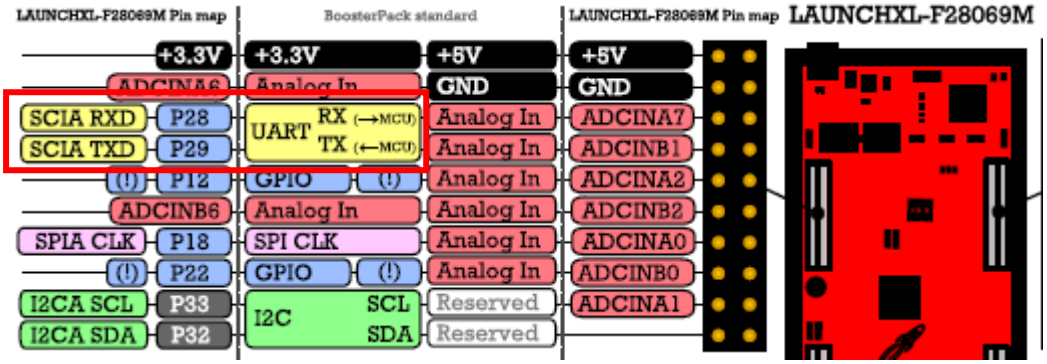

Ilustración 35: Mapa de Pines en LAUNCHXL-F28069M [15]

Para configurar el puerto virtual, sobre el que conectaremos el módulo HC-06, tenemos que mirar a la configuración que hay que hacer en la placa. Si acudimos al manual de usuario podemos ver en el apartado de conectividad serial que para usar la depuración USB, tenemos que conectar el Jumping J7 y desconectado el Jumping J6

MUX_SEL (JP7)	CH_SEL(JP 6)	Function
ON	ON	USB/UART Disabled; J1.3 and J1.4 – GPIO28 and GPIO29; J7.3 and J7.4 – GPIO15 and GPIO58
ON	OFF	USB/UART – GPIO28 and GPIO29, J1.3 and J1.4 – Hi-Z; J7.3 and J7.4 – GPIO15 and GPIO58
OFF	ON	USB/UART – GPIO15 and GPIO58; FAULT/OCTW – GPIO28 and GPIO29; J7.3 and J7.4 – Hi-Z
OFF	OFF	USB/UART – GPIO15 and GPIO58; FAULT/OCTW – GPIO28 and GPIO29; J7.3 and J7.4 – Hi-Z

Ilustración 36: Tabla de conectividad serial

Después en este mismo manual tendremos que ver que pines de nuestra placa son válidos para crear otro segundo puerto de comunicación, por ejemplo, SCITXDB y SCIRXDB.

Vamos a localizar estos puertos dentro de las tablas del fabricante para ver cómo debemos conectar el módulo Bluetooth a la placa.

Mux Value				J1 Pin	J3 Pin	Mux Value			
3	2	1	0			0	1	2	3
			+3.3V	1	21	+5V			
			ADCINA6	2	22	GND			
			J1.3	3	23	ADCINA7			
			J1.4	4	24	ADCINB1			
SPISIMOB	SCITXDA	TZ1	GPIO12	5	25	ADCINA2			
			ADCINB6	6	26	ADCINB2			
XCLKOUT	SCITXDB	SPICLKA	GPIO18	7	27	ADCINA0			
SCITXDB	MCLKXA	EQEP1S	GPIO22	8	28	ADCINB0			
ADCSOCBO	EPWMSYNCO	SCLA	GPIO33	9	29	ADCINA1			
ADCSOCAO	EWPMSYNCI	SDAA	GPIO32	10	30	NC			

Ilustración 37: Opciones de pines de salida para J1 y J2 [7]

Mux Value				J4 Pin	J2 Pin	Mux Value			
3	2	1	0			0	1	2	3
Rsvd	Rsvd	EPWM1A	GPIO0	40	20	GND			
COMP1OUT	Rsvd	EPWM1B	GPIO1	39	19	GPIO19	SPISTEA	SCIRXDB	ECAP1
Rsvd	Rsvd	EPWM2A	GPIO2	38	18	GPIO44	MFSRA	SCIRXDB	EPWM7B
COMP2OUT	SPISOMIA	EPWM2B	GPIO3	37	17	NC			
Rsvd	Rsvd	EPWM3A	GPIO4	36	16	RESET#			
ECAP1	SPISIMOA	EPWM3B	GPIO5	35	15	GPIO16	SPISIMOA	Rsvd	TZ2
SPISOMIB	Rsvd	TZ2	GPIO13	34	14	GPIO17	SPISOMIA	Rsvd	TZ3
			NC	33	13	GPIO50	EQEP1A	MDXA	TZ1
			DAC1	32	12	GPIO51	EQEP1B	MDRA	TZ2
			DAC2	31	11	GPIO55	SPISOMIA	EQEP2A	HRCAP1

Ilustración 38: Opciones de pines de salida para J4 y J2 [7]

Como vemos tenemos el pin 7 que corresponde con el GPIO18 y el pin 19 que corresponde con el GPIO19. Ahora deberemos inicializar dichos pines dentro de nuestro programa para ello deberemos cargar el InitGpio();que encontramos dentro de Controlsuite, una vez añadido este fichero a nuestro proyecto tendremos que ir a F2806x_Sci.c para activar dichos pines GPIO18 y GPIO19. Habilitamos su resistencia pull-up y después como vemos en la tabla de arriba tenemos que asignarles el valor del MUX correspondiente a la tabla, en este caso 2.

Una vez conocido los pines de conexión para el módulo ya podemos proceder con su conexionado:

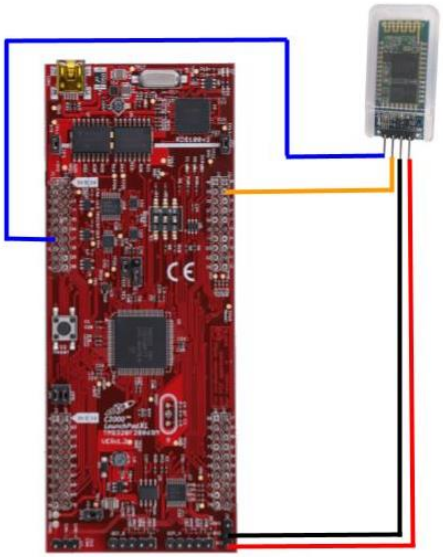

Ilustración 39: Esquema de conexión entre F28069M y HC-06

4.3.5. Funciones implementadas

Una vez inicializadas todas las funciones principales para que la comunicación serial funcione correctamente, tenemos que derivar el programa en diferentes rutinas para lograr una correcta ejecución del algoritmo.

4.3.5.1. Inicia Puerto Serial A (serial UART) y Puerto Serial B (Bluetooth)

En estas funciones tendremos que inicializar y dar los valores a los registros de configuración del puerto, según lo descrito en el manual que nos proporciona el propio fabricante. Lo vamos a configurar de igual modo, pero debemos de hacerlas por separado ya que sus registros, aunque realizan las mismas funciones su llamada es diferente (SciaRegs y ScibRegs)

En primer lugar, vamos a configurar el registro SCICCR, el cual define el formato de los caracteres, le protocolo y el modo de comunicación que vamos a utilizar. Podemos configurar cada una de las direcciones por separado o configurar su valor vía hexadecimal que es más cómodo, por lo que si establecemos el valor de este registro a:

```
SciaRegs.SCICCR.all = 0x0007;
```

Esto significa que vamos a trabajar sin bit de parada, sin paridad, sin modo loopback, con el modo Idle-line, y la longitud de los caracteres será de 8.

El siguiente registro SCICTL1, controla la habilitación del receptor/transmisión, las funciones TXWAKE y SLEEP y el reinicio del SCI. La configuración para este es:

```
SciaRegs.SCICTL1.all = 0x0003;
```

Con esto, activamos la transmisión y recepción de datos, el reloj interno y desactivamos TXWAKE y SLEEP.

Tenemos que desactivar los bits TXINTENA y RXBKINTENA del registro SCICTL2, ya que no vamos a utilizar las interrupciones.

Para poder configurar la velocidad de transmisión de los datos a 9600 como en el resto de proyecto tendremos que usar el registro SCIBAUD y lo configuramos con los siguientes parámetros:

```
SciaRegs.SCIHBAUD = 0x0001;

SciaRegs.SCILBAUD = 0x0024;
```

Por último, al utilizar la cola FIFO para la transmisión y recepción de datos deberemos configurar los siguientes registros.

SCIFFTX es el registro encargado de la transmisión, SCFFRX controla la recepción y SCIFFCT en el registro de control.

Como expresiones finales tendremos:

```
SciaRegs.SCIFFTX.all = 0xE040;

SciaRegs.SCIFFRX.all = 0x2044;

SciaRegs.SCIFFCT.all = 0x0;
```

4.3.5.2. Loop

En la ejecución principal del programa, vamos a establecer el ciclo de repetición, a través de un bucle de tipo *for*, al que no agregaremos condiciones.

Dentro del propio bucle, vamos a quedar a la espera de recibir datos por el puerto serial B, que es el configurado para Bluetooth, con el registro *ScibRegs.SCIFFRX.bit.RXFFST.* Vamos a ir leyendo y almacenando los datos en una matriz hasta dar con el byte de Fin, momento en el cual llamaremos a la función que procesa el mensaje recibido que será el encargado de derivar el programa hacia las siguientes instrucciones.

4.3.5.3. Union en C

Para este apartado podremos definir la unión de la misma manera que con Arduino, con una salvedad, que previamente hay que crear una estructura para definir el tipo y tamaño de cada byte.

```
struct BYTES
{
        Uint8 byte1 :8;
        Uint8 byte2 :8;
        Uint8 byte3 :8;
        Uint8 byte4 :8;
};
```

Ahora procedemos a definir la Union:

```
union
{
   float32 fval;
   struct BYTES bytes;
} act1, act2, act3, act4, act5;
```

4.3.5.4. Respuesta Escritura

Este mensaje se ejecuta como mensaje de confirmación de que de que la instrucción de Petición de Escritura se ha recibido correctamente. Una vez creado la matriz asignamos a cada campo su valor correspondiente y después procedemos a enviarlo gracias a la función *enviaSCIb*. A esta función tenemos que pasarla el mensaje que hemos creado con los campos correspondientes a este tipo de mensaje, y se encargara de ir cargando en el buffer de la cola FIFO cada dato del mensaje para poder enviarlo.

```
SciaRegs.SCITXBUF = a;
```

(siendo a cada uno de los campos que se leen del mensaje a enviar)

Cuando se envía el mensaje completo es importante establecer la variable que nos permite identificar el último mensaje enviado a 1, para en caso de que se produzca error, poder saber qué mensaje hay que reenviar.

4.3.5.5. Respuesta Lectura

Es función va a ser la encargada de pedir los datos al usuario a través del monitor serial del IDE Code Composer Studio y enviarlos hacia el PC. Para ello tendremos que invocar en primer lugar a la función auxiliar que pide datos.

Esta función auxiliar, como veremos en su apartado correspondiente, se encargará de recoger los datos y una vez transformados los cargará en el campo de tipo *float* en la unión. Para poder enviarlos de forma adecuada en la trama, tendremos que cargar cada dato accediendo a su posición de memoria correspondiente de tipo *byte*, para enviarlos en el mensaje lo haremos siguiendo *Big-Endian.*

En definitiva, para poder enviar por ejemplo correctamente el valor del Actuador 1, dentro de la trama tendremos que colocar los campos de la siguiente manera:

bAct1, act1.bytes.byte4, act1.bytes.byte3, act1.bytes.byte2, act1.bytes.byte1

El envío se realiza de manera análoga a la trama de Respuesta de Escritura, después pondremos la variable de último mensaje enviado a 2.

4.3.5.6. Error

Esta instrucción es la más sencilla de todas de construir ya que todos sus campos son fijos y constantes siempre. En se va a llevar el control del estado de error. Cada vez que se envíe, el contador de envío de mensajes de error aumentará, y una vez llegue al máximo establecido, será el encargado de ejecutar la rutina para finalizar la comunicación.

4.3.5.7. Procesa Datos

Función que recibe el mensaje captado y lo identifica. Se encarga de leer el número de datos que se han extraído del buffer y con esa variable, hace una primera discriminación por tamaño. Si el tamaño no coincide con los establecidos se descarta el mensaje y se notifica el error. Cuando el tamaño corresponde a las peticiones, ya sea de lectura o de escritura o del mensaje de error, se pasará después a la comprobación de que cada campo del mensaje es el adecuado, en caso de no serlo se notificara el error.

En el caso de Petición de Lectura y Error, cuyos campos son fijos y no incluyen ningún dato para ser almacenado y procesado, una vez se llegue al último campo de la instrucción y se compruebe, se invocará a la función correspondiente para que se genere la respuesta.

Cuando recibimos una Petición de Escritura, tendremos que guardar los datos correspondientes a los actuadores, para ello guardaremos cada byte dentro de nuestra unión:

```
act2.bytes.byte1 = MSG[12];
act2.bytes.byte2 = MSG[11];
act2.bytes.byte3 = MSG[10];
act2.bytes.byte4 = MSG[9];
```

Para almacenar el valor de tipo *float,* usamos la expresión equivalente que nos proporciona la Union:

```
Act1R = act1.fval;
```

Terminado de recoger y comprobar todos los datos invocamos a la función Respuesta de Escritura para enviar la confirmación de recepción de los datos.

4.3.5.8. Pide datos

Encargada de recoger los datos que se introducen, es este caso debido al rango de valores con los que se trabaja, se van a recoger 5 dígitos por cada actuador incluyendo el punto de tal modo que, si queremos enviar el número 5, tendremos que escribir vía serial 05.00, de esta manera nos aseguramos que no se produce una pérdida de datos, cuando convertimos a *float*.

Después de recoger las distancias de los 5 actuadores, usando la estructura unión creada previamente, asignamos el valor *float* a la conversión de los dígitos recogidos. Previamente tenemos que convertir la cadena recogida a su valor numérico, para eso nos apoyamos de la funcion propia del lenguaje C *atof*. Finalmente implementando ambas herramientas, tenemos la expresión:

```
act1.fval = atof(aAct1R);
```

4.3.5.9. Reenvía Mensaje

Función de control del ultima mensaje enviado para poder hacer los reenvíos de los mensajes en el caso de que se produzca algún fallo en el envío de las tramas hacia el la interfaz de Visual Studio

4.3.5.10. Calcula Checksum

Función de tipo byte, que tiene como argumentos una variable de *int* que es el tamaño del vector, y otra de tipo *byte[]* que es la matriz de bytes que se ha recibido por serial.

La función dependiendo del tamaño de los datos recibidos, va a sumar esos campos. En el caso de que el mensaje sea el de Respuesta Lectura o Petición Escritura. En este caso se sumarán los bytes fijos del mensaje para comprobar que es el correcto. Para el resto de los casos se sumarán todos los bytes del mensaje a excepción de los bytes de inicio y de fin.

Esta función también nos va a servir para comprobar la integridad del dato que recibimos. Para ello dentro de la comprobación de cada mensaje llamaremos a esta función.

CAPITULO 5: RESULTADOS

En este capítulo se van a detallar los resultados finales obtenidos durante el trascurso del desarrollo del TFG. Vamos a ver cómo se desarrolla de la comunicación entre plataformas creadas y mencionadas durante todo el trascurso de la memoria:

5.1. RESULTADOS OBTENIDOS USANDO ARDUINO

Para realizar la conexión del módulo HC-06 con Arduino (he usado uno del fabricante ELEGOO) se ha realización con una *protoboard* y unos cables para realizar la conexión.

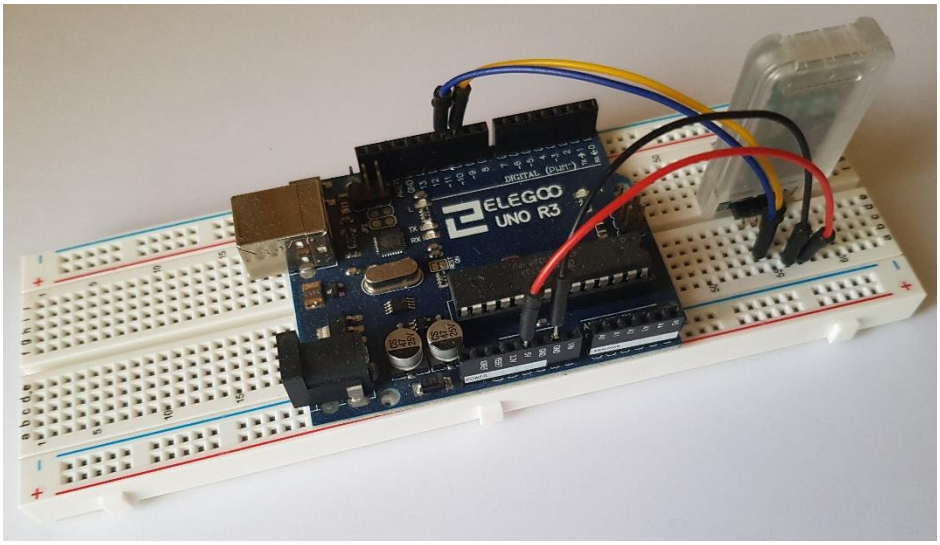

Ilustración 40: Prototipo usado con Arduino

Vamos a presentar varios casos de ejecución del programa usando este microcontrolador:

5.1.1. Comunicación completa

Vamos a ver un ejemplo de solicitud de datos, en primer lugar, petición de escritura y después de lectura.

Siguiendo los pasos previos que se desarrollaron y explicaron en el apartado *4.1.3 Interfaz Desarrollada,* se procede al intento de conexión con el módulo el cual se realiza de manera correcta, el usuario introduce los datos a enviar (Ilustración 41) y se pulsa el botón de enviar, en ese momento se produce el envío y se notifica con un mensaje vía interfaz de que se ha enviado la trama (Ilustración 42)

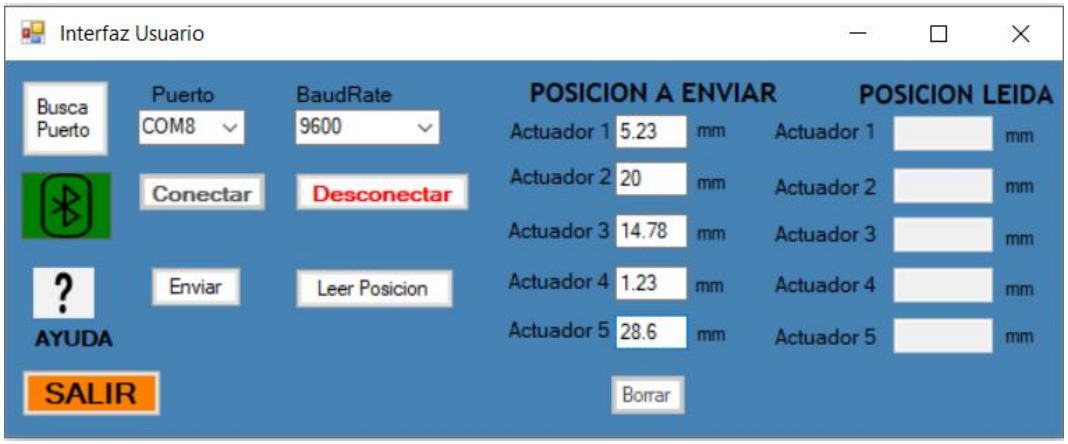

Ilustración 41: Introducción de datos para envío en la aplicación de Visual Studio

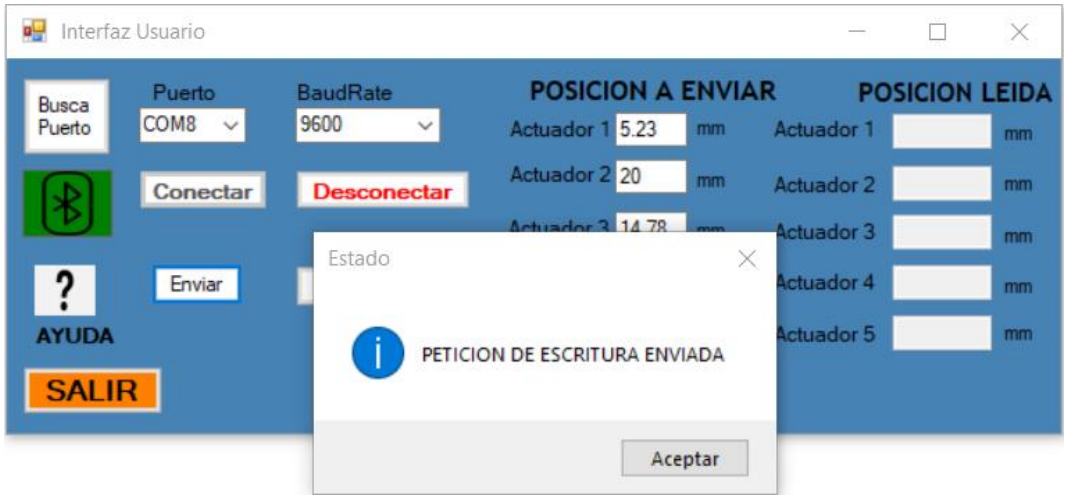

Ilustración 42: Confirmación de envío de petición escritura

Del lado de Arduino, en su monitor serial vemos un mensaje donde se indica que se ha recibido la petición de escritura, y continua después mostrándonos los datos recibidos. Una vez acaba de mostrarles, envía la respuesta de escritura y nos los indica con un mensaje a través del monitor serial

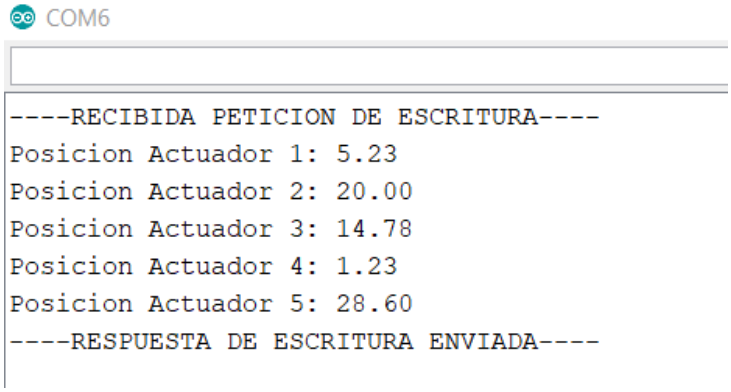

Ilustración 43: Secuencia de recepción de datos en Arduino

El computador nos muestra el mensaje (Ilustración 44) de que hemos recibido la respuesta de escritura, por lo que el ciclo de Petición de Escritura se da por finalizado.

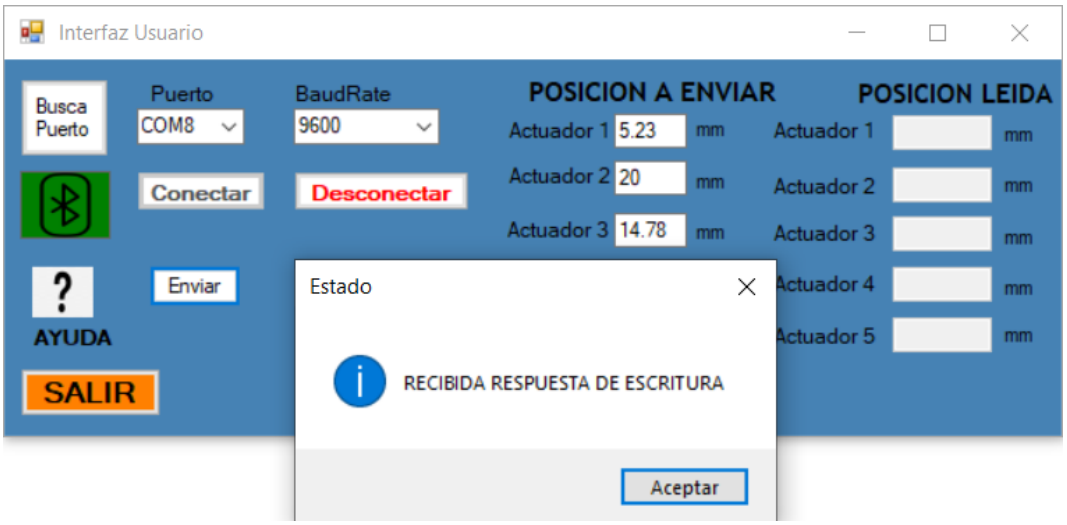

Ilustración 44: Confirmación de datos correctos con la Repuesta de Escritura

En el mismo ciclo de ejecución procedemos a solicitar datos al microcontrolador, pulsando el botón de leer posición, de igual modo que en casos anteriores se notifica que se han ha enviado la petición.

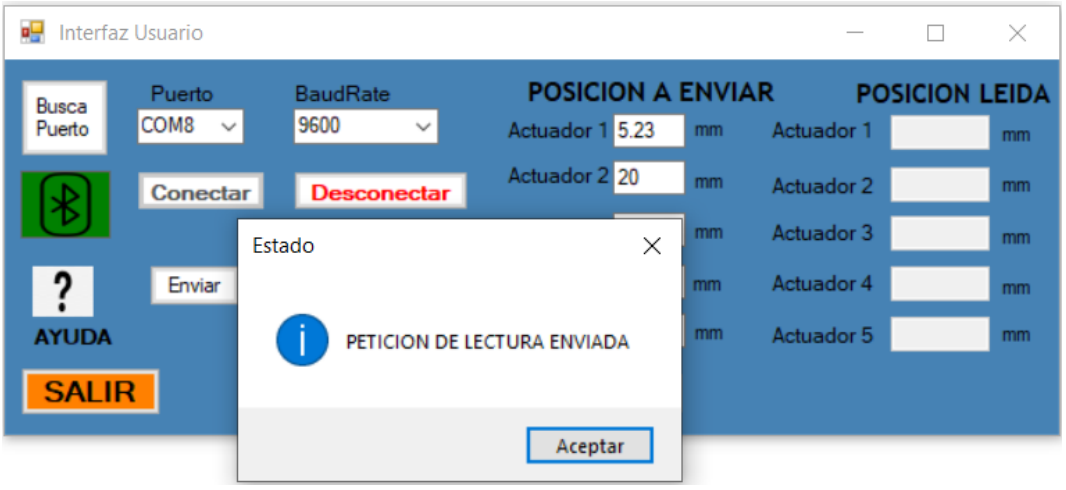

Ilustración 45: Mensaje de envío de Petición de Lectura

En el monitor serial del microcontrolador, igual que en casos anteriores, nos detalla que tipo de petición hemos recibido, y esta al ser de lectura, comienza la ejecución para recoger los datos. Una vez recogidos se envía la respuesta

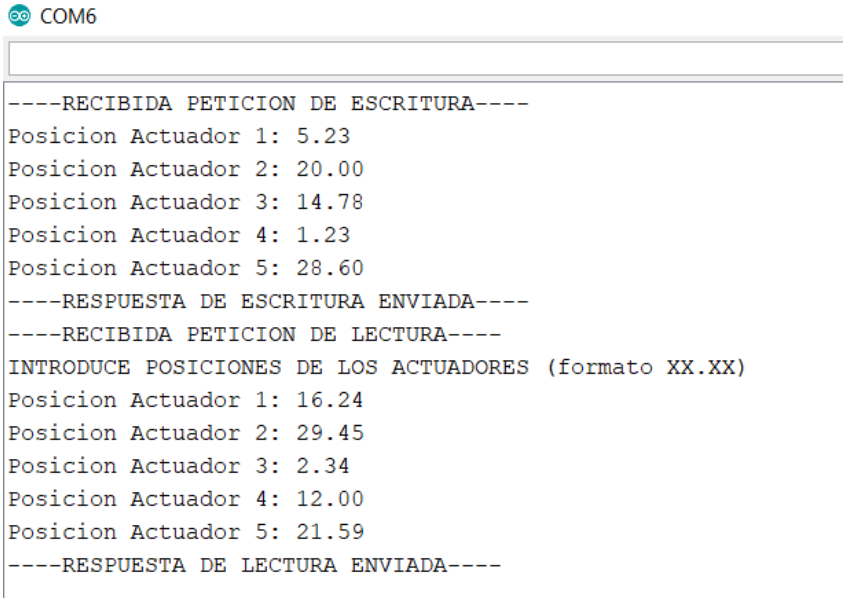

Ilustración 46: Monitor Serial tras ciclo de Petición de Lectura

Volvemos a la interfaz y vemos que la Respuesta de Lectura es correcta.

Ilustración 47: Confirmación de recepción de respuesta de lectura en interfaz

Después de confirmar el cuadro de estado, se nos muestran los valores leídos de posición de los actuadores:

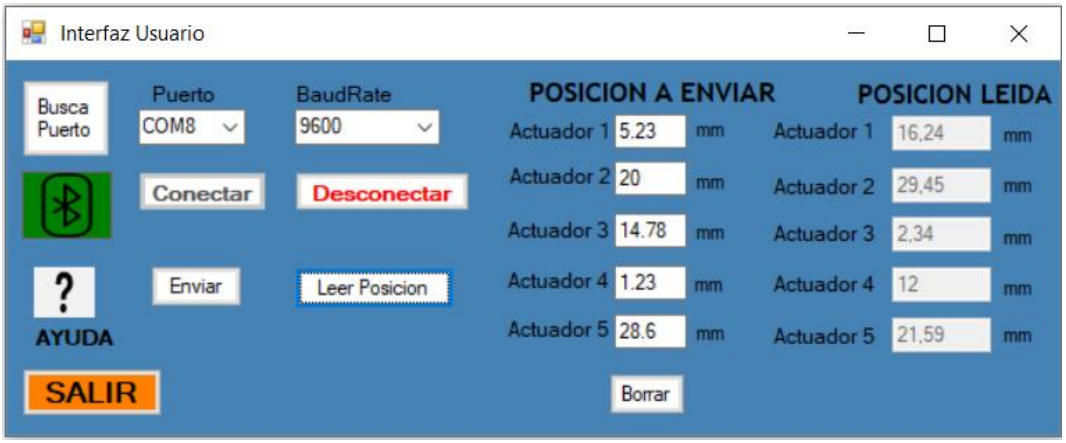

Ilustración 48:Comunicación completa después de enviar y leer posición

5.1.2. Microcontrolador detecta un error en la petición recibida

Iniciamos una secuencia en la que queremos enviar una petición de lectura.

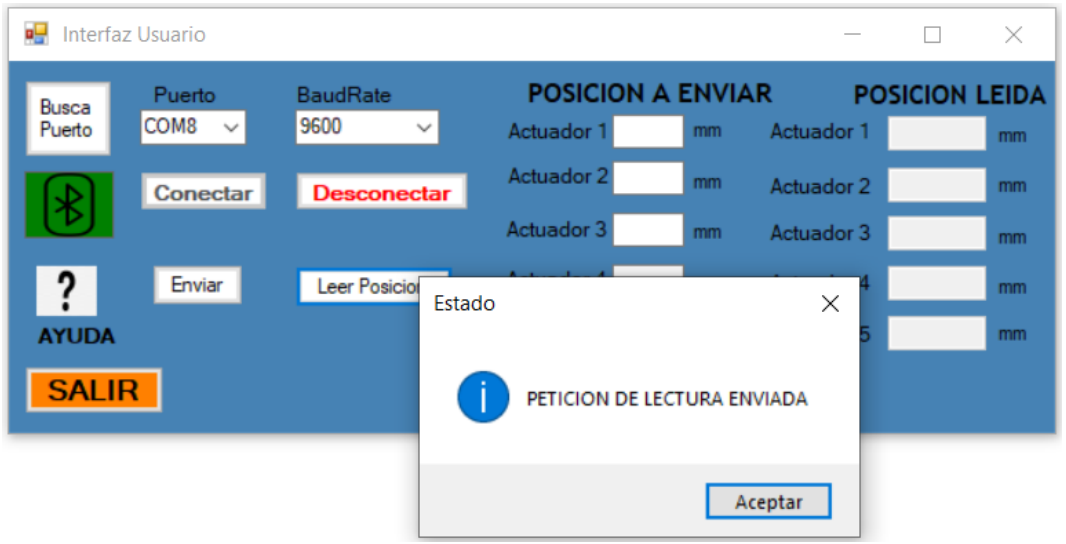

Ilustración 49: Mensaje de envío de petición de lectura

Arduino detecta el error, he inmediatamente se inicia la secuencia de notificación del error.

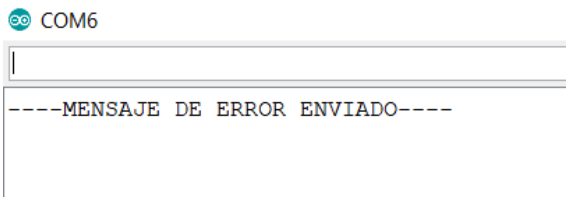

Ilustración 50: Error detectado y envío de mensaje de error

Desde la aplicación, se nos notifica que se ha recibido el error

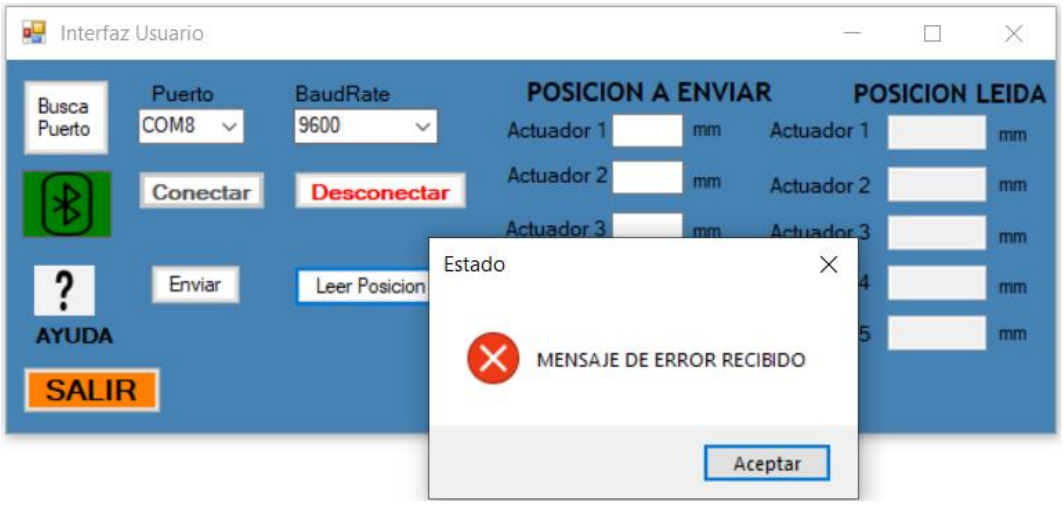

Ilustración 51: Notificación de mensaje de error recibido

Se volvería a enviar la petición de lectura, igual que la ilustración 49. Finalmente al continuar el error, se procede a poner fin a la ejecución del programa tanto en Arduino como en la aplicación de Visual Studio

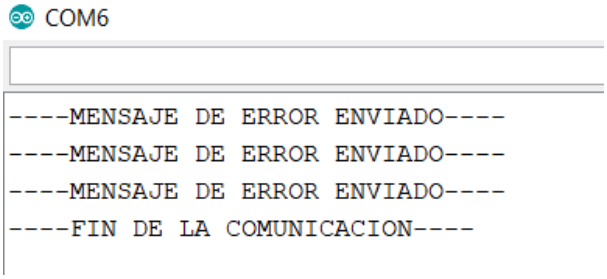

Ilustración 52: Mensajes de error y fin de comunicación en Arduino

Ilustración 53: Mensaje de fin de comunicación después de llegar al número máximo de mensajes de error

5.1.3. Aplicación detecta un error en la respuesta recibida

Vamos a forzar uno de los campos del mensaje a que no sea el diseñado, para que se genere la rutina de error.

Se inicia la rutina con una petición de lectura

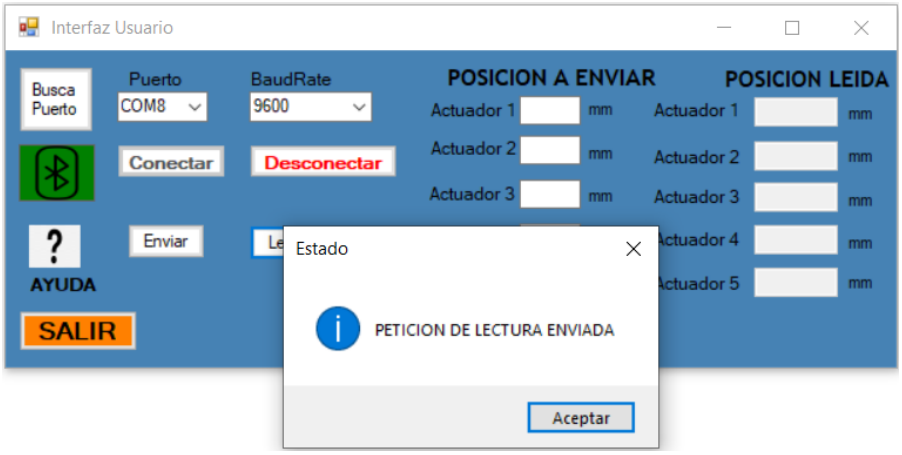

Ilustración 54: Mensaje de envío de petición de lectura

El microcontrolador recibe la petición y solicita datos al usuario, una vez los ha recogido envía la respuesta de lectura

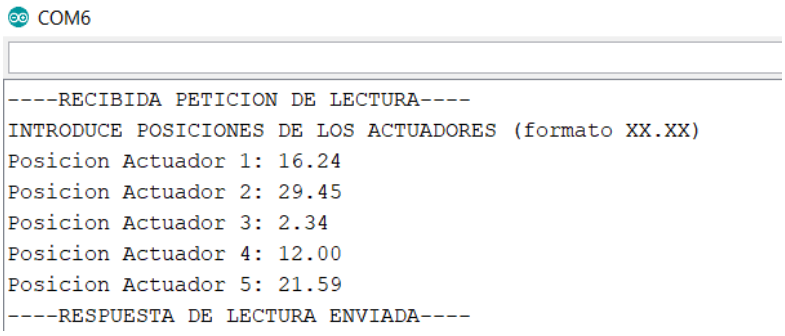

Ilustración 55: Recogida de datos y envío de Respuesta de Lectura

En la aplicación se detecta un error, y se envía el mensaje de error para notificar

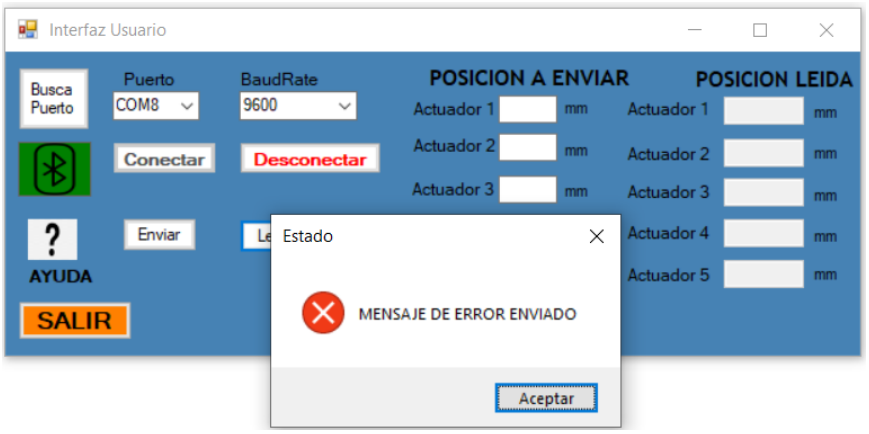

Ilustración 56: Aviso de mensaje de error enviado

Del lado del Arduino, recibimos el mensaje de error y procedemos a enviar de nuevo la petición de lectura con los valores que introdujo el usuario:

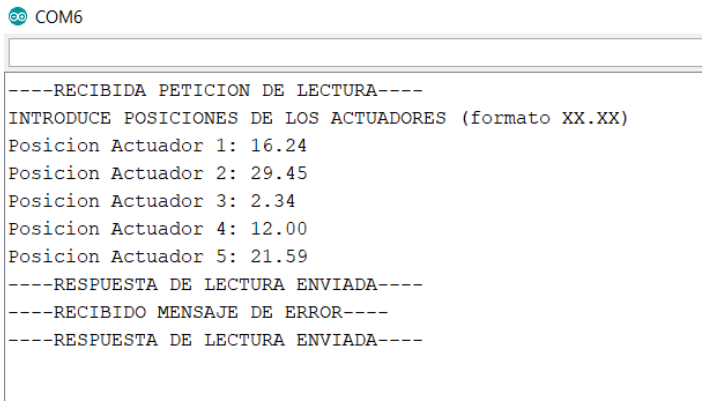

Ilustración 57: Recepción del error y reenvío de la respuesta de Lectura

Después de realizar esta secuencia el número máximo establecido, la aplicación nos informa y en segundo lugar finaliza la comunicación (Ilustración 59)

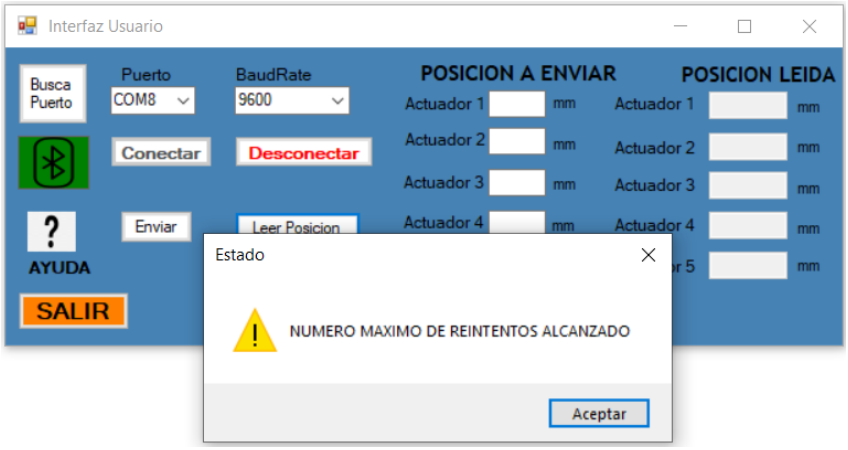

Ilustración 58: Mensaje de aviso de máximo reintentos alcanzados

Al no ser una trama correcta, en ningún momento nos muestra la aplicación los datos introducidos ya que estos pueden estar corruptos

Ilustración 59: Mensaje de fin de comunicación

En el lado del Arduino, sucede lo mismo. Sabiendo que se han recibido 3 mensajes de error, se pone fin a la comunicación.

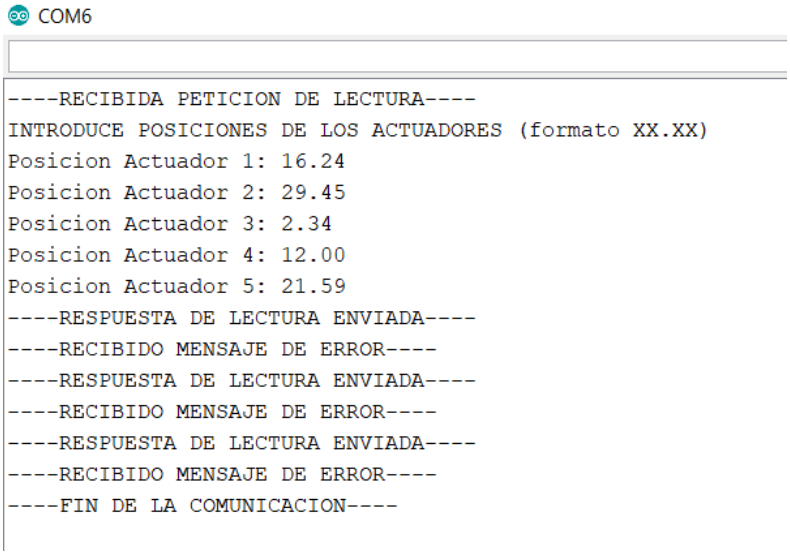

Ilustración 60: Fin de comunicación tras llegar al máximo número de reenvíos

5.2. RESULTADOS OBTENIDOS USANDO TMS320F28069M

Para conectar el módulo al circuito F2806M se ha usado una tira de cable con pines directa.

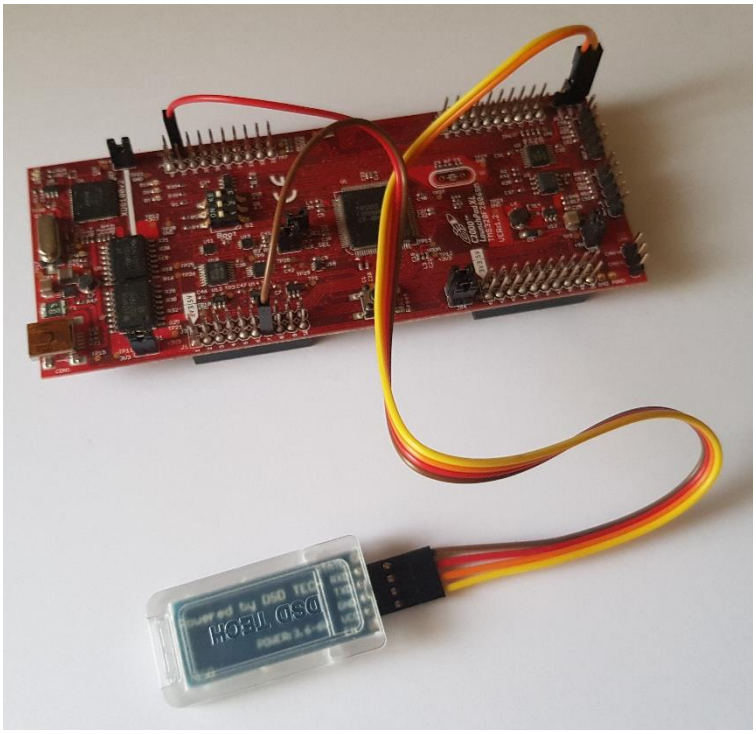

Ilustración 61: Prototipo usado con F28069M

5.2.1. Comunicación completa

Iniciamos la aplicación y decimos que datos de posición queremos enviar, una vez estos son los adecuados seleccionamos la opción de enviar y si el envío ha sido satisfactorio se nos notificara el mensaje de petición de escritura enviada (Ilustración 63)

Ilustración 62: Datos introducidos para enviar

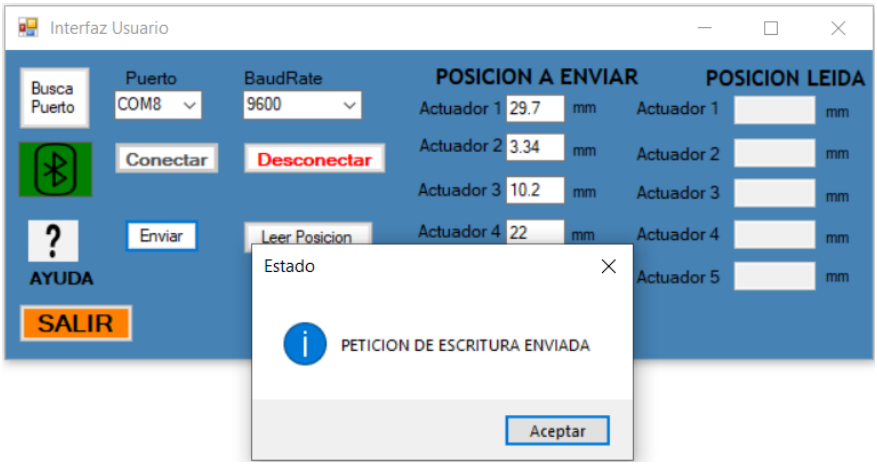

Ilustración 63: Mensaje de envío de Petición de Escritura

En este caso desde el monitor serial que nos ofrece la propia IDE de Code Composer Studio, tenemos una ventana de expresiones donde podemos monitorizar las variables que queramos, por lo que vamos a hacer uso de esta función para poder ver las variables recibidas desde la interfaz de usuario en PC. De esta manera en el monitor serial clásico vamos a recibir las consignas con el estado de la comunicación.

Como se puede observar, el monitor serie que incorpora este IDE, capta el resto de los campos de los registros de los mensajes de la cola FIFO, es por eso que aparecen esos caracteres al inicio de la transmisión, pero estos no afectan al funcionamiento general del programa.

Ilustración 64: Resultado del envío de petición de escritura en F28069M

Al procesar la trama y es correcta se envía la respuesta de escritura a la interfaz y tenemos el mensaje confirmación de respuesta de escritura.

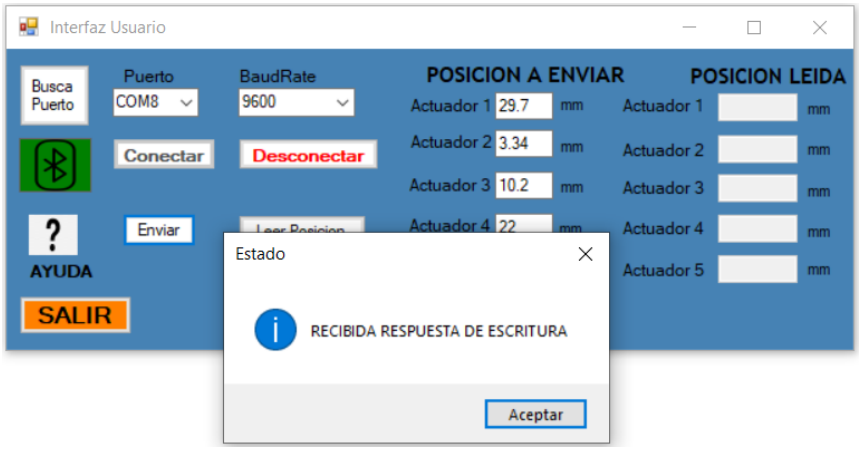

Ilustración 65: Mensaje de confirmación de respuesta de escritura

Ahora volvemos a tener el programa en el punto inicial, con lo que podemos realizar una nueva petición, vamos a realizar una petición de lectura.

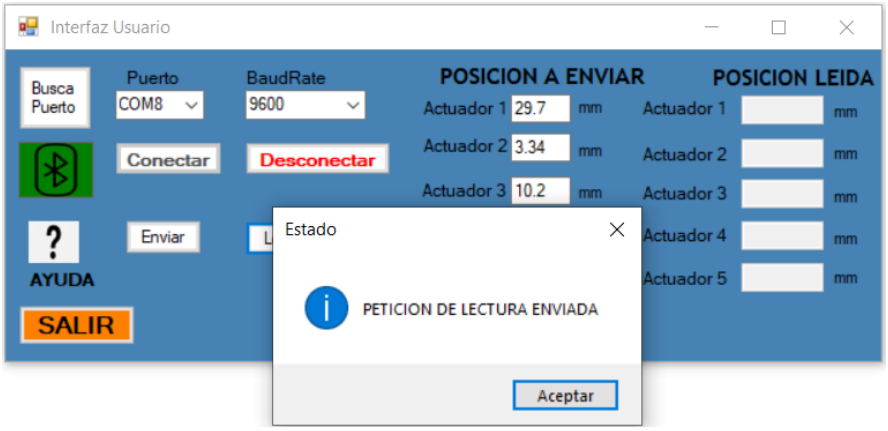

Ilustración 66: Mensaje de confirmación de envío de petición de lectura

En el microcontrolador de recibe la petición de lectura y se ejecuta la rutina para solicitar datos al usuario, una vez se introducen todos los datos de los actuadores se procede a enviar la respuesta de lectura

```
COM4
Aí@UÂ A#33A°AzáHÕ
-----RECIBIDA PETICION DE ESCRITURA-----
-----RESPUESTA DE ESCRITURA ENVIADA-----
-----RECIBIDA PETICION DE LECTURA-----
INTRODUCE POSICIONES DE LOS ACTUADORES (formato XX.XX)
Posicion Actuador 1: 02.34
Posicion Actuador 2: 21.59
Posicion Actuador 3: 12.00
Posicion Actuador 4: 28.74
Posicion Actuador 5: 17.20
-----RESPUESTA DE LECTURA ENVIADA-----
```

Ilustración 67: Estado de monitor serial después de procesar la petición de lectura

Una vez recibida la confirmación de que se ha recibido la respuesta de lectura correctamente se presentan los datos leídos (Ilustración 69)

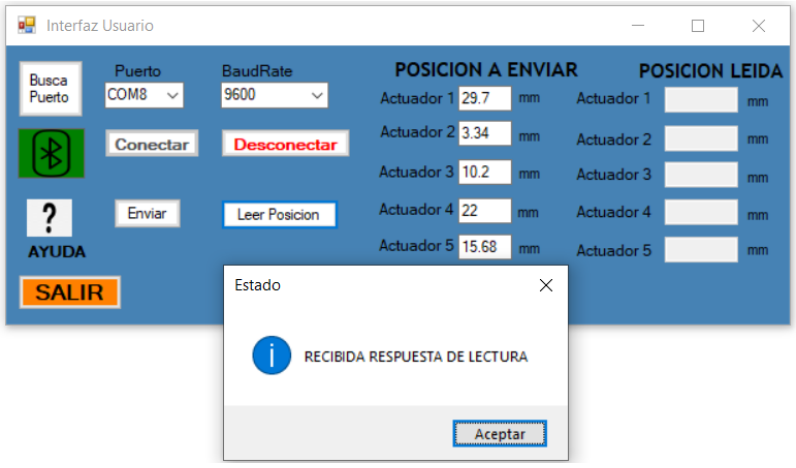

Ilustración 68: Ventana de confirmación de respuesta de lectura

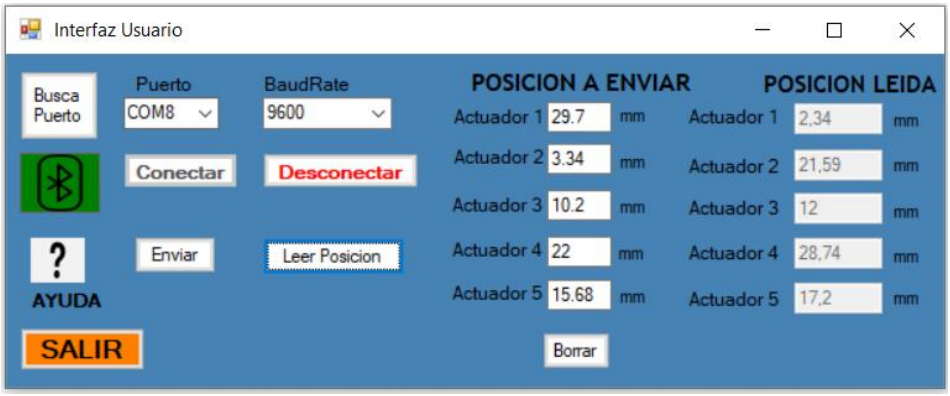

Ilustración 69: Comunicación después de enviar y leer posiciones

5.2.2. Microcontrolador detecta un error en la petición recibida

Para poder ver cómo se comporta el programa, ante esta posible situación, voy a introducir en una de las peticiones un campo incorrecto y ver así como se comporta el programa.

Vamos a iniciar una petición de lectura de datos hacia F28069M

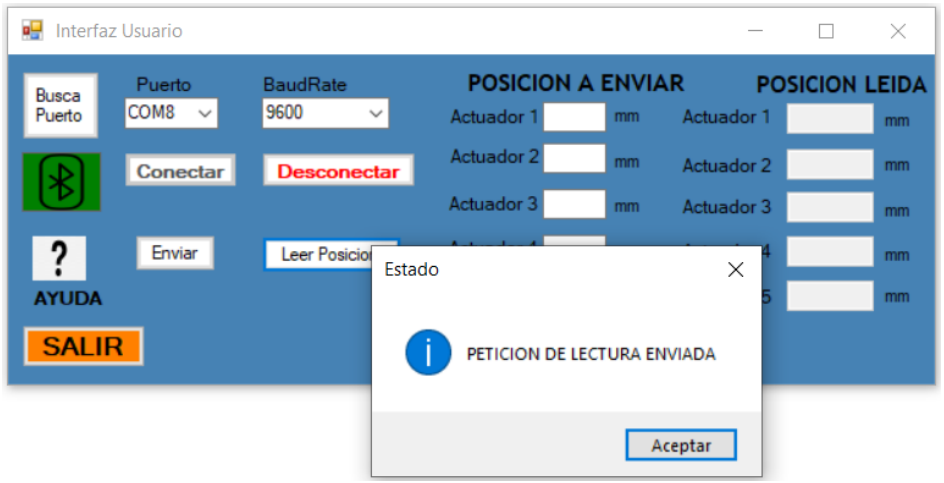

Ilustración 70: notificación de envío en interfaz de petición de lectura

Cuando se recibe el mensaje en el microcontrolador se detecta el error y se inicia la secuencia de notificación de este:

Ilustración 71: Estado del microcontrolador cuando recibe por primera vez una trama incorrecta

Se recibe en la interfaz el mensaje de error, y se procede internamente con el reenvío de la petición de lectura, por lo que se nos volvería a mostrar el mensaje de envío (Ilustración 70)

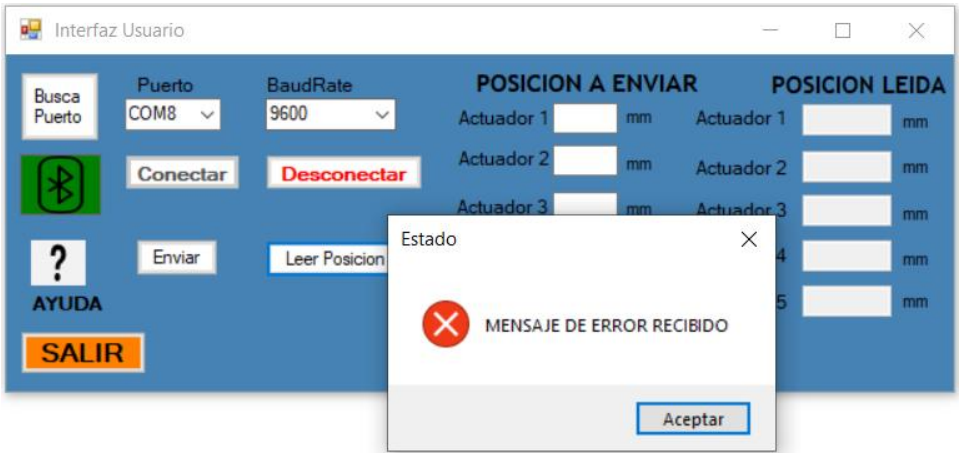

Ilustración 72: Notificación de error tras envío de petición de lectura

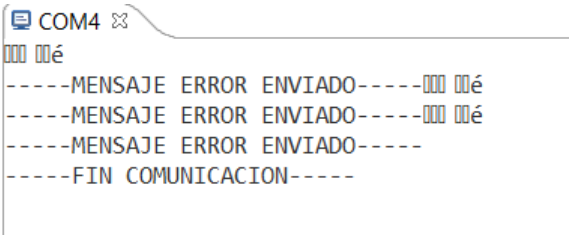

Ilustración 73: Estado del microcontrolador después de recibir una petición incorrecta

Ilustración 74: Mensaje de fin de comunicación tras finalizar reenvío sin éxito

5.2.3. Aplicación detecta un error en la respuesta recibida

Para poder ver que ocurre si se diera este caso, voy a modificar la instrucción de respuesta de lectura en el microcontrolador, poniendo un campo del mensaje que no es correcto.

Desde la interfaz realizamos una petición de lectura.

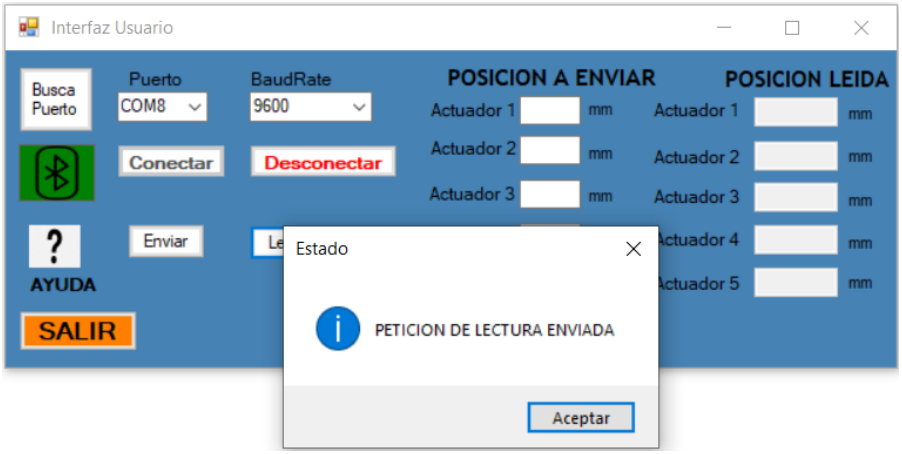

Ilustración 75: Petición de lectura realizada desde la interfaz

En la placa F28069M, se recibe la petición y se gestiona su respuesta, para ello solicita datos al usuario y genera la respuesta de lectura.

```
COM4
-----RECIBIDA PETICION DE LECTURA-----
INTRODUCE POSICIONES DE LOS ACTUADORES (formato XX.XX)
Posicion Actuador 1: 02.34
Posicion Actuador 2: 21.59
Posicion Actuador 3: 12.00
Posicion Actuador 4: 28.74
Posicion Actuador 5: 17.20
-----RESPUESTA DE LECTURA ENVIADA-----
```

Ilustración 76: Ejecución en la placa F28069M

La interfaz, recibe la respuesta, pero esta es errónea, por lo que envía el mensaje de error para notificarlo

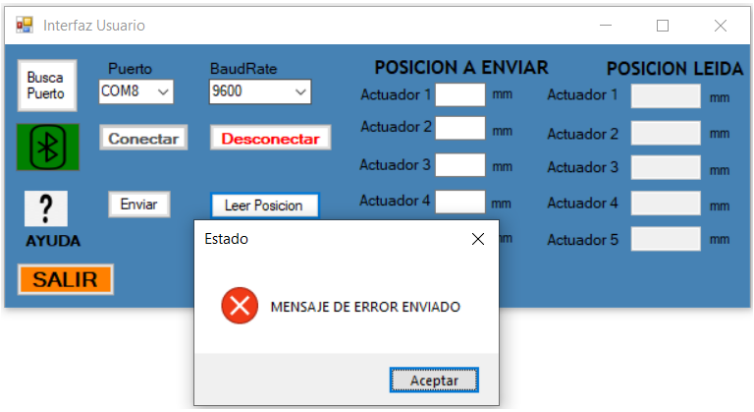

Ilustración 77: Envío de mensaje de error desde interfaz

Se recibe el mensaje de error y se reenvía la respuesta de lectura, en esta no se solicitan de nuevo datos ya que estos ya los tenemos almacenados.

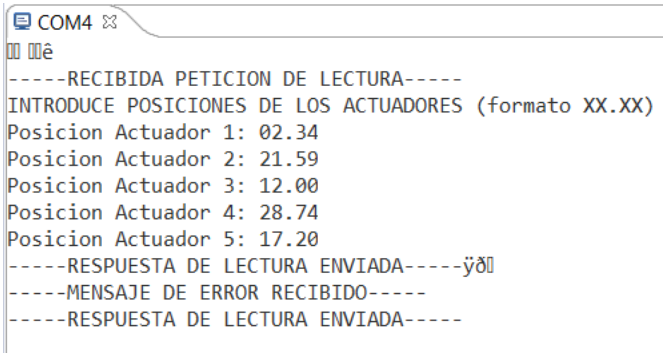

Ilustración 78: Inicio de rutina de reenvío

después de intentar varios envíos finalmente se llega al máximo soportado. Se nos notifica del tipo de error por el que se va a finalizar la comunicación (Ilustraciones 79 y 80)

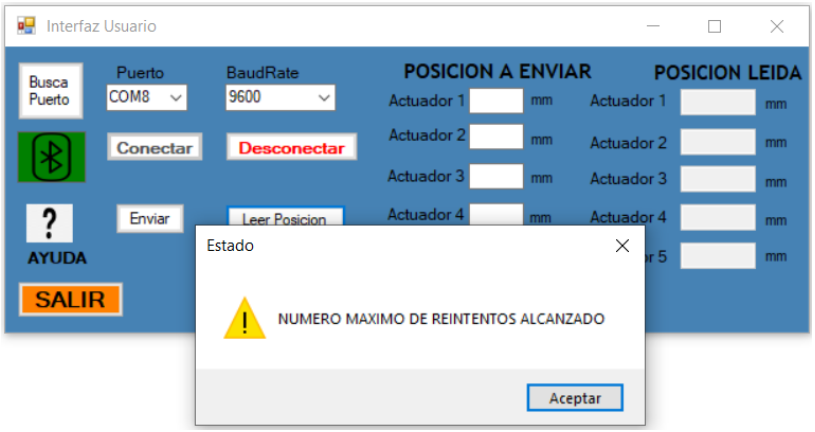

Ilustración 79: Notificación del máximo de reintentos alcanzados

Ilustración 80: Mensaje de fin de comunicación en la interfaz

En el monitor serial del Microcontrolador, vemos que ocurre lo mismo, se nos notifica el reenvío de la respuesta y los mensajes de error, hasta que llegados el máximo permitido se pone fin a la comunicación.

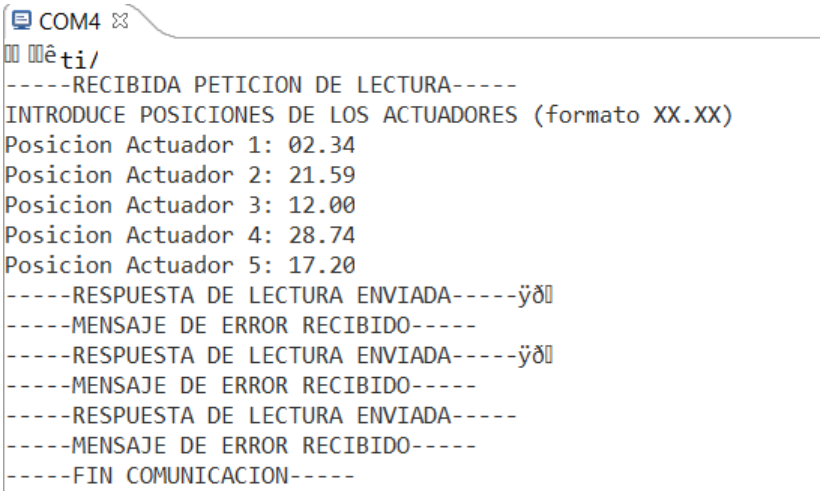

Ilustración 81: Estado del microcontrolador después de la rutina de reenvío fallida

CAPITULO 6: COSTES DEL PROYECTO

Para poder conocer la viabilidad de cualquier proyecto es necesario realizar un estudio económico. Tendremos que valorar los distintos recursos utilizados para poder obtener una estimación de este.

Para la realización del proyecto se han utilizado además del trabajo personal, las herramientas de tipo hardware y software.

6.1. RECURSOS HARDWARE

Dentro de los recursos de tipo hardware tenemos el ordenador, el módulo Bluetooth HC-06, Arduino UNO REV 3, protoboard. Además de la placa LAUNCHXL-F28069M, prestada por el ITAP para la realización del presente trabajo.

El desglose de los precios de cada dispositivo se recoge en la siguiente tabla:

RECURSOS HARDWARE	PRECIO
Ordenador Portátil ASUS N552VX-363T	699€
Modulo Bluetooth HC-06	9,99€
Arduino UNO R3 Elegoo	9,99€
Protoboard	4,99€
LAUNCHXL-F28069M	48€
TOTAL	**771.97€**

Tabla 22: Costes Recursos Hardware

6.2. RECURSOS SOFTWARE

Como recursos software tendríamos los programas informáticos utilizados para el desarrollo, que son parte igual de importante que los elementos tangibles descritos anteriormente. Todos los recursos que se han utilizado para el proceso son gratuitos, por lo que de cara al desarrollo es punto muy favorable en el proyecto.

RECURSOS SOFTWARE	PRECIO
Visual Studio 2019	Gratuito
Arduino IDE	Gratuito
Code Composer Studio	Gratuito
Microsoft Office	Gratuito
Adobe Acrobat Reader DC	Gratuito
Sistema operativo Windows 10	Gratuito (incluido con el PC)
TOTAL	**Gratuito**

Tabla 23: Costes Recursos Software

6.3. COSTE PERSONAL

Cuando se emprende un proyecto se necesita a una persona que lo lleve a cabo, el coste asociado a la persona o grupo de personas que realizan dicho proyecto es el coste personal.

Suponiendo que el sueldo medio para un ingeniero es de 34.000€, y que la cotización a la seguridad social la cual se hará cargo la empresa es de un 32% (se incluye contingencias comunes, desempleo, formación profesional, FOGASA, accidentes de trabajo o enfermedad profesional), el coste total asciende a 44.880€

Acuerdo a «BOE» núm. 30, de 4 de febrero de 2020, páginas 10498 a 10504 [16], se establece que las horas efectivas reales a partir del 1 de enero de 2021 son de 1720h.

Con estos dos últimos datos podemos calcular el coste hora:

Coste/hora = 48.880€/1720h=26,09 euros por hora de trabajo

Con este último dato necesitamos saber las horas de trabajo que conlleva la realización de dicho proyecto, para ello se ha elaborado la tabla siguiente:

TRABAJO REALIZADO	HORAS
Estudio del problema	15
Recopilación de información	80
Aprendizaje de programación en Visual Studio	15
Programación de los microcontroladores	80
Realización de la interfaz en Visual Studio	30
Pruebas	35
Realización de la memoria	90
TOTAL HORAS	**345**

Tabla 24: Aproximación de horas de trabajo empleadas

Por lo tanto, para este proyecto, el coste personal seria:

355 horas x 26,09 euro/hora= 9002,09 euros

CAPITULO 7: CONCLUSIONES

Las características que nos proporcionan las redes inalámbricas hoy en día, las hacen una de las mejores alternativas frente a las conexiones cableadas tradicionales. Podemos aprovechar todas las ventajas que nos ofrecen, como son, el ahorro de material, la agilidad de la red, la multiplicidad de conexiones y la mejora en la realización de circuitos complejos. Además de las múltiples ventajas tendremos que conocer los aspectos débiles para poder reducir la probabilidad de fallos.

En el protocolo que he diseñado se han establecido el conjunto de reglas necesarias para la correcta ejecución de la aplicación. Para ello se ha realizado un estudio para saber que funciones se tienen que realizar en cada nivel funcional para garantizar que el método escogido es el más adecuado para ello. Esto incluye la creación de las reglas que dictan como se inicia, mantiene y finaliza el dialogo. Otro de los aspectos que interviene es el formato de los mensajes que se intercambian, hay que tener claro tanto el tipo de dato que se intercambia como su valor. Con ello he logrado aprender las diferentes fases para crear un protocolo de comunicación.

Para poder desarrollar los objetivos planteados ha sido necesario realizar la programación en las tres plataformas distintas utilizadas. He tenido que profundizar en su uso, para lograr su correcta implementación. Estas son principalmente Visual Studio y el IDE de Texas Instruments, las cuales nunca había trabajado con ellas y en menor medido Arduino, ya que si lo conocía. He aprendido a buscar la mejor solución disponible, cuando se me presentó alguna limitación ya sea hardware o software, adaptándome a ellas y buscando exprimir al máximo las posibilidades que nos ofrecen los dispositivos y sus herramientas.

En las distintas fases de validación he podido comprobar los diferentes fallos que han surgido durante el desarrollo. Los cuales me han servido como método de mejora y progreso durante la realización del trabajo. Cabe destacar la mejora en búsqueda de soluciones y alternativas, para poder resolver los problemas por uno mismo.

Tras la conclusión del TFG, se ha podido cumplir el objetivo de estudio, en concreto, se ha demostrado que el módulo Bluetooth, HC-06 es adecuado para poder dotar de comunicación inalámbrica a la plataforma de rehabilitación neuromotora robotizada. Además, con la interfaz creada para el computador de gestión o supervisión, se hace posible enviar y recibir instrucciones hacia los microcontroladores para el control de los prototipos.

BIBLIOGRAFIA

[1] ITAP, «ROBHAND, UN EXOESQUELETO DE MANO PARA LA REHABILITACION NEUROMOTORA APLICANDO TERAPIAS ACTIVAS Y PASIVAS,» de *Actas de las XXXIX Jornadas de Automatica*, Badajoz, 2018.

[2] Microsoft, «Microsoft/Visual Studio,» [En línea]. Available: https://visualstudio.microsoft.com/es/vs/features/net-development/. [Último acceso: 2021].

[3] Microsoft, «Documentacion de Windows Forms,» 2020. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/desktop/winforms/overview/?view=netdesktop-5.0. [Último acceso: 2021].

[4] M. Verde, PIC Microcontrollers Programming in Basic With Examples, 2010.

[5] Arduino, «Arduino Store,» [En línea]. Available: https://store.arduino.cc/arduino-uno-rev3. [Último acceso: 2021].

[6] ATMEL, «ATMEL ATmega328 Datasheet,» 2016.

[7] T. Instruments, «LAUNCHXL-F28069M Overview,» 2019.

[8] IONOS, «¿Qué es Bluetooth? Toda la información sobre el estándar inalámbrico,» 2020. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-bluetooth/. [Último acceso: 2021].

[9] DSD, «DSD TECH Oficial Website,» 2017. [En línea]. Available: http://www.dsdtech-global.com/search/label/HC-06. [Último acceso: 2021].

[10] Microsoft, «Microsfot Docs,» [En línea]. Available: https://docs.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2019. [Último acceso: 2021].

[11] R. Concepcion, «rjconcepcion,» 2020. [En línea]. Available: https://www.rjconcepcion.com/podcast/que-lenguaje-de-programacion-usa-arduino/. [Último acceso: 2021].

[12] ArduWiki, «SoftwareSerial,» 2019. [En línea]. Available: https://arduwiki.perut.org/index.php/SoftwareSerial. [Último acceso: 2021].

[13] TutorialsPoint, «C-Unions,» [En línea]. Available: https://www.tutorialspoint.com/cprogramming/c_unions.htm. [Último acceso: 2021].

[14] Amlendra, «Little endian and Big endian Concept,» 2016. [En línea]. Available: https://aticleworld.com/little-and-big-endian-importance/. [Último acceso: 2021].

[15] T. Instruments, «TMS320F28069M Kit Quick Start Guide,» 2014.

[16] BOE, «Resolución de 27 de enero de 2020, de la Dirección General de Trabajo, por la que se registra y publica el Acta de modificación del Convenio colectivo de Fertiberia, SA,» 2020.

ESCUELA DE INGENIERÍAS INDUSTRIALES

ANEXOS

ANEXO I: CODIGO VISUAL STUDIO

```
///-------------------------------------------------///
/// TRABAJO FIN DE GRADO                            ///
/// INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA  ///
/// UNIVERSIDAD DE VALLADOLID                       ///
///                                                 ///
///  Implementación de un entorno de comunicación   ///
///  Bluetooth basado en el módulo HC-06            ///
///                                                 ///
/// CODIGO  - VISUAL STUDIO                         ///
///                                                 ///
/// AUTOR - DAVID MUCIENTES SAN JOSÉ                ///
///-------------------------------------------------///

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Reflection;
using System.IO;
using System.Runtime.Serialization;
using System.Globalization;

namespace P1
{
    public partial class Hand3 : Form
    {
        public Hand3()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
        }

        void getAvailablePorts()
        {
            string[] ports = SerialPort.GetPortNames();
            cboPuerto.Items.AddRange(ports);
        }
        private void btnBuscaPuertos_Click(object sender, EventArgs e)
        {
            cboPuerto.Items.Clear();
            getAvailablePorts();
        }
        private void btnConectar_Click(object sender, EventArgs e)
        {
            if (cboPuerto.Text == "" || cboBaud.Text == "")
            {
                MessageBox.Show("Puerto o Baudios No seleccionados", "Avertencia",
                MessageBoxButtons.OK,MessageBoxIcon.Exclamation);
            }
            else
            {
                serialBT.PortName = cboPuerto.Text;
                serialBT.BaudRate = Convert.ToInt32(cboBaud.Text);
                serialBT.Parity = Parity.None;
                serialBT.DataBits = 8;
                serialBT.StopBits = StopBits.One;
                serialBT.DataReceived += serialBT_DataReceived;
                serialBT.DataReceived += new SerialDataReceivedEventHandler(
                serialBT_DataReceived);
                serialBT.ReceivedBytesThreshold = 6;
                try
                {
                    serialBT.Open();
                    pictureBoxBT.BackColor = Color.Green;
                    btnConectar.Enabled = false;
                    btnDesconectar.Enabled = true;
                    btnPideDatos.Enabled = true;
```

```
                    btnEnviar.Enabled = true;

                    txtAct1Env.Enabled = true;
                    txtAct2Env.Enabled = true;
                    txtAct3Env.Enabled = true;
                    txtAct4Env.Enabled = true;
                    txtAct5Env.Enabled = true;

                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                    pictureBoxBT.BackColor = Color.Red;
                    MessageBox.Show("Error al Abrir el puerto","Error",
                    MessageBoxButtons.OK,MessageBoxIcon.Error);
                }

            }
        }
        private void btnDesconectar_Click(object sender, EventArgs e)
        {
            if (serialBT.IsOpen==true)
            {
                serialBT.Close();
                pictureBoxBT.BackColor = Color.Red;
                serialBT.Dispose();

                Reset();
                btnDesconectar.Enabled = false;
                btnConectar.Enabled = true;
                btnEnviar.Enabled = false;
                btnPideDatos.Enabled = false;
                txtAct1Env.Enabled = false;
                txtAct2Env.Enabled = false;
                txtAct3Env.Enabled = false;
                txtAct4Env.Enabled = false;
                txtAct5Env.Enabled = false;
                txtAct1Rec.Clear();
                txtAct2Rec.Clear();
                txtAct3Rec.Clear();
                txtAct4Rec.Clear();
                txtAct5Rec.Clear();

            }
        }
        private void Desconexion()
        {
            if (serialBT.IsOpen == true)
            {
                serialBT.Close();
                pictureBoxBT.BackColor = Color.Red;
                serialBT.Dispose();

                Reset();
                btnDesconectar.Enabled = false;
                btnConectar.Enabled = true;
                btnEnviar.Enabled = false;
                btnPideDatos.Enabled = false;
                txtAct1Env.Enabled = false;
                txtAct2Env.Enabled = false;
                txtAct3Env.Enabled = false;
                txtAct4Env.Enabled = false;
                txtAct5Env.Enabled = false;
                txtAct1Rec.Clear();
                txtAct2Rec.Clear();
                txtAct3Rec.Clear();
                txtAct4Rec.Clear();
                txtAct5Rec.Clear();
                Close();

            }
        }
        private void btt_Salir_Click(object sender, EventArgs e)
```

```
144         {
145             Close();
146         }
147         private void btt_Enviar_Click(object sender, EventArgs e)
148         {
149             try
150             {
151                 if (serialBT.IsOpen == true)
152                 {
153                         CompruebaDatos();
154                     if (ok == true)
155                     {
156                         PeticionEscritura();
157                         btnClean.Enabled = true;
158                     }
159 
160                 }
161             }
162             catch (Exception ex)
163             {
164                 MessageBox.Show(ex.Message);
165             }
166 
167         }
168         private void btnClean_Click(object sender, EventArgs e)
169         {
170             txtAct1Env.Clear();
171             txtAct2Env.Clear();
172             txtAct3Env.Clear();
173             txtAct4Env.Clear();
174             txtAct5Env.Clear();
175 
176         }
177         public void Reset()
178         {
179             txtAct1Env.Clear();
180             txtAct2Env.Clear();
181             txtAct3Env.Clear();
182             txtAct4Env.Clear();
183             txtAct5Env.Clear();
184         }
185         private void btnPideDatos_Click(object sender, EventArgs e)
186         {
187             if (serialBT.IsOpen == true)
188             {
189                 txtAct1Rec.Clear();
190                 txtAct2Rec.Clear();
191                 txtAct3Rec.Clear();
192                 txtAct4Rec.Clear();
193                 txtAct5Rec.Clear();
194                 PeticionLectura();
195             }
196             else
197             {
198                 MessageBox.Show("Establezca Conexion con Microcontrolador Primero",
                    "Advertencia",MessageBoxButtons.OK,MessageBoxIcon.Exclamation);
199             }
200         }
201         private void picBoxAyuda_Click(object sender, EventArgs e)
202         {
203             MessageBox.Show("Pulse Buscar Puerto para seleccionar Puerto COM",
                "Cuadro de Ayuda", MessageBoxButtons.OK,MessageBoxIcon.Information);
204             MessageBox.Show("Boton Conectar para establecer/cancelar conexion con el
                equipo", "Cuadro de Ayuda", MessageBoxButtons.OK, MessageBoxIcon.
                Information);
205             MessageBox.Show("Boton Enviar/Borrar para enviar datos/limpiar contenido
                de las casillas", "Cuadro de Ayuda", MessageBoxButtons.OK, MessageBoxIcon
                .Information);
206             MessageBox.Show("Boton Pedir Datos para solicitar datos al
                microcontrolador", "Cuadro de Ayuda", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
207 
208         }
```

```
209
210         private void serialBT_DataReceived(object sender, SerialDataReceivedEventArgs
             e)
211         {
212                 while (serialBT.BytesToRead > 0)
213                 {
214                     int bytes = serialBT.BytesToRead;
215                     byte[] buffer = new byte[bytes];
216 
217                     serialBT.Read(buffer, 0, bytes);
218                     Console.WriteLine(bytes);
219                     Console.WriteLine(BitConverter.ToString(buffer, 0, buffer.Length
                     ));
220 
221                     if (buffer[1] == tam1 && buffer.Length == 11 && buffer[buffer.
                     Length-1]==Fin)
222                     {
223                         administraMSGrecibido(bytes, buffer);
224                     }
225                     else if (buffer[1] == tam2 && buffer.Length == 31 && buffer[
                     buffer.Length - 1] == Fin)
226                     {
227                         administraMSGrecibido(bytes, buffer);
228                     }
229                     else if (buffer[1] == tamE && buffer.Length == 6 && buffer[buffer
                     .Length - 1] == Fin)
230                     {
231                         administraMSGrecibido(bytes, buffer);
232                     }
233                     else
234                     {
235                         int bytesAux = serialBT.BytesToRead;
236                         byte[] aux = new byte[bytesAux];
237                         serialBT.Read(aux, 0, bytesAux);
238                         Console.WriteLine(bytesAux);
239                         Console.WriteLine(BitConverter.ToString(aux, 0, aux.Length));
240                         administraMSGrecibido(bytes + bytesAux, Combine(buffer, aux));
241                     }
242 
243             }
244         }
245 
246         String sAct1;
247         String sAct2;
248         String sAct3;
249         String sAct4;
250         String sAct5;
251 
252         int msgUltimo;
253         int envioError = 0;
254         int contError = 0;
255 
256         private void administraMSGrecibido(int Nbytes, byte[] bRecib)
257         {
258             if (Nbytes == 31)
259                 {
260                     if (bRecib[0] == Cabecera)
261                     {
262                     //Console.WriteLine("Cabecera OK");
263 
264                     if (bRecib[Nbytes - 1] == Fin)
265                     {
266                         //Console.WriteLine("Fin OK");
267 
268                         if (bRecib[1] == tam2)
269                         {
270                             //Console.WriteLine("Respuesta Lectura");
271 
272                             if (bRecib[2] == resLec)
273                             {
274                                 //Console.WriteLine("COD Respuesta Lectura Correcto");
275 
276                                 if (bRecib[3] == varAct1)
```

```
277                                 {
278                                     //Console.WriteLine("varPulgar Correcto");
279
280                                     if (bRecib[8] == varAct2)
281                                     {
282                                         //Console.WriteLine("varIndice Correcto");
283
284                                         if (bRecib[13] == varAct3)
285                                         {
286                                             //Console.WriteLine("varMedio Correcto");
287
288                                             if (bRecib[18] == varAct4)
289                                             {
290                                                 //Console.WriteLine("varAnular
                                                    Correcto");
291
292                                                 if (bRecib[23] == varAct5)
293                                                 {
294                                                     //Console.WriteLine("varMenique
                                                        Correcto");
295
296                                                     if (bRecib[28] == errNoerror)
297                                                     {
298                                                         //Console.WriteLine("COD_ERR
                                                            Correcto");
299
300                                                         if (bRecib[29] ==
                                                            CalculaChecksum(Nbytes,
                                                            bRecib))
301                                                         {
302
                                                             //Console.WriteLine("ChekS
                                                             um OK");
303                                                          MessageBox.Show(
                                                             "RECIBIDA RESPUESTA DE
                                                             LECTURA","Estado",
                                                             MessageBoxButtons.OK,
                                                             MessageBoxIcon.
                                                             Information);
304
305                                                          byte[] aAct1R = new byte[
                                                             4];
306                                                          aAct1R[0] = bRecib[7];
307                                                          aAct1R[1] = bRecib[6];
308                                                          aAct1R[2] = bRecib[5];
309                                                          aAct1R[3] = bRecib[4];
310                                                          byte[] aAct2R = new byte[
                                                             4];
311                                                          aAct2R[0] = bRecib[12];
312                                                          aAct2R[1] = bRecib[11];
313                                                          aAct2R[2] = bRecib[10];
314                                                          aAct2R[3] = bRecib[9];
315                                                          byte[] aAct3R = new byte[
                                                             4];
316                                                          aAct3R[0] = bRecib[17];
317                                                          aAct3R[1] = bRecib[16];
318                                                          aAct3R[2] = bRecib[15];
319                                                          aAct3R[3] = bRecib[14];
320                                                          byte[] aAct4R = new byte[
                                                             4];
321                                                          aAct4R[0] = bRecib[22];
322                                                          aAct4R[1] = bRecib[21];
323                                                          aAct4R[2] = bRecib[20];
324                                                          aAct4R[3] = bRecib[19];
325                                                          byte[] aAct5R = new byte[
                                                             4];
326                                                          aAct5R[0] = bRecib[27];
327                                                          aAct5R[1] = bRecib[26];
328                                                          aAct5R[2] = bRecib[25];
329                                                          aAct5R[3] = bRecib[24];
330
331                                                          float fAct1R = (float)
                                                             Math.Round(BitConverter.
```

```
                                                    ToSingle(aAct1R, 0),2);
332                                                 float fAct2R = (float)
                                                    Math.Round(BitConverter.
                                                    ToSingle(aAct2R, 0),2);
333                                                 float fAct3R = (float)
                                                    Math.Round(BitConverter.
                                                    ToSingle(aAct3R, 0),2);
334                                                 float fAct4R = (float)
                                                    Math.Round(BitConverter.
                                                    ToSingle(aAct4R, 0),2);
335                                                 float fAct5R = (float)
                                                    Math.Round(BitConverter.
                                                    ToSingle(aAct5R, 0),2);
336
337                                                 sAct1 = Convert.ToString(
                                                    fAct1R);
338                                                 this.Invoke(new
                                                    EventHandler(MostrarDatos
                                                    ));
339                                                 sAct2 = Convert.ToString(
                                                    fAct2R);
340                                                 this.Invoke(new
                                                    EventHandler(
                                                    MostrarDatos1));
341                                                 sAct3 = Convert.ToString(
                                                    fAct3R);
342                                                 this.Invoke(new
                                                    EventHandler(
                                                    MostrarDatos2));
343                                                 sAct4 = Convert.ToString(
                                                    fAct4R);
344                                                 this.Invoke(new
                                                    EventHandler(
                                                    MostrarDatos3));
345                                                 sAct5 = Convert.ToString(
                                                    fAct5R);
346                                                 this.Invoke(new
                                                    EventHandler(
                                                    MostrarDatos4));
347
348                                             }
349                                             else
350                                             {
351
                                                    //Console.WriteLine("ChekS
                                                    um Incorrecto");
352                                                 MSGerror();
353                                             }
354                                         }
355                                         else
356                                         {
357                                             //Console.WriteLine("Tamaño
                                                Incorrecto");
358                                             MSGerror();
359                                         }
360                                     }
361                                     else
362                                     {
363                                         //Console.WriteLine("varMenique
                                            Incorrecto");
364                                         MSGerror();
365                                     }
366                                 }
367                                 else
368                                 {
369                                     //Console.WriteLine("varAnular
                                        Incorrecto");
370                                     MSGerror();
371                                 }
372                             }
373                             else
374                             {
375                                 //Console.WriteLine("varMedio
```

```
                                        Incorrecto");
376                                     MSGerror();
377                                 }
378                             }
379                             else
380                             {
381                                 //Console.WriteLine("varIndice Incorrecto");
382                                 MSGerror();
383                             }
384                         }
385                         else
386                         {
387                             //Console.WriteLine("varPulgar Incorrecto");
388                             MSGerror();
389                         }
390                     }
391                     else
392                     {
393                         //Console.WriteLine("COD Respuesta Lectura
                        Incorrecto");
394                         MSGerror();
395                     }
396                 }
397                 else
398                 {
399                     //Console.WriteLine("Tamaño Incorrecto");
400                     MSGerror();
401                 }
402             }
403             else
404             {
405                 //Console.WriteLine("Fin Erroneo");
406                 MSGerror();
407             }
408         }
409             else
410             {
411                 //Console.WriteLine("Cabecera Erronea");
412                 MSGerror();
413             }
414         }
415     else if (Nbytes == 11)
416         {
417
418             if (bRecib[0] == Cabecera)
419             {
420             //Console.WriteLine("Cabecera OK");
421
422             if (bRecib[Nbytes - 1] == Fin)
423             {
424                 //Console.WriteLine("Fin OK");
425
426                 if (bRecib[1] == tam1)
427                 {
428                     //Console.WriteLine("Tamaño correcto");
429
430                     if (bRecib[2] == resEsc)
431                     {
432                         //Console.WriteLine("Respuesta Escritura Recibida");
433
434                         if (bRecib[3] == varAct1)
435                         {
436                             //Console.WriteLine("Pulgar OK");
437
438                             if (bRecib[4] == varAct2)
439                             {
440                                 //Console.WriteLine("Indice OK");
441
442                                 if (bRecib[5] == varAct3)
443                                 {
444                                     //Console.WriteLine("Medio OK");
445
446                                     if (bRecib[6] == varAct4)
```

```
447                                     {
448                                         //Console.WriteLine("Anular OK");
449
450                                         if (bRecib[7] == varAct5)
451                                         {
452                                             //Console.WriteLine("Menique OK");
453
454                                             if (bRecib[8] == errNoerror)
455                                             {
456                                                 //Console.WriteLine("COD_ERR
                                                    Correcto");
457
458                                                 if (bRecib[9] ==
                                                    CalculaChecksum(Nbytes,
                                                    bRecib))
459                                                 {
460
                                                     //Console.WriteLine("ChekS
                                                        um OK");
461                                                     MessageBox.Show(
                                                        "RECIBIDA RESPUESTA DE
                                                        ESCRITURA","Estado",
                                                        MessageBoxButtons.OK,
                                                        MessageBoxIcon.
                                                        Information);
462                                                 }
463                                                 else
464                                                 {
465
                                                     //Console.WriteLine("ChekS
                                                        um Incorrecto");
466                                                     MSGerror();
467                                                 }
468                                             }
469                                             else
470                                             {
471                                                 //Console.WriteLine("COD_ERR
                                                    Incorrecto");
472                                                 MSGerror();
473                                             }
474                                         }
475                                         else
476                                         {
477                                             //Console.WriteLine("Menique
                                                Incorrecto");
478                                             MSGerror();
479                                         }
480                                     }
481                                     else
482                                     {
483                                         //Console.WriteLine("Anular
                                            Incorrecto");
484                                         MSGerror();
485                                     }
486                                 }
487                                 else
488                                 {
489                                     //Console.WriteLine("Medio Incorrecto");
490                                     MSGerror();
491                                 }
492                             }
493                             else
494                             {
495                                 //Console.WriteLine("Indice Incorrecto");
496                                 MSGerror();
497                             }
498                         }
499                         else
500                         {
501                             //Console.WriteLine("Pulgar Incorrecto");
502                             MSGerror();
503                         }
504                     }
```

```
                            else
                            {
                                //Console.WriteLine("COD_Inst Incorrecto");
                                MSGerror();
                            }
                        }
                        else
                        {
                            //Console.WriteLine("Tamaño Incorrecto");
                            MSGerror();
                        }
                    }
                    else
                    {
                        //Console.WriteLine("Fin Erroneo");
                        MSGerror();
                    }
                }
                    else
                    {
                    //Console.WriteLine("Cabecera Erronea");
                    MSGerror();
                    }

                }
            else if (Nbytes == 6)
                {
                if (bRecib[0] == Cabecera)
                {
                    //Console.WriteLine("Cabecera OK");

                    if (bRecib[Nbytes - 1] == Fin)
                    {
                        //Console.WriteLine("Fin OK");

                        if (bRecib[1] == tamE)
                        {
                        //Console.WriteLine("Tamaño error recibido");

                            if (bRecib[2] == Error)
                            {
                                //Console.WriteLine("COD_Inst recibido");

                                if (bRecib[3] == errInval)
                                {
                                    //Console.WriteLine("COD_Error Correcto");

                                    if (bRecib[4] == CalculaChecksum(Nbytes, bRecib))
                                    {
                                        //Console.WriteLine("ChekSum OK");
                                        MessageBox.Show("MENSAJE DE ERROR RECIBIDO",
                                        "Estado",MessageBoxButtons.OK,MessageBoxIcon.
                                        Error);
                                        contError++;
                                        if(contError==3)
                                        {
                                            MessageBox.Show("FIN COMUNICACION",
                                            "Estado", MessageBoxButtons.OK,
                                            MessageBoxIcon.Exclamation);

                                                Desconexion();
                                        }

                                    }
                                    else
                                    {
                                        //Console.WriteLine("ChekSum Incorrecto");
                                        MSGerror();
                                    }
                                }
                                else
                                {
                                    //Console.WriteLine("COD_Error Incorrecto");
```

```
                                    MSGerror();
                                }
                            }
                            else
                            {
                                //Console.WriteLine("COD_Inst Incorrecto");
                                MSGerror();
                            }
                        }
                        else
                        {
                            //Console.WriteLine("Tamaño Incorrecto");
                            MSGerror();
                        }
                    }
                    else
                    {
                        //Console.WriteLine("Fin Erroneo");
                        MSGerror();
                    }
                }
                else
                {
                    //Console.WriteLine("Cabecera Erronea");
                    MSGerror();
                }

                ReenvioMSG();

            }
            else
                {
                    //Console.WriteLine("Tamaño Incorrecto");
                    MSGerror();
                }
        }

        byte Cabecera = 0x1B;
        byte Fin = 0x1D;
        byte tam1 =0x0B;
        byte tam2 =0x1F;
        byte tamE = 0x06;
        byte petLec = 0x01;
        byte resLec = 0x11;
        byte petEsc = 0x02;
        byte resEsc = 0x22;
        byte Error = 0xFF;
        byte errInval = 0xF0;
        byte errNoerror = 0x00;
        byte varAct1 = 0x00;
        byte varAct2 = 0x01;
        byte varAct3 = 0x02;
        byte varAct4 = 0x03;
        byte varAct5 = 0x04;

        float fAct1;
        float fAct2;
        float fAct3;
        float fAct4;
        float fAct5;
        byte[] aAct1;
        byte[] aAct2;
        byte[] aAct3;
        byte[] aAct4;
        byte[] aAct5;
        bool ok;

        private void PeticionEscritura()
        {

            int tPETESC = 31;
            byte[] mPetEsc = new byte[tPETESC];

```

```
647             mPetEsc[0] = Cabecera;
648             mPetEsc[1] = tam2;
649             mPetEsc[2] = petEsc;
650             mPetEsc[3] = varAct1;
651             mPetEsc[4] = aAct1[3];
652             mPetEsc[5] = aAct1[2];
653             mPetEsc[6] = aAct1[1];
654             mPetEsc[7] = aAct1[0];
655             mPetEsc[8] = varAct2;
656             mPetEsc[9] = aAct2[3];
657             mPetEsc[10] = aAct2[2];
658             mPetEsc[11] = aAct2[1];
659             mPetEsc[12] = aAct2[0];
660             mPetEsc[13] = varAct3;
661             mPetEsc[14] = aAct3[3];
662             mPetEsc[15] = aAct3[2];
663             mPetEsc[16] = aAct3[1];
664             mPetEsc[17] = aAct3[0];
665             mPetEsc[18] = varAct4;
666             mPetEsc[19] = aAct4[3];
667             mPetEsc[20] = aAct4[2];
668             mPetEsc[21] = aAct4[1];
669             mPetEsc[22] = aAct4[0];
670             mPetEsc[23] = varAct5;
671             mPetEsc[24] = aAct5[3];
672             mPetEsc[25] = aAct5[2];
673             mPetEsc[26] = aAct5[1];
674             mPetEsc[27] = aAct5[0];
675             mPetEsc[28] = errNoerror;
676             mPetEsc[29] = CalculaChecksum(tPETESC, mPetEsc);
677             mPetEsc[30] = Fin;
678 
679             serialBT.Write(mPetEsc, 0, mPetEsc.Length);
680             MessageBox.Show("PETICION DE ESCRITURA ENVIADA", "Estado",
                MessageBoxButtons.OK,MessageBoxIcon.Information);
681 
682             msgUltimo = 1;
683 
684         }
685         private void PeticionLectura()
686         {
687             int tPETLEC = 11;
688             byte[] mPetLec = new byte[tPETLEC];
689 
690             mPetLec[0] = Cabecera;
691             mPetLec[1] = tam1;
692             mPetLec[2] = petLec;
693             mPetLec[3] = varAct1;
694             mPetLec[4] = varAct2;
695             mPetLec[5] = varAct3;
696             mPetLec[6] = varAct4;
697             mPetLec[7] = varAct5;
698             mPetLec[8] = errNoerror;
699             mPetLec[9] = CalculaChecksum(tPETLEC, mPetLec);
700             mPetLec[10] = Fin;
701 
702             serialBT.Write(mPetLec, 0, mPetLec.Length);
703             MessageBox.Show("PETICION DE LECTURA ENVIADA", "Estado",MessageBoxButtons
                .OK,MessageBoxIcon.Information);
704 
705             msgUltimo = 2;
706 
707 
708         }
709         private void MSGerror()
710         {
711             int tERROR = 6;
712             byte[] mError = new byte[tERROR];
713 
714             mError[0] = Cabecera;
715             mError[1] = tamE;
716             mError[2] = Error;
717             mError[3] = errInval;
```

```
718             mError[4] = CalculaChecksum(tERROR, mError);
719             mError[5] = Fin;
720 
721             MessageBox.Show("MENSAJE DE ERROR ENVIADO", "Estado",MessageBoxButtons.OK
                ,MessageBoxIcon.Error);
722             serialBT.Write(mError, 0, mError.Length);
723 
724             envioError++;
725 
726             if (envioError == 3)
727             {
728                 MessageBox.Show("NUMERO MAXIMO DE REINTENTOS ALCANZADO", "Estado",
                    MessageBoxButtons.OK,MessageBoxIcon.Warning);
729                 MessageBox.Show("FIN COMUNICACION", "Estado",MessageBoxButtons.OK,
                    MessageBoxIcon.Exclamation);
730                 Desconexion();
731             }
732         }
733         private void ReenvioMSG()
734         {
735             if (msgUltimo == 1)
736             {
737                 PeticionEscritura();
738             }
739             else if (msgUltimo == 2)
740             {
741                 PeticionLectura();
742             }
743         }
744         private byte CalculaChecksum(int tam, byte[] msg)
745         {
746             byte ck = 0;
747             byte s = 0;
748             if(tam==31)
749             {
750                 s = (byte)(msg[1] + msg[2] + msg[3] + msg[8] + msg[13] + msg[18] +
                    msg[23]+msg[28]);
751                 ck = (byte)(-s);
752 
753             }
754             else
755             {
756                 for (int i = 1; i < tam - 2; i++)
757                 {
758                     s += msg[i];
759                 }
760                 ck = (byte)(-s);
761             }
762 
763             return ck;
764         }
765         public static byte[] Combine(byte[] first, byte[] second)
766         {
767             byte[] ret = new byte[first.Length + second.Length];
768             Buffer.BlockCopy(first, 0, ret, 0, first.Length);
769             Buffer.BlockCopy(second, 0, ret, first.Length, second.Length);
770             return ret;
771         }
772         private void CompruebaDatos()
773         {
774             fAct1 = float.Parse(txtAct1Env.Text.Trim(), CultureInfo.InvariantCulture.
                NumberFormat);
775             fAct2 = float.Parse(txtAct2Env.Text.Trim(), CultureInfo.InvariantCulture.
                NumberFormat);
776             fAct3 = float.Parse(txtAct3Env.Text.Trim(), CultureInfo.InvariantCulture.
                NumberFormat);
777             fAct4 = float.Parse(txtAct4Env.Text.Trim(), CultureInfo.InvariantCulture.
                NumberFormat);
778             fAct5 = float.Parse(txtAct5Env.Text.Trim(), CultureInfo.InvariantCulture.
                NumberFormat);
779 
780             ok = true;
781 
```

```
782                 if (fAct1 > 30)
783                 {
784                     MessageBox.Show("Actuador 1, supera el rango máximo del vastago
                        permitido", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
785                     MessageBox.Show("Introduce de nueva valor permitido RANGO
                        [0-30]mm", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
786                     txtAct1Env.Clear();
787                     ok = false;
788 
789                 }
790                 if (fAct2 > 30)
791                 {
792                     MessageBox.Show("Actuador 2, supera el rango máximo del vastago
                        permitido", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
793                     MessageBox.Show("Introduce de nueva valor permitido RANGO
                        [0-30]mm", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
794                     txtAct2Env.Clear();
795                     ok = false;
796                 }
797                 if (fAct3 > 30)
798                 {
799                     MessageBox.Show("Actuador 3, supera el rango máximo del vastago
                        permitido", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
800                     MessageBox.Show("Introduce de nueva valor permitido RANGO
                        [0-30]mm", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
801                     txtAct3Env.Clear();
802                     ok = false;
803                 }
804                 if (fAct4 > 30)
805                 {
806                     MessageBox.Show("Actuador 4, supera el rango máximo del vastago
                        permitido", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
807                     MessageBox.Show("Introduce de nueva valor permitido RANGO
                        [0-30]mm", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
808                     txtAct4Env.Clear();
809                     ok = false;
810                 }
811                 if (fAct5 > 30)
812                 {
813                     MessageBox.Show("Actuador 5, supera el rango máximo del vastago
                        permitido", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
814                     MessageBox.Show("Introduce de nueva valor permitido RANGO
                        [0-30]mm", "ATENCION", MessageBoxButtons.OK, MessageBoxIcon.
                        Warning);
815                     txtAct5Env.Clear();
816                     ok = false;
817                 }
818 
819             aAct1 = BitConverter.GetBytes(fAct1);
820             aAct2 = BitConverter.GetBytes(fAct2);
821             aAct3 = BitConverter.GetBytes(fAct3);
822             aAct4 = BitConverter.GetBytes(fAct4);
823             aAct5 = BitConverter.GetBytes(fAct5);
824 
825         }
826 
827         private void MostrarDatos(object sender, EventArgs e)
828         {
829             txtAct1Rec.AppendText(sAct1);
830         }
831         private void MostrarDatos1(object sender, EventArgs e)
832         {
833             txtAct2Rec.AppendText(sAct2);
834         }
```

```
835         private void MostrarDatos2(object sender, EventArgs e)
836         {
837             txtAct3Rec.AppendText(sAct3);
838         }
839         private void MostrarDatos3(object sender, EventArgs e)
840         {
841             txtAct4Rec.AppendText(sAct4);
842         }
843         private void MostrarDatos4(object sender, EventArgs e)
844         {
845             txtAct5Rec.AppendText(sAct5);
846         }
847 
848         private void txtPulgar_KeyPress(object sender, KeyPressEventArgs e)
849         {
850             if((e.KeyChar >= 32 && e.KeyChar<=45) || e.KeyChar == 47 || (e.KeyChar>=
                58 && e.KeyChar<=255))
851             {
852 
853                 MessageBox.Show("Solo se permiten Numeros y Positivos Rango [0-30]mm"
                    , "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
854                 e.Handled = true;
855                 return;
856             }
857 
858         }
859 
860         private void txtIndice_KeyPress(object sender, KeyPressEventArgs e)
861         {
862             if ((e.KeyChar >= 32 && e.KeyChar <= 45) || e.KeyChar == 47 || (e.KeyChar
                 >= 58 && e.KeyChar <= 255))
863             {
864                 MessageBox.Show("Solo se permiten Numeros y Positivos Rango [0-30]mm"
                    , "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
865                 e.Handled = true;
866                 return;
867             }
868 
869         }
870 
871         private void txtMedio_KeyPress(object sender, KeyPressEventArgs e)
872         {
873             if ((e.KeyChar >= 32 && e.KeyChar <= 45) || e.KeyChar == 47 || (e.KeyChar
                 >= 58 && e.KeyChar <= 255))
874             {
875                 MessageBox.Show("Solo se permiten Numeros y Positivos Rango [0-30]mm"
                    , "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
876                 e.Handled = true;
877                 return;
878             }
879         }
880 
881         private void txtAnular_KeyPress(object sender, KeyPressEventArgs e)
882         {
883             if ((e.KeyChar >= 32 && e.KeyChar <= 45) || e.KeyChar == 47 || (e.KeyChar
                 >= 58 && e.KeyChar <= 255))
884             {
885                 MessageBox.Show("Solo se permiten Numeros y Positivos Rango [0-30]mm"
                    , "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
886                 e.Handled = true;
887                 return;
888             }
889         }
890 
891         private void txtMenique_KeyPress(object sender, KeyPressEventArgs e)
892         {
893             if ((e.KeyChar >= 32 && e.KeyChar <= 45) || e.KeyChar == 47 || (e.KeyChar
                 >= 58 && e.KeyChar <= 255))
894             {
895                 MessageBox.Show("Solo se permiten Numeros y Positivos Rango [0-30]mm"
                    , "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
896                 e.Handled = true;
897                 return;
```

```
898                 }
899
900             }
901
902         }
903     }
```

ANEXO II: CODIGO ARDUINO

```
///-------------------------------------------------///
/// TRABAJO FIN DE GRADO                            ///
/// INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA  ///
/// UNIVERSIDAD DE VALLADOLID                       ///
///                                                 ///
///  Implementación de un entorno de comunicación   ///
///  Bluetooth basado en el módulo HC-06            ///
///                                                 ///
/// CODIGO 1 - ARDUINO                              ///
///                                                 ///
/// AUTOR - DAVID MUCIENTES SAN JOSÉ                ///
///-------------------------------------------------///

#include <SoftwareSerial.h>
SoftwareSerial ble(10, 11);

#define TAM_MAX 31

byte bCabecera = 0x1B;
byte bFin = 0x1D;
byte bTam1 = 0x1F;
byte bTam2 = 0x0B;
byte bTamError = 0x06;
byte bPetLec = 0x01;
byte bResLec = 0x11;
byte bPetEsc = 0x02;
byte bResEsc = 0x22;
byte bError = 0xFF;
byte bTramaInval = 0xF0;
byte bNoerror = 0x00;
byte bAct1 = 0x00;
byte bAct2 = 0x01;
byte bAct3 = 0x02;
byte bAct4 = 0x03;
byte bAct5 = 0x04;

union datos {
  float fval;
  byte  b[4];
} act1, act2, act3, act4, act5;

float fAct1;
float fAct2;
float fAct3;
float fAct4;
float fAct5;

byte aAct1[4];
byte aAct2[4];
byte aAct3[4];
byte aAct4[4];
byte aAct5[4];

byte aAct1R[5];
byte aAct2R[5];
byte aAct3R[5];
byte aAct4R[5];
byte aAct5R[5];

float fAct1R;
float fAct2R;
float fAct3R;
float fAct4R;
float fAct5R;

int tam;
byte bLeido[TAM_MAX];
int msgUltimo;
int contError = 0;
int contMSG = 0;
String inString = "";

void setup() {
```

```
  Serial.begin(9600);
  ble.begin(9600);
}

void loop() {
  if (ble.available()) {
    ble.listen();
    ble.flush();
    delay(1000);
    tam = ble.available();
    bLeido[tam];
    ProcesaDatos();
  }
}

void ProcesaDatos() {
  if (tam == 31) {
    for (int i = 0; i < 31; i++) {
      bLeido[i] = ble.read();
    }
    if (bLeido[0] == bCabecera) {
      //Serial.println("Cabecera OK");
      if (bLeido[30] == bFin) {
        //Serial.println("FIN OK");
        if (bLeido[1] == bTam1) {

          //Serial.println("Tamaño Escritura Correcto");
          if (bLeido[2] == bPetEsc) {
            //Serial.println("Peticion Escritura OK");
            if (bLeido[3] == bAct1) {

              aAct1[0] = bLeido[7];
              aAct1[1] = bLeido[6];
              aAct1[2] = bLeido[5];
              aAct1[3] = bLeido[4];
              float* fp = (float*)aAct1;
              fAct1 = *fp;

              if (bLeido[8] == bAct2) {

                aAct2[0] = bLeido[12];
                aAct2[1] = bLeido[11];
                aAct2[2] = bLeido[10];
                aAct2[3] = bLeido[9];
                float* fp = (float*)aAct2;
                fAct2 = *fp;

                if (bLeido[13] == bAct3) {

                  aAct3[0] = bLeido[17];
                  aAct3[1] = bLeido[16];
                  aAct3[2] = bLeido[15];
                  aAct3[3] = bLeido[14];
                  float* fp = (float*)aAct3;
                  fAct3 = *fp;

                  if (bLeido[18] == bAct4) {

                    aAct4[0] = bLeido[22];
                    aAct4[1] = bLeido[21];
                    aAct4[2] = bLeido[20];
                    aAct4[3] = bLeido[19];
                    float* fp = (float*)aAct4;
                    fAct4 = *fp;

                    if (bLeido[23] == bAct5) {

                      aAct5[0] = bLeido[27];
                      aAct5[1] = bLeido[26];
                      aAct5[2] = bLeido[25];
                      aAct5[3] = bLeido[24];
                      float* fp = (float*)aAct5;
                      fAct5 = *fp;
```

```
146
147                         if (bLeido[28] == bNoerror) {
148                           //Serial.println("COD_ERROR Correcto");
149                           if (bLeido[29] == calculaChecksum(tam, bLeido)) {
150                             //Serial.println("CheckSum Correcto");
151
152                             Serial.println("----RECIBIDA PETICION DE ESCRITURA----");
153                             delay(500);
154                             Serial.print("Posicion Actuador 1: ");
155                             delay(100);
156                             Serial.println(fAct1, 2);
157                             Serial.print("Posicion Actuador 2: ");
158                             delay(100);
159                             Serial.println(fAct2, 2);
160                             Serial.print("Posicion Actuador 3: ");
161                             delay(100);
162                             Serial.println(fAct3, 2);
163                             Serial.print("Posicion Actuador 4: ");
164                             delay(100);
165                             Serial.println(fAct4, 2);
166                             Serial.print("Posicion Actuador 5: ");
167                             delay(100);
168                             Serial.println(fAct5, 2);
169
170                             RespuestaEscritura();
171
172                           } else {
173                             //Serial.println("CheckSum Incorrecto");
174                             MSGerror ();
175                           }
176                         } else {
177                           //Serial.println("COD_ERROR Incorrecto");
178                           MSGerror ();
179                         }
180                       } else {
181                         //Serial.println("VAR_ID Menique Incorrecta");
182                         MSGerror ();
183                       }
184                     } else {
185                       //Serial.println("VAR_ID Anular Incorrecta");
186                       MSGerror ();
187                     }
188                   } else {
189                     //Serial.println("VAR_ID Medio Incorrecta");
190                     MSGerror ();
191                   }
192                 } else {
193                   //Serial.println("VAR_ID Indice Incorrecta");
194                   MSGerror ();
195                 }
196
197               } else {
198                 //Serial.println("VAR_ID Pulgar Incorrecta");
199                 MSGerror ();
200               }
201             } else {
202               //Serial.println("Tamaño Peticion escritura Incorrecto");
203             }
204
205           } else {
206             //Serial.println("Tamaño Escritura Incorrecto");
207             MSGerror ();
208           }
209         } else {
210           //Serial.println("Fin Incorrecto");
211           MSGerror ();
212         }
213       } else {
214         //Serial.println("Cabecera Incorrecta");
215         MSGerror ();
216       }
217
218     }
```

```
219     else if (tam == 11) {
220       for (int i = 0; i < 11; i++) {
221         bLeido[i] = ble.read();
222         //Serial.println(bLeido[i], HEX);
223       }
224       if (bLeido[0] == bCabecera) {
225         //Serial.println("Cabecera Correcta");
226         if (bLeido[10] == bFin) {
227           //Serial.println("Fin Correcto");
228 
229           if (bLeido[1] == bTam2) {
230             //Serial.println("Tamaño byteTam2 Correcto");
231 
232             if (bLeido[2] == bPetLec) {
233               //Serial.println("PetLec Correcto");
234 
235               if (bLeido[3] == bAct1) {
236                 //Serial.println("VAR_ID Pulgar Correcto");
237 
238                 if (bLeido[4] == bAct2) {
239                   //Serial.println("VAR_ID Indice Correcto");
240 
241                   if (bLeido[5] == bAct3) {
242                     //Serial.println("VAR_ID Medio Correcto");
243 
244                     if (bLeido[6] == bAct4) {
245                       //Serial.println("VAR_ID Anular Correcto");
246 
247                       if (bLeido[7] == bAct5) {
248                         //Serial.println("VAR_ID Menique Correcto");
249 
250                         if (bLeido[8] == bNoerror) {
251                           //Serial.println("COD_ERROR Correcto");
252 
253                           if (bLeido[9] == calculaChecksum(tam, bLeido)) {
254                             //Serial.println("CheckSum Correcto");
255                             delay(500);
256                             Serial.println("----RECIBIDA PETICION DE LECTURA----");
257                             RespuestaLectura();
258                           } else {
259                             //Serial.println("CheckSum Incorrecto");
260                             MSGerror ();
261                           }
262                         } else {
263                           //Serial.println("COD_ERROR Incorrecto");
264                           MSGerror ();
265                         }
266                       } else {
267                         //Serial.println("VAR_ID Menique Incorrecto");
268                         MSGerror ();
269                       }
270                     } else {
271                       //Serial.println("VAR_ID Anular Incorrecto");
272                       MSGerror ();
273                     }
274                   } else {
275                     //Serial.println("VAR_ID Medio Incorrecto");
276                     MSGerror ();
277                   }
278                 } else {
279                   //Serial.println("VAR_ID Indice Incorrecto");
280                   MSGerror ();
281                 }
282               } else {
283                 //Serial.println("VAR_ID Pulgar Incorrecto");
284                 MSGerror ();
285               }
286             } else {
287               //Serial.println("COD_INST Incorrecto");
288               MSGerror ();
289             }
290           } else {
291             //Serial.println("Tamaño byteTam2 Incorrecto");
```

```
292             MSGerror ();
293           }
294         } else {
295           //Serial.println("Fin Incorrecto");
296           MSGerror ();
297         }
298       } else {
299         //Serial.println("Cabecera Incorrecta");
300         MSGerror ();
301       }
302
303     }
304     else if (tam == 6) {
305       for (int i = 0; i < 6; i++) {
306         bLeido[i] = ble.read();
307         //Serial.println(bLeido[i], HEX);
308       }
309       if (bLeido[0] == bCabecera) {
310         //Serial.println("Cabecera OK");
311         if (bLeido[5] == bFin) {
312           //Serial.println("Fin OK");
313           if (bLeido[1] == bTamError) {
314             //Serial.println("TamError OK");
315             if (bLeido[2] == bError) {
316               //Serial.println("COD_INST Error OK");
317               if (bLeido[3] == bTramaInval) {
318                 //Serial.println("COD_ERR OK");
319                 if (bLeido[4] == calculaChecksum(tam, bLeido)) {
320                   //Serial.println("CheckSum OK");
321                   Serial.println("----RECIBIDO MENSAJE DE ERROR----");
322
323                   if (contError < 2)
324                   {
325                     ReenvioMSG();
326                     contError++;
327                   }
328                   if (contError == 2)
329                   {
330                     delay(500);
331                     Serial.println("----RECIBIDO MENSAJE DE ERROR----");
332                     Serial.println("----FIN DE LA COMUNICACION----");
333                     Serial.end();
334                     ble.end();
335                   }
336                 } else {
337                   //Serial.println("CheckSum Incorrecto");
338                   MSGerror ();
339                 }
340               } else {
341                 //Serial.println("COD_ERR Incorrecto");
342                 MSGerror ();
343               }
344             } else {
345               //Serial.println("COD_INST Error Incorrecto");
346               MSGerror ();
347             }
348           } else {
349             //Serial.println("Tamaño de mensaje no valido");
350             MSGerror ();
351           }
352
353         } else {
354           //Serial.println("Fin Incorrecto");
355           MSGerror ();
356         }
357
358       } else {
359         //Serial.println("Cabecera Incorrecta");
360         MSGerror ();
361       }
362     }
363   }
364
```

```
365  void RespuestaEscritura() {
366    int tESC = 11;
367    byte msgRespEscr[tESC];
368    msgRespEscr[0] = bCabecera;
369    msgRespEscr[1] = bTam2;
370    msgRespEscr[2] = bResEsc;
371    msgRespEscr[3] = bAct1;
372    msgRespEscr[4] = bAct2;
373    msgRespEscr[5] = bAct3;
374    msgRespEscr[6] = bAct4;
375    msgRespEscr[7] = bAct5;
376    msgRespEscr[8] = bNoerror;
377    msgRespEscr[9] = calculaChecksum(tESC, msgRespEscr);
378    msgRespEscr[10] = bFin;
379
380    for (int i = 0; i < tESC; i++) {
381      ble.write(msgRespEscr[i]);
382    }
383    Serial.println("----RESPUESTA DE ESCRITURA ENVIADA----");
384    msgUltimo = 1;
385  }
386
387  void RespuestaLectura() {
388    int tRESP = 31;
389    byte msgRespLec[tRESP];
390
391    if(msgUltimo!=2)
392    {
393      PideDatos();
394    }
395
396    act1.fval = fAct1R;
397    act2.fval = fAct2R;
398    act3.fval = fAct3R;
399    act4.fval = fAct4R;
400    act5.fval = fAct5R;
401
402    msgRespLec[0] = bCabecera;
403    msgRespLec[1] = bTam1;
404    msgRespLec[2] = bResLec;
405    msgRespLec[3] = bAct1;
406    msgRespLec[4] = act1.b[3];
407    msgRespLec[5] = act1.b[2];
408    msgRespLec[6] = act1.b[1];
409    msgRespLec[7] = act1.b[0];
410    msgRespLec[8] = bAct2;
411    msgRespLec[9] = act2.b[3];
412    msgRespLec[10] = act2.b[2];
413    msgRespLec[11] = act2.b[1];
414    msgRespLec[12] = act2.b[0];
415    msgRespLec[13] = bAct3;
416    msgRespLec[14] = act3.b[3];
417    msgRespLec[15] = act3.b[2];
418    msgRespLec[16] = act3.b[1];
419    msgRespLec[17] = act3.b[0];
420    msgRespLec[18] = bAct4;
421    msgRespLec[19] = act4.b[3];
422    msgRespLec[20] = act4.b[2];
423    msgRespLec[21] = act4.b[1];
424    msgRespLec[22] = act4.b[0];
425    msgRespLec[23] = bAct5;
426    msgRespLec[24] = act5.b[3];
427    msgRespLec[25] = act5.b[2];
428    msgRespLec[26] = act5.b[1];
429    msgRespLec[27] = act5.b[0];
430    msgRespLec[28] = bNoerror;
431    msgRespLec[29] = calculaChecksum(tRESP, msgRespLec);
432    msgRespLec[30] = bFin;
433
434    for (int i = 0; i < tRESP; i++) {
435      ble.write(msgRespLec[i]);
436    }
437    Serial.println("----RESPUESTA DE LECTURA ENVIADA----");
```

```
438     msgUltimo = 2;
439   }
440
441   void MSGerror () {
442     int tER = 6;
443     byte msgERROR[tER];
444     msgERROR[0] = bCabecera;
445     msgERROR[1] = bTamError;
446     msgERROR[2] = bError;
447     msgERROR[3] = bTramaInval;
448     msgERROR[4] = calculaChecksum(tER, msgERROR);
449     msgERROR[5] = bFin;
450
451     for (int i = 0; i < tER; i++) {
452       ble.write(msgERROR[i]);
453     }
454     Serial.println("----MENSAJE DE ERROR ENVIADO----");
455
456     contMSG++;
457
458     if (contMSG == 3)
459     {
460       delay(500);
461       Serial.println("----FIN DE LA COMUNICACION----");
462       Serial.end();
463       ble.end();
464     }
465     ok = 0;
466   }
467
468   void ReenvioMSG() {
469     if (msgUltimo == 1)
470     {
471       RespuestaEscritura();
472     }
473     else if (msgUltimo == 2)
474     {
475       RespuestaLectura();
476     }
477   }
478
479   void PideDatos() {
480     String str;
481
482     Serial.println("INTRODUCE POSICIONES DE LOS ACTUADORES (formato XX.XX) ");
483     delay(200);
484     Serial.print("Posicion Actuador 1: ");
485     while (Serial.available() == 0)
486     {
487
488     }
489     str = Serial.readStringUntil('\n');
490     fAct1R = str.toFloat();
491     Serial.println(fAct1R);
492     str="";
493
494     Serial.print("Posicion Actuador 2: ");
495     while (Serial.available() == 0)
496     {
497
498     }
499     str = Serial.readStringUntil('\n');
500     fAct2R = str.toFloat();
501     Serial.println(fAct2R);
502     str="";
503
504     Serial.print("Posicion Actuador 3: ");
505     while (Serial.available() == 0)
506     {
507
508     }
509     str = Serial.readStringUntil('\n');
510     fAct3R = str.toFloat();
```

```
    Serial.println(fAct3R);
    str="";

    Serial.print("Posicion Actuador 4: ");
    while (Serial.available() == 0)
    {

    }
    str = Serial.readStringUntil('\n');
    fAct4R = str.toFloat();
    Serial.println(fAct4R);
    str="";

    Serial.print("Posicion Actuador 5: ");
    while (Serial.available() == 0)
    {

    }
    str = Serial.readStringUntil('\n');
    fAct5R = str.toFloat();
    Serial.println(fAct5R);
    str="";
  }

  byte calculaChecksum(int tam, byte msg[]) {

    byte sum = 0;
    byte calculated_cksum;
    if (tam == 31) {
      sum = msg[1] + msg[2] + msg[3] + msg[8] + msg[13] + msg[18] + msg[23] + msg[28];
      calculated_cksum = -sum;
    }
    else {
      for (int i = 1; i < tam - 2; i++)
      {
        sum += msg[i];
      }

      calculated_cksum = -sum;
    }


    return calculated_cksum;
  }
```

ANEXO III: CODIGO LAUNCHXL-F28069M

```
///-------------------------------------------------///
/// TRABAJO FIN DE GRADO                            ///
/// INGENIERIA ELECTRONICA INDUSTRIAL Y AUTOMATICA  ///
/// UNIVERSIDAD DE VALLADOLID                       ///
///                                                 ///
///  Implementación de un entorno de comunicación   ///
///  Bluetooth basado en el módulo HC-06            ///
///                                                 ///
/// CODIGO 2 - F28069M                              ///
///                                                 ///
/// AUTOR - DAVID MUCIENTES SAN JOSÉ                ///
///-------------------------------------------------///

#include "DSP28x_Project.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

void iniciaSCIa(void);
void enviaSCIa(int a);
void msgSCIa(char *msg);

void iniciaSCIb(void);
void enviaSCIb(int a);
void msgSCIb(char *msg);

void RespuestaLectura();
void RespuestaEscritura();
void Error();
Uint8 calculaChecksum(int tam, Uint8 msg[]);
void ReenvioMSG();

void MuestraDatosRecibidos();
void ProcesaDatos();
void PideDatos();

Uint8 bCabecera = 0x1B;
Uint8 bFin = 0x1D;
Uint8 bTam1 = 0x1F;
Uint8 bTam2 = 0x0B;
Uint8 bTamError = 0x06;
Uint8 bPetLec = 0x01;
Uint8 bResLec = 0x11;
Uint8 bPetEsc = 0x02;
Uint8 bResEsc = 0x22;
Uint8 bError = 0xFF;
Uint8 bCheckSum;
Uint8 bTramaInval = 0xF0;
Uint8 bNoerror = 0x00;
Uint8 bAct1 = 0x00;
Uint8 bAct2 = 0x01;
Uint8 bAct3 = 0x02;
Uint8 bAct4 = 0x03;
Uint8 bAct5 = 0x04;

char aAct1[4];
char aAct2[4];
char aAct3[4];
char aAct4[4];
char aAct5[4];

char aAct1R[5];
char aAct2R[5];
char aAct3R[5];
char aAct4R[5];
char aAct5R[5];
char aAct6R[5];

float32 fAct1;
float32 fAct2;
float32 fAct3;
float32 fAct4;
```

```
74  float32 fAct5;
75  float32 fAct6;
76
77  float32 Act1R;
78  float32 Act2R;
79  float32 Act3R;
80  float32 Act4R;
81  float32 Act5R;
82  float32 Act6R;
83
84  int LoopCount;
85  Uint16 ErrorCount;
86
87  struct BYTES
88  {
89      Uint8 byte1 :8;
90      Uint8 byte2 :8;
91      Uint8 byte3 :8;
92      Uint8 byte4 :8;
93  };
94
95  union
96  {
97      float32 fval;
98      struct BYTES bytes;
99  } act1, act2, act3, act4, act5,act6;
100
101 Uint8 MSG[31];
102 char *msg;
103 Uint8 DatoRecibido;
104 int i;
105 int msgUltimo;
106 int contMSG = 0;
107 int contError = 0;
108
109 void main(void)
110 {
111
112     InitSysCtrl();
113
114     InitGpio();
115     EALLOW;
116
117     InitSciaGpio();
118     InitScibGpio();
119
120     DINT;
121
122     InitPieCtrl();
123
124     IER = 0x0000;
125     IFR = 0x0000;
126
127     InitPieVectTable();
128
129     LoopCount = 0;
130     ErrorCount = 0;
131
132     iniciaSCIa();
133     iniciaSCIb();
134
135     i = 0;
136     DatoRecibido = 0;
137     msgUltimo = 0;
138     contMSG = 0;
139     contError = 0;
140
141     for (;;)
142     {
143         DatoRecibido = 0;
144         i = 0;
145         LoopCount = 0;
146         while (DatoRecibido != bFin)
```

```
147          {
148              while (ScibRegs.SCIFFRX.bit.RXFFST == 0)
149              {
150
151              }
152
153              DatoRecibido = ScibRegs.SCIRXBUF.all;
154
155              MSG[i] = DatoRecibido;
156              enviaSCIa(MSG[i]);
157              i++;
158
159              LoopCount++;
160          }
161          ProcesaDatos();
162
163          ScibRegs.SCICTL1.bit.SWRESET = 1;
164      }
165
166  }
167
168  void ProcesaDatos()
169  {
170
171      if (LoopCount == 31)
172      {
173          if (MSG[0] == bCabecera)
174          {
175  //            msg = "\r\nCABECERA OK";
176  //            msgSCIa(msg);
177              if (MSG[1] == bTam1)
178              {
179  //                msg = "\r\nTAMAÑO PETICION ESCRITURA";
180  //                msgSCIa(msg);
181
182                  if (MSG[2] == bPetEsc)
183                  {
184  //                    msg = "\r\nPET ESC";
185  //                    msgSCIa(msg);
186                      if (MSG[3] == bAct1)
187                      {
188  //                        msg = "\r\nPULGAR OK";
189  //                        msgSCIa(msg);
190
191                          act1.bytes.byte1 = MSG[7];
192                          act1.bytes.byte2 = MSG[6];
193                          act1.bytes.byte3 = MSG[5];
194                          act1.bytes.byte4 = MSG[4];
195                          Act1R = act1.fval;
196
197                          if (MSG[8] == bAct2)
198                          {
199  //                            msg = "\r\nINDICE OK";
200  //                            msgSCIa(msg);
201
202                              act2.bytes.byte1 = MSG[12];
203                              act2.bytes.byte2 = MSG[11];
204                              act2.bytes.byte3 = MSG[10];
205                              act2.bytes.byte4 = MSG[9];
206                              Act2R = act2.fval;
207
208                              if (MSG[13] == bAct3)
209                              {
210  //                                msg = "\r\nMEDIO OK";
211  //                                msgSCIa(msg);
212
213                                  act3.bytes.byte1 = MSG[17];
214                                  act3.bytes.byte2 = MSG[16];
215                                  act3.bytes.byte3 = MSG[15];
216                                  act3.bytes.byte4 = MSG[14];
217                                  Act3R = act3.fval;
218
219                                  if (MSG[18] == bAct4)
```

```
                                {
//                                    msg = "\r\nANULAR OK";
//                                    msgSCIa(msg);

                                    act4.bytes.byte1 = MSG[22];
                                    act4.bytes.byte2 = MSG[21];
                                    act4.bytes.byte3 = MSG[20];
                                    act4.bytes.byte4 = MSG[19];
                                    Act4R = act4.fval;

                                    if (MSG[23] == bAct5)
                                    {
//                                      msg = "\r\nMENIQUE OK";
//                                      msgSCIa(msg);

                                        act5.bytes.byte1 = MSG[27];
                                        act5.bytes.byte2 = MSG[26];
                                        act5.bytes.byte3 = MSG[25];
                                        act5.bytes.byte4 = MSG[24];
                                        Act5R = act5.fval;

                                        if (MSG[28] == bNoerror)
                                        {
//                                          msg = "\r\nNO ERROR OK";
//                                          msgSCIa(msg);

                                            if (MSG[29]
                                                    == calculaChecksum(
                                                            LoopCount,
                                                            MSG)/*bCheckSum*/)
                                            {
//                                              msg = "\r\nCHECKSUM OK";
//                                              msgSCIa(msg);
                                                if (MSG[30] == bFin)
                                                {
//                                                  msg = "\r\nFIN OK";
//                                                  msgSCIa(msg);
                                                    msg =
                                                            "\r\n-----RECIBIDA
                                                            PETICION DE
                                                            ESCRITURA-----";
                                                    msgSCIa(msg);
                                                    RespuestaEscritura();
                                                }
                                                else
                                                {
//                                                  msg = "\r\nFIN incorrecto";
//                                                  msgSCIa(msg);
                                                    Error();
                                                }
                                            }
                                            else
                                            {
//                                              msg = "\r\nCHECKSUM incorrecto";
//                                              msgSCIa(msg);
                                                Error();
                                            }
                                        }
                                        else
                                        {
//                                          msg = "\r\nNO ERROR incorrecto";
//                                          msgSCIa(msg);
                                            Error();
                                        }
                                    }
                                    else
                                    {
//                                      msg = "\r\nMENIQUE incorrecto";
//                                      msgSCIa(msg);
                                        Error();
                                    }
                                }
                                else
```

```
                                    {
//                                        msg = "\r\nANULAR incorrecto";
//                                        msgSCIa(msg);
                                      Error();
                                    }
                                }
                                else
                                {
//                                    msg = "\r\nMEDIO incorrecto";
//                                    msgSCIa(msg);
                                    Error();
                                }
                            }
                            else
                            {
//                                msg = "\r\nINDICE incorrecto";
//                                msgSCIa(msg);
                                Error();
                            }
                        }
                        else
                        {
//                            msg = "\r\nPULGAR incorrecto";
//                            msgSCIa(msg);
                            Error();
                        }

                    }
                    else
                    {
//                        msg = "\r\nPET LEC incorrecta";
//                        msgSCIa(msg);
                        Error();
                    }

                }
                else
                {
//                    msg = "\r\nTAMAÑO incorrecta";
//                    msgSCIa(msg);
                    Error();
                }
            }
            else
            {
//                msg = "\r\nCABECERA incorrecta";
//                msgSCIa(msg);
                Error();
            }
        }
        else if (LoopCount == 11)
        {
            if (MSG[0] == bCabecera)
            {
                //            msg = "\r\nCABECERA OK";
                //            msgSCIa(msg);
                if (MSG[1] == bTam2)
                {
//                    msg = "\r\nTAMAÑO PETICION LECTURA";
//                    msgSCIa(msg);
                    if (MSG[2] == bPetLec)
                    {
//                        msg = "\r\nPET LEC OK";
//                        msgSCIa(msg);
                        if (MSG[3] == bAct1)
                        {
//                            msg = "\r\nPULGAR OK";
//                            msgSCIa(msg);
                            if (MSG[4] == bAct2)
                            {
//                                msg = "\r\nINDICE OK";
//                                msgSCIa(msg);
                                if (MSG[5] == bAct3)
```

```
364                                 {
365   //                                  msg = "\r\nMEDIO OK";
366   //                                  msgSCIa(msg);
367                                     if (MSG[6] == bAct4)
368                                     {
369   //                                      msg = "\r\nANULAR OK";
370   //                                      msgSCIa(msg);
371                                         if (MSG[7] == bAct5)
372                                         {
373   //                                          msg = "\r\nMENIQUE OK";
374   //                                          msgSCIa(msg);
375                                             if (MSG[8] == bNoerror)
376                                             {
377   //                                              msg = "\r\nNO ERROR OK";
378   //                                              msgSCIa(msg);
379
380                                                 if (MSG[9]
381                                                     == calculaChecksum(
382                                                             LoopCount,
383                                                             MSG)/*bCheckSum*/)
384                                                 {
385   //                                                  msg = "\r\nCHECKSUM OK";
386   //                                                  msgSCIa(msg);
387                                                     if (MSG[10] == bFin)
388                                                     {
389   //                                                      msg = "\r\nFIN OK";
390   //                                                      msgSCIa(msg);
391
392                                                         msg =
393                                                                 "\r\n-----RECIBIDA
                                                                    PETICION DE LECTURA-----";
394                                                         msgSCIa(msg);
395
396                                                         RespuestaLectura();
397                                                     }
398                                                     else
399                                                     {
400   //                                                      msg = "\r\nFIN incorrecto";
401   //                                                      msgSCIa(msg);
402                                                         Error();
403                                                     }
404                                                 }
405                                                 else
406                                                 {
407   //                                                  msg = "\r\nCHECKSUM incorrecto";
408   //                                                  msgSCIa(msg);
409                                                     Error();
410                                                 }
411                                             }
412                                             else
413                                             {
414   //                                              msg = "\r\nNO ERROR incorrecto";
415   //                                              msgSCIa(msg);
416                                                 Error();
417                                             }
418                                         }
419                                         else
420                                         {
421   //                                          msg = "\r\nMENIQUE incorrecto";
422   //                                          msgSCIa(msg);
423                                             Error();
424                                         }
425                                     }
426                                     else
427                                     {
428   //                                      msg = "\r\nANULAR incorrecto";
429   //                                      msgSCIa(msg);
430                                         Error();
431                                     }
432                                 }
433                                 else
434                                 {
435   //                                  msg = "\r\nMEDIO incorrecto";
```

```
436   //                                msgSCIa(msg);
437                                   Error();
438                               }
439                           }
440                           else
441                           {
442   //                            msg = "\r\nINDICE incorrecto";
443   //                            msgSCIa(msg);
444                               Error();
445                           }
446                       }
447                       else
448                       {
449   //                        msg = "\r\nPULGAR incorrecto";
450   //                        msgSCIa(msg);
451                           Error();
452                       }
453
454                   }
455                   else
456                   {
457   //                    msg = "\r\nPET LEC incorrecta";
458   //                    msgSCIa(msg);
459                       Error();
460                   }
461               }
462               else
463               {
464   //                msg = "\r\nTAMAÑO incorrecta";
465   //                msgSCIa(msg);
466                   Error();
467               }
468           }
469           else
470           {
471   //            msg = "\r\nCABECERA incorrecta";
472   //            msgSCIa(msg);
473               Error();
474           }
475
476       }
477
478       else if (LoopCount == 6)
479       {
480           if (MSG[0] == bCabecera)
481           {
482               //            msg = "\r\nCABECERA OK";
483               //            msgSCIa(msg);
484               if (MSG[1] == bTamError)
485               {
486   //                msg = "\r\nTAM ERROR OK";
487   //                msgSCIa(msg);
488
489                   if (MSG[2] == bError)
490                   {
491   //                    msg = "\r\nERROR OK";
492   //                    msgSCIa(msg);
493                       if (MSG[3] == bTramaInval)
494                       {
495   //                        msg = "\r\nTRAMA INVALIDA OK";
496   //                        msgSCIa(msg);
497
498                           if (MSG[4] == 0x0B/*calculaChecksum(LoopCount, MSG)*/)
499                           {
500   //                            msg = "\r\nCHECKSUM OK";
501   //                            msgSCIa(msg);
502                               if (MSG[5] == bFin)
503                               {
504   //                                msg = "\r\nFIN OK";
505   //                                msgSCIa(msg);
506                                   msg =
507                                           "\r\n-----MENSAJE DE ERROR RECIBIDO-----\0";
508                                   msgSCIa(msg);
```

```
509
510                                 if (contError < 3)
511                                 {
512                                     ReenvioMSG();
513                                     contError++;
514                                 }
515                                 if (contError == 2)
516                                 {
517                                     msg =
518                                             "\r\n-----MENSAJE DE ERROR
                                            RECIBIDO-----\0";
519                                     msgSCIa(msg);
520                                     msg =
521                                             "\r\n-----FIN COMUNICACION-----\0";
522                                     msgSCIa(msg);
523                                 }
524
525                             }
526                             else
527                             {
528 //                                msg = "\r\nFIN incorrecto";
529 //                                msgSCIa(msg);
530                                 Error();
531
532                             }
533                         }
534                         else
535                         {
536 //                            msg = "\r\nCHECKSUM incorrecto";
537 //                            msgSCIa(msg);
538                             Error();
539                         }
540                     }
541                     else
542                     {
543 //                        msg = "\r\nTRAMA INVALIDA incorrecto";
544 //                        msgSCIa(msg);
545                         Error();
546                     }
547                 }
548                 else
549                 {
550 //                    msg = "\r\nbERROR incorrecto";
551 //                    msgSCIa(msg);
552                     Error();
553                 }
554             }
555             else
556             {
557 //                msg = "\r\nTAM ERROR incorrecto";
558 //                msgSCIa(msg);
559                 Error();
560             }
561
562         }
563         else
564         {
565 //            msg = "\r\nCABECERA incorrecto";
566 //            msgSCIa(msg);
567             Error();
568         }
569     }
570     else
571     {
572         msg = "\r\n-----MENSAJE INCORRECTO-----\0";
573         msgSCIa(msg);
574         Error();
575     }
576 }
577
578 void RespuestaEscritura()
579 {
580     int i;
```

```
581         int tESC = 11;
582         Uint8 msgRespEsc[11] = { bCabecera, bTam2, bResEsc, bAct1, bAct2,
583                                  bAct3, bAct4, bAct5, bNoerror, calculaChecksum(
584                                          tESC, msgRespEsc),bFin };
585 
586         for (i = 0; i < tESC; i++)
587         {
588             enviaSCIb(msgRespEsc[i]);
589         }
590         msgUltimo = 1;
591 
592         msg = "\r\n-----RESPUESTA DE ESCRITURA ENVIADA-----";
593         msgSCIa(msg);
594     }
595 
596     void RespuestaLectura()
597     {
598         int i;
599         int tLEC = 31;
600 
601         if (msgUltimo != 2)
602         {
603             PideDatos();
604         }
605 
606         Uint8 msgRespLec[31] = { bCabecera, bTam1, bResLec, bAct1,
607                                  act1.bytes.byte4, act1.bytes.byte3,
608                                  act1.bytes.byte2, act1.bytes.byte1, bAct2,
609                                  act6.bytes.byte4, act6.bytes.byte3,
610                                  act6.bytes.byte2, act6.bytes.byte1, bAct3,
611                                  act3.bytes.byte4, act3.bytes.byte3,
612                                  act3.bytes.byte2, act3.bytes.byte1, bAct4,
613                                  act4.bytes.byte4, act4.bytes.byte3,
614                                  act4.bytes.byte2, act4.bytes.byte1, bAct5,
615                                  act5.bytes.byte4, act5.bytes.byte3,
616                                  act5.bytes.byte2, act5.bytes.byte1, bNoerror,
617                                  calculaChecksum(tLEC, msgRespLec), bFin };
618 
619         for (i = 0; i < tLEC; i++)
620         {
621             enviaSCIb(msgRespLec[i]);
622         }
623 
624         msgUltimo = 2;
625 
626         msg = "\r\n-----RESPUESTA DE LECTURA ENVIADA-----";
627         msgSCIa(msg);
628     }
629 
630     void Error()
631     {
632         int i;
633         int tER = 6;
634         Uint8 msgERROR[6] = { bCabecera, bTamError, bError, bTramaInval,
635                               calculaChecksum(tER, msgERROR), bFin };
636 
637         for (i = 0; i < tER; i++)
638         {
639             enviaSCIb(msgERROR[i]);
640         }
641 
642         msg = "\r\n-----MENSAJE ERROR ENVIADO-----";
643         msgSCIa(msg);
644 
645         contMSG++;
646 
647         if (contMSG == 3)
648         {
649             msg = "\r\n-----FIN COMUNICACION-----";
650             msgSCIa(msg);
651         }
652 
653     }
```

```
654
655  void ReenvioMSG()
656  {
657      if (msgUltimo == 1)
658      {
659          RespuestaEscritura();
660      }
661      else if (msgUltimo == 2)
662      {
663          RespuestaLectura();
664      }
665  }
666
667  void PideDatos()
668  {
669      int cont;
670
671      msg = "\r\nINTRODUCE POSICIONES DE LOS ACTUADORES (formato XX.XX)\0";
672      msgSCIa(msg);
673
674      msg = "\r\nPosicion Actuador 1: \0";
675      msgSCIa(msg);
676
677      for (cont = 0; cont < 5; cont++)
678      {
679
680          while (SciaRegs.SCIFFRX.bit.RXFFST != 1)
681          {
682
683          }
684
685          aAct1R[cont] = SciaRegs.SCIRXBUF.all;
686
687          enviaSCIa(aAct1R[cont]);
688
689      }
690
691      msg = "\r\nPosicion Actuador 2: \0";
692          msgSCIa(msg);
693
694          for (cont = 0; cont < 5; cont++)
695          {
696
697              while (SciaRegs.SCIFFRX.bit.RXFFST != 1)
698              {
699
700              }
701
702              aAct2R[cont] = SciaRegs.SCIRXBUF.all;
703
704              enviaSCIa(aAct2R[cont]);
705
706          }
707
708      msg = "\r\nPosicion Actuador 3: \0";
709      msgSCIa(msg);
710
711      for (cont = 0; cont < 5; cont++)
712      {
713
714          while (SciaRegs.SCIFFRX.bit.RXFFST != 1)
715          {
716
717          }
718
719          aAct3R[cont] = SciaRegs.SCIRXBUF.all;
720
721          enviaSCIa(aAct3R[cont]);
722
723      }
724
725      msg = "\r\nPosicion Actuador 4: \0";
726      msgSCIa(msg);
```

```
727
728         for (cont = 0; cont < 5; cont++)
729         {
730
731             while (SciaRegs.SCIFFRX.bit.RXFFST != 1)
732             {
733
734             }
735
736             aAct4R[cont] = SciaRegs.SCIRXBUF.all;
737
738             enviaSCIa(aAct4R[cont]);
739
740         }
741
742         msg = "\r\nPosicion Actuador 5: \0";
743         msgSCIa(msg);
744
745         for (cont = 0; cont < 5; cont++)
746         {
747
748             while (SciaRegs.SCIFFRX.bit.RXFFST != 1)
749             {
750
751             }
752
753             aAct5R[cont] = SciaRegs.SCIRXBUF.all;
754
755             enviaSCIa(aAct5R[cont]);
756
757         }
758
759         act1.fval = atof(aAct1R);
760         act6.fval = atof(aAct2R);
761         act3.fval = atof(aAct3R);
762         act4.fval = atof(aAct4R);
763         act5.fval = atof(aAct5R);
764
765     }
766
767     Uint8 calculaChecksum(int tam, Uint8 msg[])
768     {
769         int i;
770         Uint8 sum = 0;
771         Uint8 calculated_cksum;
772         if (tam == 31)
773         {
774             sum = msg[1] + msg[2] + msg[3] + msg[8] + msg[13] + msg[18] + msg[23]
775                     + msg[28];
776             calculated_cksum = 0xFF - sum + 1;
777         }
778         else
779         {
780             for (i = 1; i < tam - 2; i++)
781             {
782                 sum += msg[i];
783             }
784
785             calculated_cksum = 0xFF - sum + 1;
786         }
787
788         return calculated_cksum;
789     }
790
791     void iniciaSCIa()
792     {
793         SciaRegs.SCICCR.all = 0x0007;
794
795         SciaRegs.SCICTL1.all = 0x0003;
796
797         SciaRegs.SCICTL2.bit.TXINTENA = 0;
798         SciaRegs.SCICTL2.bit.RXBKINTENA = 0;
799
```

```
800         SciaRegs.SCIHBAUD = 0x0001;
801         SciaRegs.SCILBAUD = 0x0024;
802
803         SciaRegs.SCICTL1.all = 0x0023;
804
805         SciaRegs.SCIFFTX.all = 0xE040;
806         SciaRegs.SCIFFRX.all = 0x2044;
807         SciaRegs.SCIFFCT.all = 0x0;
808     }
809
810     void enviaSCIa(int a)
811     {
812         while (SciaRegs.SCIFFTX.bit.TXFFST != 0)
813         {
814
815         }
816         SciaRegs.SCITXBUF = a;
817     }
818
819     void msgSCIa(char *msg)
820     {
821         int i;
822         i = 0;
823         while (msg[i] != '\0')
824         {
825             enviaSCIa(msg[i]);
826             i++;
827         }
828     }
829
830     void iniciaSCIb()
831     {
832         ScibRegs.SCICCR.all = 0x0007;
833
834         ScibRegs.SCICTL1.all = 0x0003;
835
836         ScibRegs.SCICTL2.bit.TXINTENA = 0;
837         ScibRegs.SCICTL2.bit.RXBKINTENA = 0;
838
839         ScibRegs.SCIHBAUD = 0x0001;
840         ScibRegs.SCILBAUD = 0x0024;
841
842         ScibRegs.SCICTL1.all = 0x0023;
843
844         ScibRegs.SCIFFTX.all = 0xE040;
845         ScibRegs.SCIFFRX.all = 0x0022;
846         ScibRegs.SCIFFCT.all = 0x0;
847
848         ScibRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
849         ScibRegs.SCIFFRX.bit.RXFIFORESET = 1;
850     }
851
852     void enviaSCIb(int a)
853     {
854         while (ScibRegs.SCIFFTX.bit.TXFFST != 0)
855         {
856
857         }
858         ScibRegs.SCITXBUF = a;
859     }
860
861     void msgSCIb(char *msg)
862     {
863         int i;
864         i = 0;
865         while (msg[i] != '\0')
866         {
867             enviaSCIb(msg[i]);
868             i++;
869         }
870     }
```