_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DESARROLLO DE UN ESTIMULADOR ELÉCTRICO PROGRAMABLE PARA EL ENTRENAMIENTO DE CORAZONES BIOARTIFICIALES

Roberto Montañana Grau

Tutor: Dra. María de la Salud Guillem Sánchez

Cotutor: D. Jorge Pedrón Torrecilla

Cotutor: Dr.Andreu Martínez Climent

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2013-14

Valencia, 3 de septiembre de 2014

Resumen

Diversos estudios médicos han concluido que es posible regenerar tejido cardíaco a partir de células madre. A pesar de ello, la tasa de supervivencia de dichos corazones resulta inferior al 10%. Un preacondicionamiento electromecánico de las células a implantar será efectivo a la hora de aumentar el efecto paracrino, puesto que las células estarían ya adaptadas a un entorno con estiramiento cíclico y, adicionalmente, el entrenamiento promoverá la diferenciación hacia tejido cardiaco adulto. Por ello, en este proyecto se ha diseñado e implementado un estimulador eléctrico programable de dos canales para entrenar corazones bioartificiales, permitiendo la selección de la amplitud del pulso del estímulo, el ancho del pulso y el periodo entre pulsos.

Se explicará cómo se ha planteado la solución al proyecto, empezando por el hardware. Al ser un estimulador programable se requiere un microcontrolador que sirva de unidad de control del hardware, por lo que se detallará también el firmware que se le ha programado. También, pensando en la sencillez de uso para el futuro usuario, se ha realizado un programa gráfico en .NET que de forma transparente al usuario se comunicará con el hardware.

Por ultimo pero no menos importante, se ha realizado un presupuesto de nuestro proyecto y se expondrán las conclusiones y propuestas de mejora.

Resum

Diversos estudis mèdics han conclòs que es possible regenerar teixit cardíac a partir de cèl·lules mare. A pesar d'això, la tassa de supervivència de dits cors resulta inferior al 10%. Un precondicionament electromecànic de les cèl·lules mare a implantar serà efectiu a l'hora de augmentar l'efecte paracrino, ja que les cèl·lules estarien ja adaptades a un entorn amb estirament cíclic i, addicionalment, l'entrenament promourà la diferenciació cap a teixit cardíac adult. Per això, en aquest projecte s'ha dissenyat i implementat un estimulador elèctric programable de dos canals per entrenar cors bioartificials, permetent la selecció de l'amplitud del pols de l'estímul, l'amplada de l'estímul i el període entre polsos.

S'explicarà com s'ha plantejat la solució al projecte, començant pel hardware. Al ser un estimulador programable, es requereix un microcontrolador que servisca d'unitat de control del hardware, raó per la que es detallarà també el firmware que se li ha programat. També, pensant en la senzillesa d'us per al futur usuari, s'ha realitzat un programa gràfic en .NET que de forma transparent a l'usuari es comunicarà amb el hardware.

Per últim, però no menys important, s'ha realitzat un pressupost del nostre projecte i s'exposaran les conclusions i propostes de millora.

Abstract

Several medical studies have concluded that it is possible to regenerate cardiac tissue from stem cells. However, the survival rate of those hearts is less than 10%. A previous electromecanic conditioning of the stem cells to implant, can be effective to improve the paracrine effect, as the stem cells would be already adapted to an environment with cyclic elongation, and also, the training will promote the differentiation to an adult cardiac tissue. Because of this, in this project an electric programmable stimulator with two channels for training bioartificial hearts has been designed and developed, allowing the selection of the amplitude of the stimulus, the width of the stimulus or the period between pulses.

It will be explained how the solution to the project has been outlined, beginning with the hardware. Because it is a programmable stimulator, a microcontroller that is used as a control unit for the hardware is required; therefore, the firmware that has been programmed into it will be detailed. Also, considering the ease of use for the future user, a graphic program in .NET that

communicates with the hardware transparently from the user's point of view has been developed.

Last but not least, a quote for the project has been made, and the conclusions and proposals for improving the project have been exposed.

Índice

Índice de figuras

Capítulo 1. Introducción

1.1 Motivación.

En primer lugar explicaremos cual es el contexto en el que se va a emplear el estimulador eléctrico que hemos desarrollado.

En los últimos años se han desarrollado numerosos ensayos clínicos de terapia celular que han perseguido regenerar el tejido dañado mediante la administración de células madre adultas [1], [2], [3], [4]. El principal efecto cardiorregenerativo de la inyección de células madre adultas es fundamentalmente paracrino, es decir, debido a la liberación de citoquinas y otros factores que potencian la vascularización del tejido [5], [6], [7], [8]. Los resultados han demostrado que la implantación de células madre adultas es una técnica segura y mejora la función cardiaca, sin embargo, los índices de retención celular y supervivencia han sido inferiores al 10% [9].

Un preacondicionamiento electromecánico de las células a implantar será efectivo a la hora de aumentar el efecto paracrino, puesto que las células estarían ya adaptadas a un entorno con estiramiento cíclico y, adicionalmente, el entrenamiento promoverá la diferenciación hacia tejido cardiaco adulto. Diversos autores han estudiado el efecto beneficioso de un entrenamiento eléctrico o mecánico en cultivos de células procedentes de corazones neonatos durante su maduración. Eschenhagen y Zimmerman demostraron la eficiencia de un entrenamiento mecánico durante el cultivo de células cardiacas neonatales [10], [11]. Bajo estas condiciones, las células formaron músculo cardiaco altamente diferenciado que exhibía propiedades contráctiles y electrofisiológicas más similares al miocardio adulto que aquellos tejidos que no habían sido entrenados.

En conclusión, todos estos estudios han destacado la importancia del entrenamiento en el proceso de maduración del tejido miocárdico. Dado que la contracción mecánica de los miocitos está mediada principalmente por una estimulación eléctrica, el estimulador eléctrico para el entrenamiento de los tejidos miocárdicos se plantea como una herramienta fundamental para la maduración celular.

1.2 Otras necesidades técnicas.

En este apartado pasaré a describir otros sistemas técnicos, que a pesar de no ser parte de nuestro desarrollo, se emplearán conjuntamente con él por ser necesarios para la supervivencia de las células.

1.2.1 Biorreactor.

Para lograr la supervivencia y diferenciación de las células que constituirán el tejido bioartificial miocárdico, éste será mantenido en un biorreactor previamente a su implante.

Un biorreactor consta fundamentalmente de una cámara que permite el mantenimiento y crecimiento del tejido bioartificial en su interior en condiciones fisiológicas adecuadas de perfusión, entorno y esterilidad.

Este biorreactor se empleará conjuntamente con el estimulador eléctrico programable que hemos desarrollado para proporcionar un entrenamiento electromecánico que simule las condiciones físicas del corazón. Esta estimulación electromecánica provocará una mayor eficiencia en el proceso de diferenciación y una mayor especialización en las células progenitoras, lo cual provocará un aumento en el número de citoquinas y factores de crecimiento específicos secretados por el implante y que deberán permitir una mayor y más rápida recuperación de la función cardiaca.

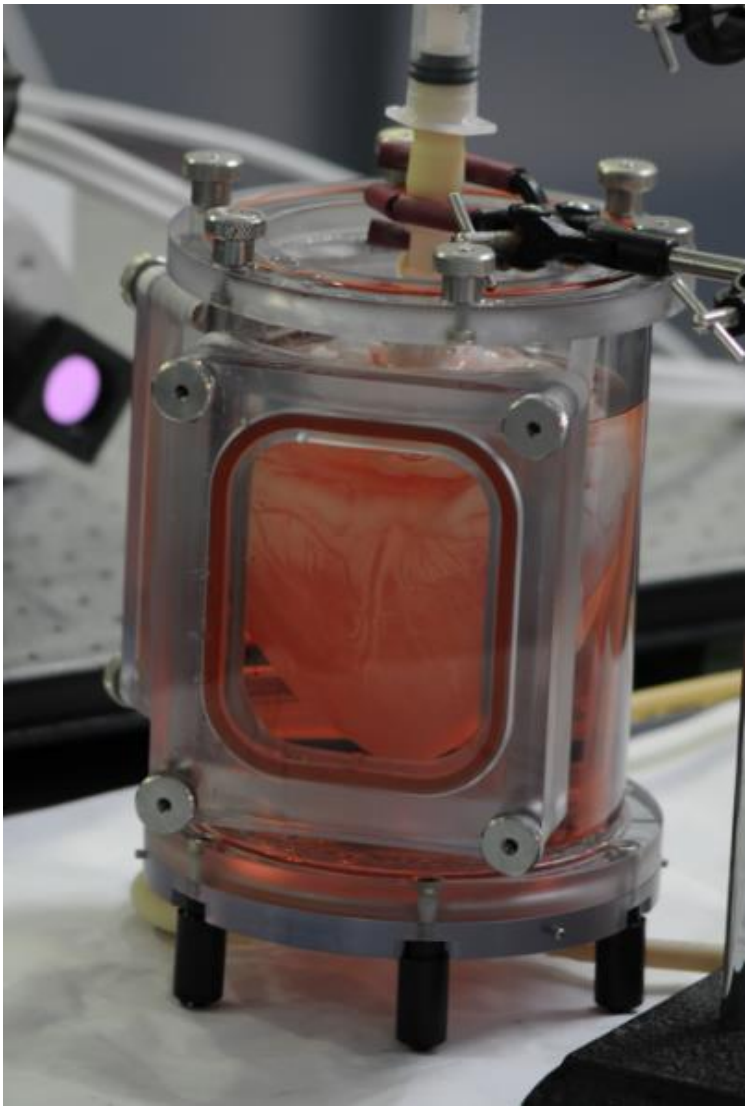

Ilustración 1. Corazón en el interior del biorreactor

En concreto, el biorreactor que se empleará en conjunción con este proyecto incluye una cámara de metacrilato con conductos de teflón. La cámara incluye un sistema de soporte para fijar el tejido durante el proceso de maduración. La perfusión se realiza mediante difusión, facilitada por la circulación de medio en la cámara controlada por la utilización de bombas peristálticas tanto a la entrada como a la salida, reguladas automáticamente en función de la presión ejercida por el medio sobre el tejido. Este sistema permite la reposición del medio tras ser filtrado, oxigenado y calentado en el exterior de la cámara, haciendo uso de filtros, baño termorregulador, y oxigenadores.

Capítulo 2. Objetivos del TFG

El objetivo del proyecto es el desarrollo e implementación de un estimulador eléctrico programable de dos canales para entrenar corazones bioartificiales. Para ello se tienen que cumplir los siguientes objetivos:

- Diseño del esquemático del hardware del estimulador.
- Programar el firmware del Arduino para permitir el control y generación de un estímulo programable, así como la carga y realización de protocolos de estimulación.
- Programar el software de control con el PC.
- Implementar en una placa de prototipo un canal del estimulador, y testear conjuntamente que todo el sistema, hardware, firmware, y software, funcionan correctamente.
- Diseñar y montar la placa PCB del estimulador.
- Testear el funcionamiento completo del estimulador bajo las condiciones necesarias para su incorporación al sistema.

Capítulo 3. Metodología de trabajo del TFG.

3.1 Gestión del proyecto.

Éste proyecto se realizará en el Instituto Universitario ITACA de la Universitat Politècnica de València y en colaboración con el Hospital General Universitario Gregorio Marañón, bajo la supervisión de la Dra. María Guillem Sanchez, el Dr. Andreu Martínez Climent, y D. Jorge Pedrón Torrecilla.

Ya que se trata de un sistema electrónico que trabajará de forma conjunta con otros sistemas, tanto electrónicos, como mecánicos, se supervisará el avance del proyecto cada vez que se finalice cada una de las tareas asignadas para la consecución del proyecto, analizando los resultados obtenidos a la fecha para comprobar que se ciñen a las especificaciones impuestas.

Dentro del marco de colaboración, el Instituto Itaca facilitará los materiales y herramientas necesarias para la realización del proyecto: ordenador, osciloscopio, multímetro, estación de soldadura, componentes necesarios para la fabricación del proyecto, y fabricación de la PCB.

3.2 Distribución en tareas.

1. Estudio del diseño de estimuladores eléctricos ya desarrollados para conocer el funcionamiento y mejorar el hardware.
2. Diseño del esquemático del nuevo estimulador y elección de los componentes electrónicos.
3. Programación del firmware del microcontrolador.
4. Programación del software de control para PC.
5. Montaje en placa de prototipo de un único canal del estimulador.
6. Test conjunto, software, firmware, hardware, y comprobación de que el estimulador puede proporcionar estímulos bajo todas las condiciones solicitadas.
7. Diseño de una PCB para el estimulador de dos canales.
8. Montaje de una PCB para el estimulador de dos canales.
9. Test del estimulador montado en PCB con dos canales.

3.3 Diagrama temporal.

	JUL	AGO	SEP	OCT	NOV	DIC	ENE	FEB	MAR	ABR	MAY	JUN
1	X											
2	X	X										
3		X	X	X	X	X	X	X				
4			X	X	X	X	X	X	X			
5			X	X	X	X	X	X	X	X	X	
6								X	X	X	X	X
7												X
8												X
9												X

Capítulo 4. Desarrollo y resultados del trabajo.

4.1 Descripción de la solución empleada.

4.1.1 Hardware.

4.1.1.1 Diagrama de bloques del sistema.

En esta sección mostraremos un diagrama de bloques del sistema que servirá como referencia para entender cómo funciona el sistema en todo su conjunto. Posteriormente se analizarán cada uno de los bloques para entender con más detalle su funcionamiento y elección de los componentes (Ilustración 2).

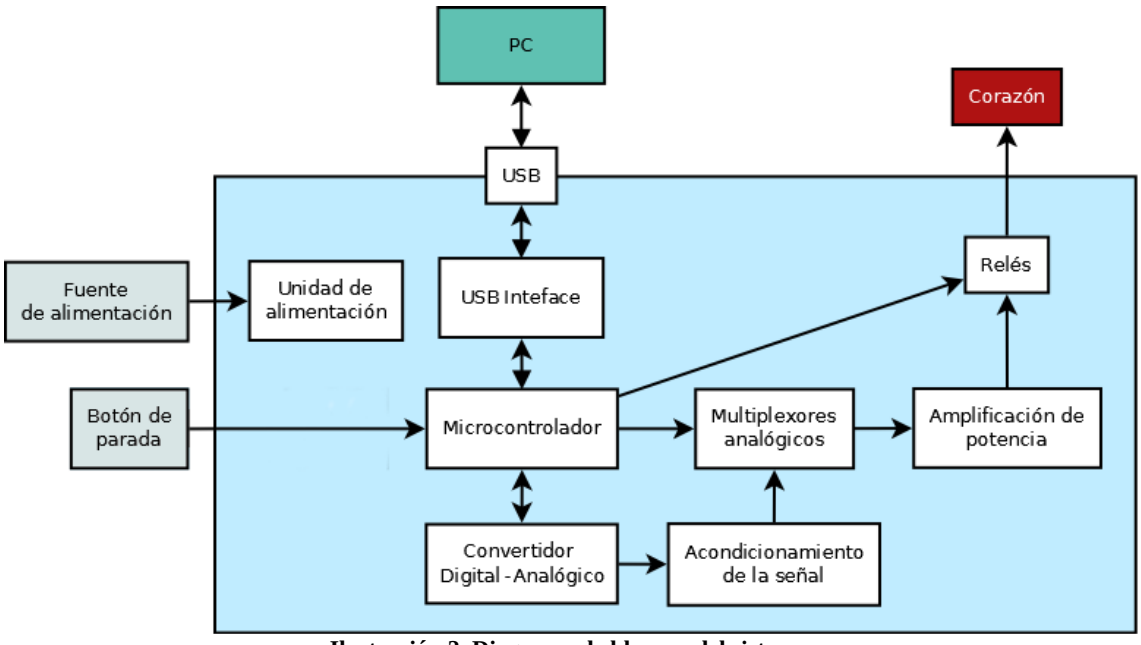

Ilustración 2. Diagrama de bloques del sistema.

En el ordenador hemos programado un software que nos sirve como interfaz de usuario para comunicarnos con nuestro sistema. Dicho programa nos permitirá cambiar todos los parámetros de nuestro estímulo; tensión, ancho del pulso y periodo del estímulo. Nos permite a su vez seleccionar cuál de los dos canales queremos configurar y si queremos que se aplique un protocolo de estimulación. Todo esto se hace de forma transparente al usuario, generando y enviando por el USB, conectado a la interfaz USB-serie de la Arduino, los comandos necesarios para reconfigurar nuestro sistema.

Por otra parte, en el microcontrolador se ejecuta un firmware que está continuamente escuchando su puerto serie. En el caso de que reciba un comando por parte del ordenador, modifica las variables de las que dispone para generar el estímulo que deseamos.

Para poder generar la tensión que hemos configurado para el estímulo, el microcontrolador se comunicará con un convertidor digital-analógico (DAC), el cual, a partir de lo indicado por el microcontrolador, generará una tensión de salida determinada.

Nuestro estimulador es bifásico, por lo tanto necesitamos una tensión tanto positiva como negativa. Como nuestro DAC es unipolar y solo genera la tensión positiva, diseñaremos una etapa de acondicionamiento que a partir de la señal proveniente del DAC nos genere también la tensión negativa.

Dispondremos también de unos multiplexores analógicos que siendo temporizados correctamente, controlarán cuando a la siguiente etapa no pasa señal, pasa la tensión positiva o la tensión negativa, dando de esta manera forma al pulso bifásico.

A continuación diseñaremos un bloque que nos realizará la amplificación de potencia, tanto en tensión para llegar a los 40V que exige el pliego de condiciones, como en corriente.

La última etapa de la cadena es un bloque de relés que nos permiten desconectar y aislar completamente el estimulador al corazón del sistema cuando no se está aplicando un estímulo. Esto es necesario para evitar posibles corrientes de retorno del corazón a nuestro sistema.

Necesitamos también por condición del proyecto un botón que nos permita detener el estímulo en caso de una emergencia.

Por último todo nuestro sistema tiene que ir alimentado. Además de las fuentes de alimentación, nuestro sistema necesitará un subsistema de alimentación que genere a partir de las fuentes, todas las tensiones necesarias para nuestro sistema.

4.1.1.2 Elección del microcontrolador.

En nuestro diseño necesitamos un microcontrolador que nos genere una señal programable.

Dicho microcontrolador deberá cumplir las siguientes condiciones:

- Deberá poder conectarse vía USB a un ordenador, de forma que sus parámetros de estímulo puedan ser reconfigurados de forma transparente al usuario.
- Para obtener la amplitud del estímulo a partir de una variable del microcontrolador emplearemos un DAC. Dicho DAC puede funcionar a través de un bus paralelo o de un bus serie. El uso de un bus serie nos permite emplear un menor número de líneas en la placa. La tasa de transferencia de buses serie, como I^2C a 100kHz, o aún más veloces SPI a 1MHz, son más que suficientes para nuestra aplicación. Por lo tanto, buscaremos un microcontrolador que disponga entre sus periféricos de controladores para comunicaciones serie.
- Nuestro sistema dispondrá de un pulsador para detener la estimulación en caso de emergencia. Para no tener que estar haciendo polling continuamente al pin de entrada y que coincidiese con la pulsación del botón, buscaremos un microcontrolador que nos permita configurar una interrupción externa a uno de los pines para detectar la pulsación.
- Para obtener el periodo entre estímulos de una forma programada, el microcontrolador deberá tener un timer y una interrupción de timer que controle el tiempo transcurrido.

Buscando entre las diferentes opciones que nos ofrecía el mercado, optamos por elegir como microcontrolador un Arduino Uno (Ilustración 3), el cual pasaremos a describir a continuación, y comprobaremos que cumple todas las especificaciones pedidas.

Arduino es una familia de placas de desarrollo de microcontroladores basadas en microcontroladores de la arquitectura ATMEL AVR y en microcontroladores de arquitectura ARM.

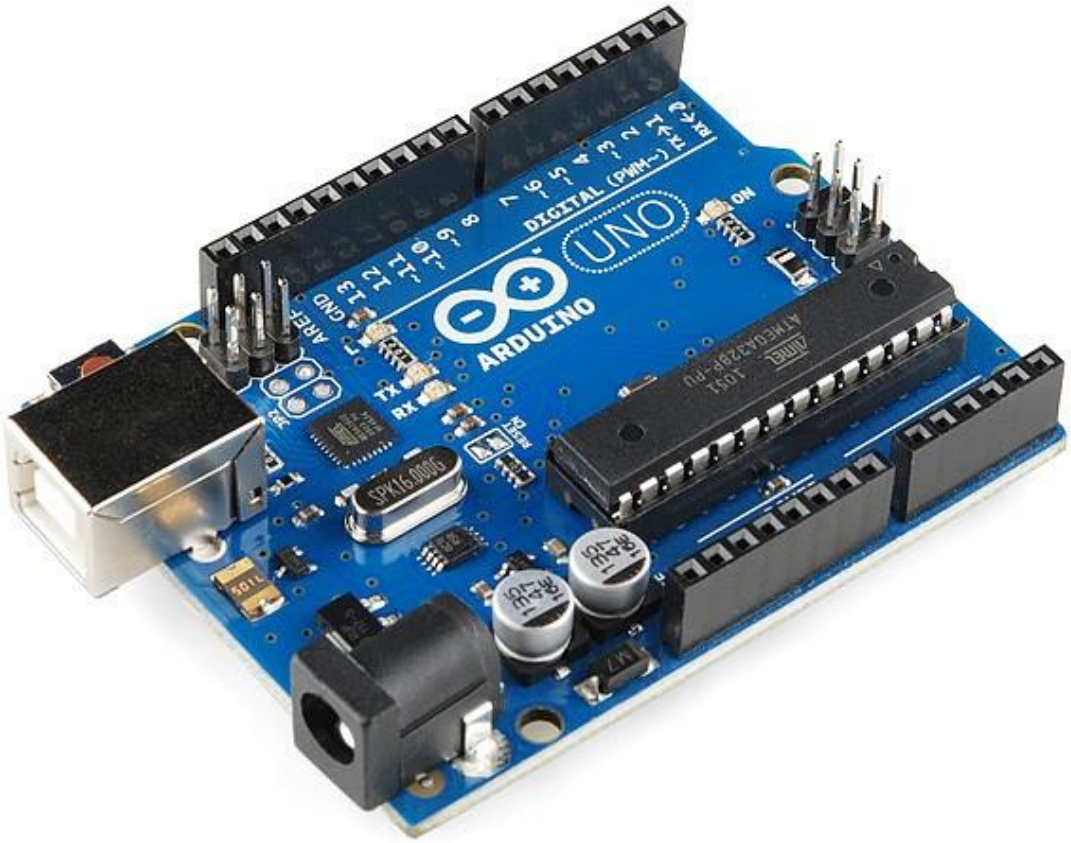

Ilustración 3. Placa del Arduino Uno

El Arduino Uno es una placa de esta familia basada en el microcontrolador ATmega328. Sus características principales son las que figuran en la Tabla 1 [12].

Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje de entrada recomendado	7-12V
Pines digitales E/S	14 (6 disponen de salida PWM)
Pines de entrada analógica	6
Consumo por pin E/S	40mA
Consumo del pin 3.3V	50mA
Memoria flash	32kB (0.5kB empleados por el bootloader)
SRAM	2kB

EEPROM	1kB
Frecuencia de reloj	16MHz

Tabla 1. Resumen de características del Arduino Uno

El Arduino puede ser alimentado a través de su puerto USB o de una fuente de alimentación externa. La alimentación externa se puede conectar a través del jack del que dispone la placa o a través de los pines GND y VIN. Ya que deseamos que en nuestro sistema se pueda desconectar el cable USB y que siga funcionando, dicha alimentación queda descartada. Nuestro Arduino irá montado sobre una PCB principal que dispondrá de alimentación, por lo que elegimos alimentarlo a través de los pines GND y VIN.

La placa del Arduino dispone de un regulador lineal de 5V que tomará la tensión de entrada de entre 7-12V y generará una tensión de 5V que sirve para alimentar el microcontrolador. Adicionalmente dispone de un pin en la placa llamado 5V conectado a la salida de dicho regulador, por lo que nos sirve como pin de alimentación para el resto de dispositivos externos al Arduino que tengamos alimentados a 5V [12].

El Arduino Uno dispone de un puerto USB y de un Atmega16U2 programado como convertidor USB a puerto serie, por lo que nos permite comunicar el ordenador con el Arduino estableciendo una conexión con el puerto serie emulado [12].

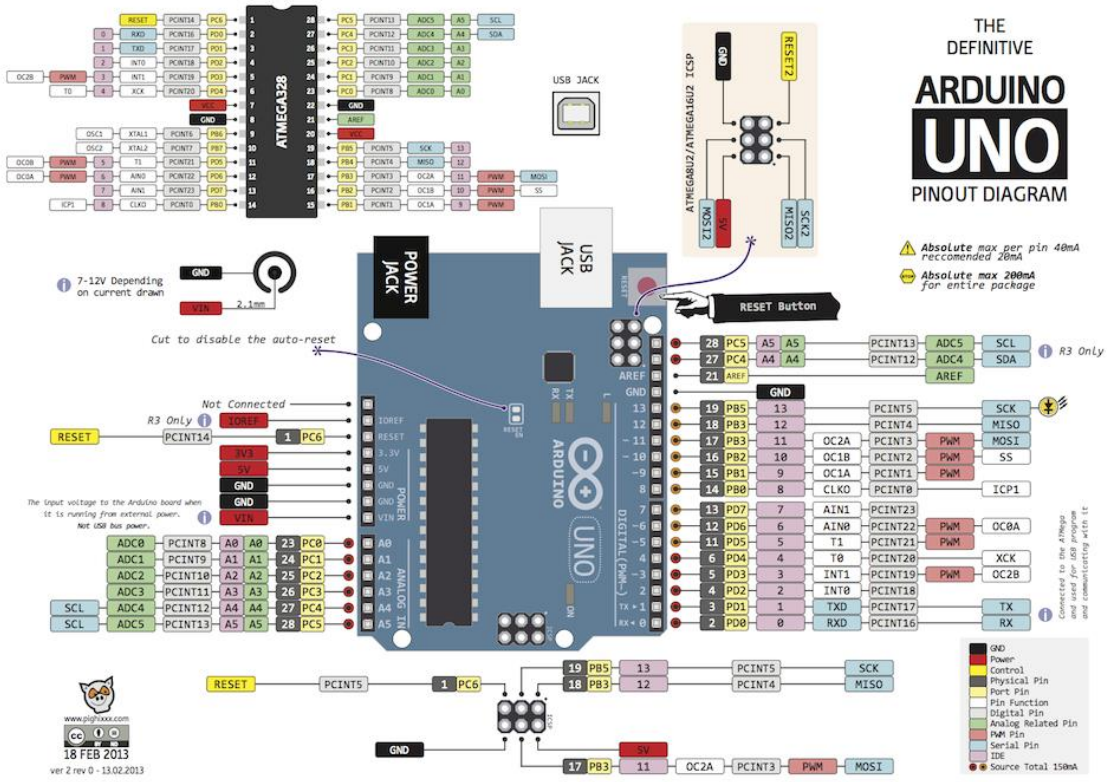

Ilustración 4. Pinout del Arduino Uno

En la Ilustración 4 podemos ver el pinout del Arduino y como van conectados a los distintos pines del ATMega328. De los 14 pines digitales de E/S que dispone, algunos disponen de funciones adicionales [12]. Las más importantes de cara a nuestro proyecto son:

- Serie: pines 0 y 1. Usados como pines RX y TX de la UART. Ya que la comunicación UART con el PC a través del convertidor USB-Serie es imprescindible en nuestro proyecto, estos pines deberán estar configurados para dicho propósito y no podrán ser empleados como E/S.

- Interrupciones externas: pines 2 y 3. Estos pines se pueden configurar para disparar una interrupción por nivel bajo o por flanco. Reservaremos uno de estos pines para la detección de la pulsación de la parada de emergencia.
- SPI (Serial Peripheral Interface): pin 10 (Chip Select), pin 11(MOSI), pin 12(MISO), pin 13(SCK).

De los seis pines de entrada analógica, hay dos que disponen de una segunda funcionalidad útil en nuestro proyecto.

- TWI (Two Wire Interface): pin A4 (SDA) y pin A5 (SCL). TWI es el nombre que le dan Atmel y otros fabricantes al bus I^2C.

Adicionalmente, el ATmega328 dispone de dos timers de 8 bits y de un timer de 16 bits, con sus respectivas interrupciones, que nos permitirán controlar el tiempo transcurrido y por lo tanto temporizar los estímulos [13].

Para facilitar el control del timer, hay una función para Arduino llamada MsTimer2 que emplea uno de los timers de 8 bits, con la cual, simplemente hay que configurar cada cuantos milisegundos se desea que salte la interrupción, y a que función tiene que acudir [14]. Por su sencillez, emplearemos dicha librería para controlar la temporización del estímulo.

4.1.1.3 Elección del DAC.

Para generar la tensión de salida deseada, recurriremos a un DAC. Queremos emplear pocas líneas para comunicar el microcontrolador con el DAC por lo que buscaremos DAC que tengan comunicación por medio de algún protocolo de comunicación serie, más concretamente, I^2C o SPI.

Como DAC elegimos el MCP4726 de Microchip. Este DAC permite seleccionar como tensión de referencia la tensión de alimentación que es 5V. Al ser rail-to-rail, la tensión máxima de salida será de 5V [15], por lo que posteriormente tendremos que amplificar y acondicionar la señal para obtener los 40V máximos que deseamos tener.

El DAC tiene 12 bits de resolución [15], cuyo valor máximo corresponde a una salida de 5V. Como queremos tener a la salida de nuestro sistema una tensión de 40V, cuando programemos en el firmware que deseamos dicha tensión máxima, al DAC tendrá que pasarse el siguiente valor:

$$V_{DAC} = \frac{V_{Deseado} \cdot 4095}{5 \cdot 8} + 0.5 \qquad (4.1)$$

La adición de 0.5 se emplea ya que al pasar una variable de coma flotante a entera, el microcontrolador redondea al entero inferior; con la suma de 0.5 podemos redondear al entero más próximo.

En la Ilustración 5 podemos observar el patillaje del MCP4726:

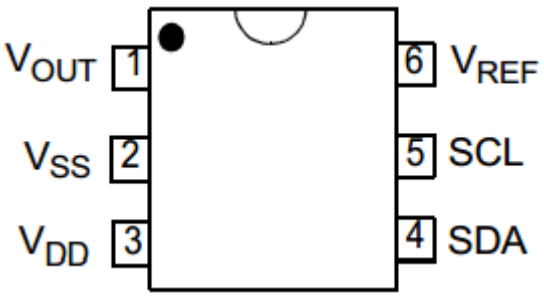

Ilustración 5. Patillaje del MCP4726

Los pines que emplearemos son los siguientes:

- V_{OUT}: Salida de tensión analógica del DAC.

- V_{SS}: Alimentación negativa del DAC, en nuestro caso 0V.
- V_{DD}: Alimentación positiva del DAC, en nuestro caso 5V.
- SDA: Pin de conexión al bus SDA del I^2C.
- SCL: Pin de conexión al bus SCL del I^2C.

Este DAC se comunica con el microcontrolador por medio del protocolo I^2C, el cual identifica los dispositivos conectados en el bus por medio de una dirección de esclavo. En nuestro caso, la ventaja que nos aporta I^2C, es que para obtener los dos canales de estimulación independientes, únicamente tenemos que poner dos MCP4726 en nuestra placa, uno con dirección cero y otro con dirección uno [16]. Esta facilidad para emplear distintos periféricos en un mismo bus es lo que decantará el uso del bus I^2C en el proyecto, ya que la elevada velocidad de transferencia del bus SPI no es necesaria para nuestra aplicación y, de esta forma, optimizamos el número de pines utilizados a dos únicos pines, como se explica a continuación.

4.1.1.4 Protocolo I²C.

El bus I^2C fue desarrollado por Philips en los años ochenta, que tiene la finalidad de comunicar periféricos de baja velocidad [17]. Las velocidades de transferencia que emplearemos en nuestro proyecto será la del modo estándar, 100kbit/s.

Es un protocolo que únicamente emplea dos líneas, una línea de reloj (SCL), y otra línea para datos (SDA), las cuales se conectan a todos los dispositivos que pertenecen al bus. Estas líneas son pull-up, lo que implica que el controlador puede ponerlas a nivel bajo pero no a nivel alto (nivel al que el bus estará por defecto), por lo que hay que conectar una resistencia entre cada una de las líneas y VDD [18].

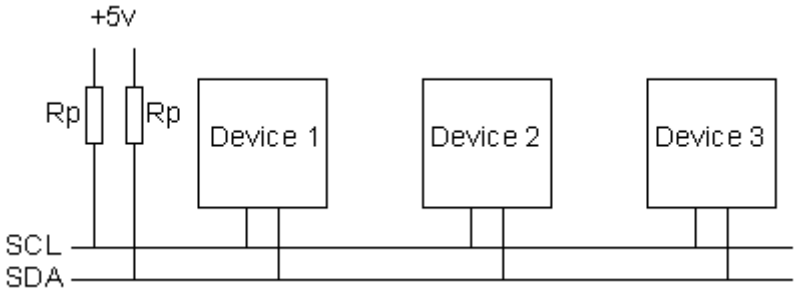

Ilustración 6. Conexión del bus I2C

Se basa en una comunicación maestro-esclavo. El maestro es el dispositivo que controla la línea de reloj y es el que puede iniciar las comunicaciones, en cambio, el esclavo tiene que estar siempre escuchando las líneas para detectar el inicio de una comunicación. Lo habitual es disponer únicamente de un maestro, que en nuestro caso es el microcontrolador, que interaccionará con los múltiples esclavos conectados al bus, diferenciados por una dirección única de esclavo [18].

El funcionamiento del protocolo es el siguiente:

1. Para empezar una comunicación, el maestro genera el bit de start, consistente en poner la línea SDA a nivel bajo mientras la línea SCL está a nivel alto, es junto al bit de stop, los únicos que pueden cambiar la línea SDA mientras la línea SCL está a nivel alto. A continuación vendrán los bits de datos, durante los cuales, la línea SDA cambia mientras la línea SCL está a nivel bajo. (Ver Ilustración 7)

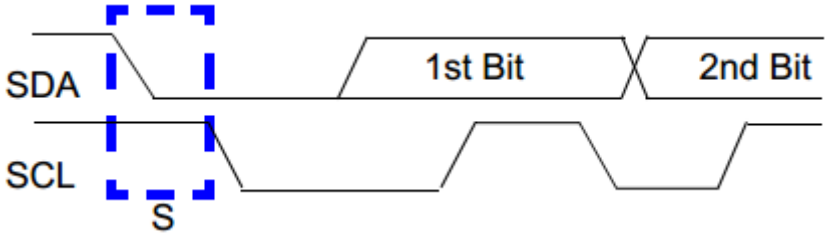

Ilustración 7. Bit de start del I2C

2. El primer byte que se transmite después del bit de start es el de dirección que consiste en los siete bits de la dirección de esclavo, y el bit de lectura/escritura que se pone a uno para indicar que se quiere hacer una lectura del esclavo, o a cero si se quiere hacer una escritura en el esclavo. (Ver Ilustración 8)

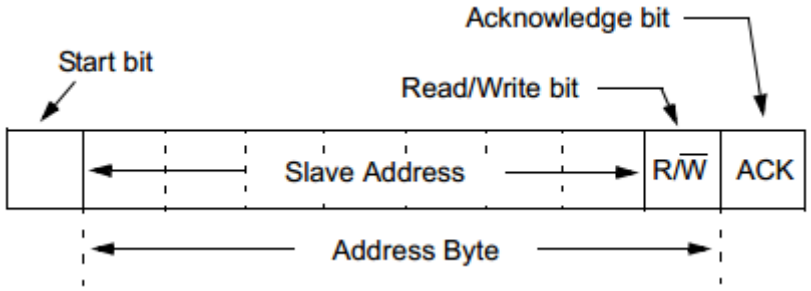

Ilustración 8. Byte de dirección.

3. Al terminar de transmitir cada byte, el dispositivo receptor transmitirá un bit de ACK. El bit de ACK consiste en mantener la línea a nivel bajo. En caso de que haya algún error, como por ejemplo que el esclavo con el que se quiere comunicar no se encuentre en el bus, en vez de mantenerse la línea a nivel bajo, se pondrá a nivel alto, siendo esto un bit de NACK. (Ver Ilustración 9).

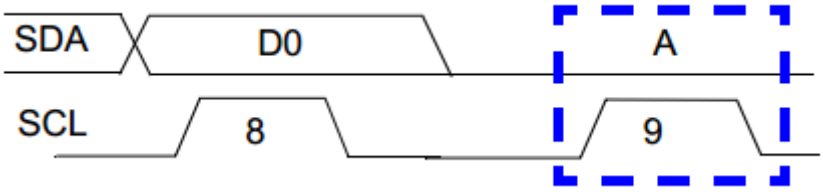

Ilustración 9. Bit de ACK.

4. Al terminar de transmitir el último byte y después del bit de ACK/NACK, se transmite el bit de stop, que consiste en poner la línea SDA a nivel alto mientras la línea SCL está a nivel alto. (Ver Ilustración 10).

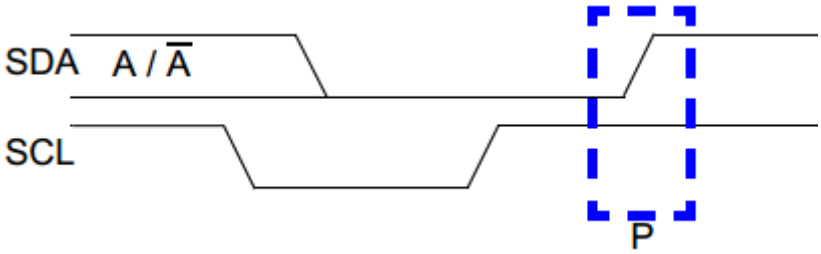

Ilustración 10. Bit de stop.

4.1.1.5 Acondicionamiento de la señal: generación de la tensión bipolar.

A la salida del DAC tenemos una tensión programable que se encuentra en el rango de los 0V a los 5V.

Para conseguir un pulso bipolar, tendremos que amplificar la señal. La primera etapa de amplificación consistirá en un amplificador no inversor de ganancia 2 y un inversor de ganancia 1 en cascada, implementados con dos amplificadores operacionales, obteniendo una señal positiva y negativa entre -10 y +10 V.

Como amplificadores elegiremos el LT1097 de Linear Technology, el cual es un amplificador de precisión de bajo consumo. Por ejemplo, su tensión de offset de entrada es como máximo de 50μV [19], que superpuesta a la señal que deseamos amplificar, con un rango de 0 a 5V, en pasos de 1V, es despreciable y no es necesario corregirla.

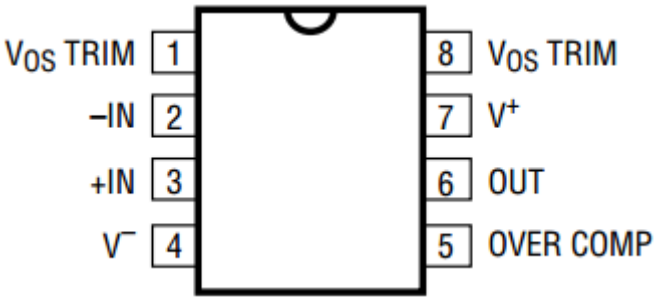

Ilustración 11. Patillaje del LT1097

En la Ilustración 11 podemos ver el patillaje del LT1097, siendo los siguientes pines los que emplearemos en nuestro sistema:

- -IN: Terminal inversor del amplificador operacional.
- +IN: Terminal no inversor del amplificador operacional.
- V-: Alimentación negativa, de -15 a 0V.
- OUT: Salida del amplificador operacional.
- V+: Alimentación positiva de 0 a +15V.

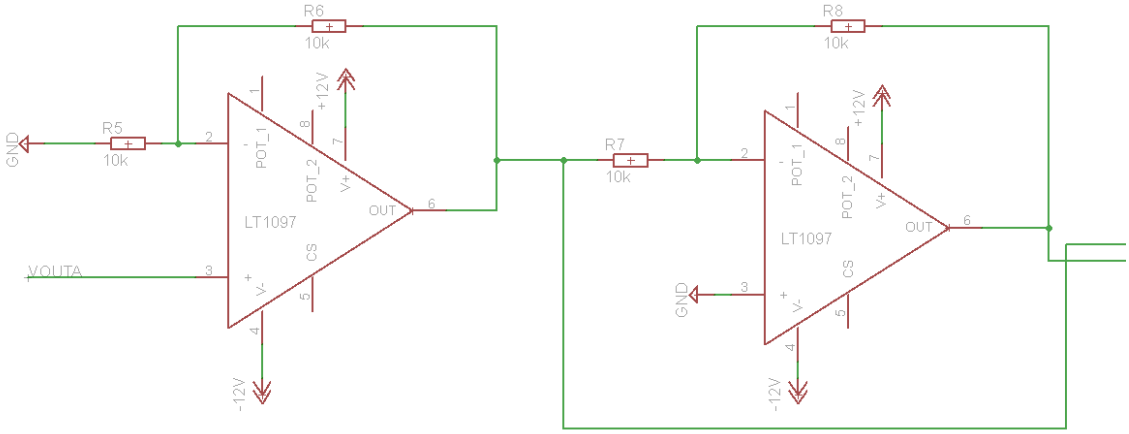

Ilustración 12. Primera etapa de amplificación.

En la Ilustración 12 podemos ver el conexionado que hemos realizado con los amplificadores operacionales para obtener dos tensiones de $\pm$10V.

El primer amplificador de la etapa es un amplificador no inversor de ganancia dos, por lo que a su salida tenemos una tensión de +10V.

El segundo amplificador actúa como amplificador inversor de ganancia unidad, por lo que a su salida tenemos una salida de -10V.

Podemos ver entonces que a la salida de la etapa de amplificación tenemos dos señales, una con +10V y otra con -10V, por lo que ya hemos conseguido ambas polaridades a partir de una única señal unipolar del DAC.

4.1.1.6 Multiplexado analógico: Generación del pulso bifásico:

Una vez tenemos las dos tensiones deseadas, tenemos que generar un pulso bifásico de ancho programable.

Para poder modular la señal emplearemos un multiplexor analógico, concretamente el ADG408 de Analog Devices.

El ADG408 es un multiplexor analógico de 8 canales de entrada y una salida. La salida se controla por medio de tres bits digitales de control, como podemos ver en su diagrama funcional en la Ilustración 13.

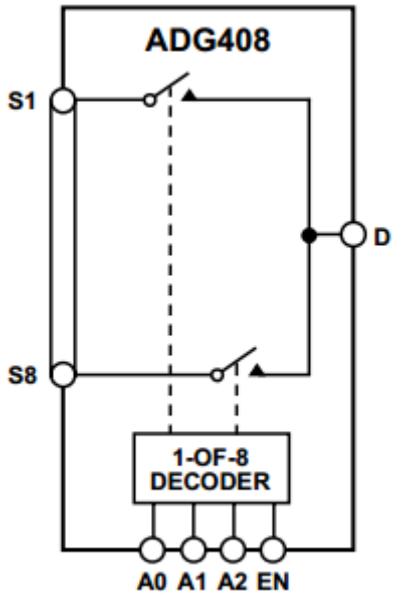

Ilustración 13. Diagrama funcional del ADG408

- Las señales S0…S8 son los ocho canales de entrada de que dispone el ADG408.
- Los bits A0…A2 son los tres bits binarios de control que seleccionan cual de dichos canales pasa a la salida D.
- Por último, la señal EN es una entrada digital activa a nivel alto. Si se encuentra a nivel bajo el dispositivo esta desconectado.

Las señales analógicas tienen que estar en el rango de Vss a Vdd. Alimentaremos el circuito con V_{DD}=+12V y V_{SS}=-12V dado que las señales de entrada tendrán como máximo una magnitud de 10V.

Para generar el pulso bifásico conectaremos el ADG408 como se observa en la Ilustración 14.

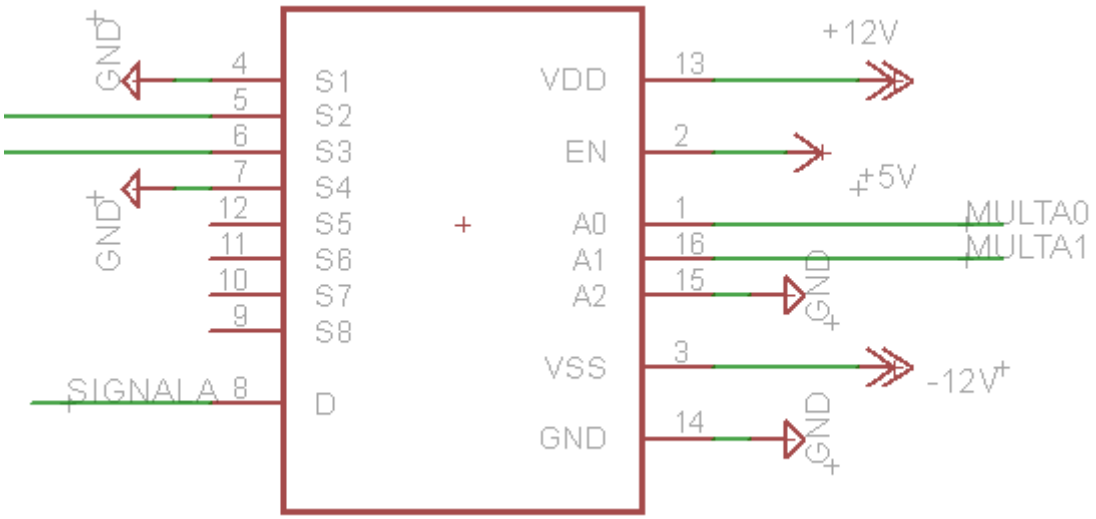

Ilustración 14. Conexionado del ADG408 para la generación del pulso bifásico.

El funcionamiento es el siguiente:

1. Inicialmente A1-A0 valen 00, por lo tanto en D tenemos la señal de masa (0V).
2. Cuando se tiene que generar un estímulo, el microcontrolador cambia el valor de A1-A0 a 01, por lo que la señal conectada a S2 pasa a la salida y tenemos +10V.
3. El microcontrolador hace un delay programado cuyo valor es la mitad del ancho del pulso indicado por el usuario, durante ese periodo de tiempo a la salida se tienen +10V.
4. Una vez pasa el delay, el microcontrolador cambia el valor de A1-A0 a 10, de forma que S3 pasa a la salida y tenemos -10V.
5. El microcontrolador vuelve a hacer un delay cuyo valor es la mitad del ancho del pulso indicado por el usuario, durante ese periodo de tiempo a la salida se tienen -10V.
6. Una vez pasa el delay, el microcontrolador cambia el valor de A1-A0 a 00, de forma que en la salida tenemos la señal a masa y se repite el procedimiento en el siguiente estimulo.

4.1.1.7 Amplificación de potencia.

Una vez conseguido el pulso bifásico, tenemos que amplificar en tensión y en corriente.

A la entrada de la etapa tenemos una tensión de entrada con un rango de 0 a 10V, por lo que tendremos que amplificar la tensión por cuatro. Esta amplificación nos la proporciona el OPA454 de Texas Instruments, que es un amplificador operacional diseñado para tensiones de alimentación de hasta ± 50V [20].

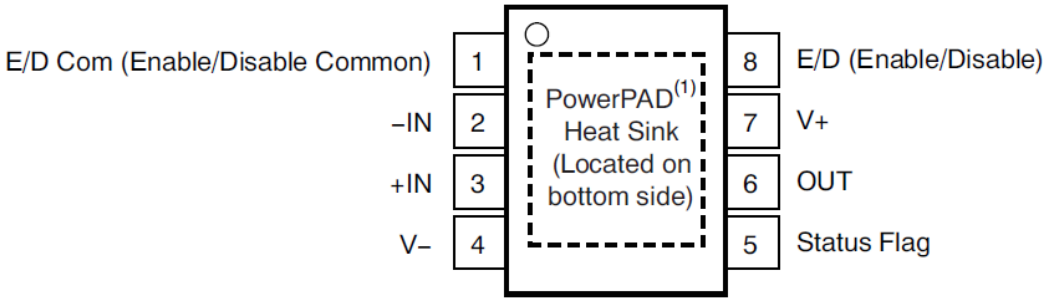

Ilustración 15. Pines del OPA454

En la Ilustración 15 tenemos el patillaje del OPA454.

Para amplificar en corriente pondremos una etapa push-pull clase B formada por dos transistores Darlington, TIP142 (NPN), y TIP147 (PNP), que dan una ganancia en corriente de más de 1000 [21], permitiendo la aplicación de estímulos con corrientes de hasta 1A

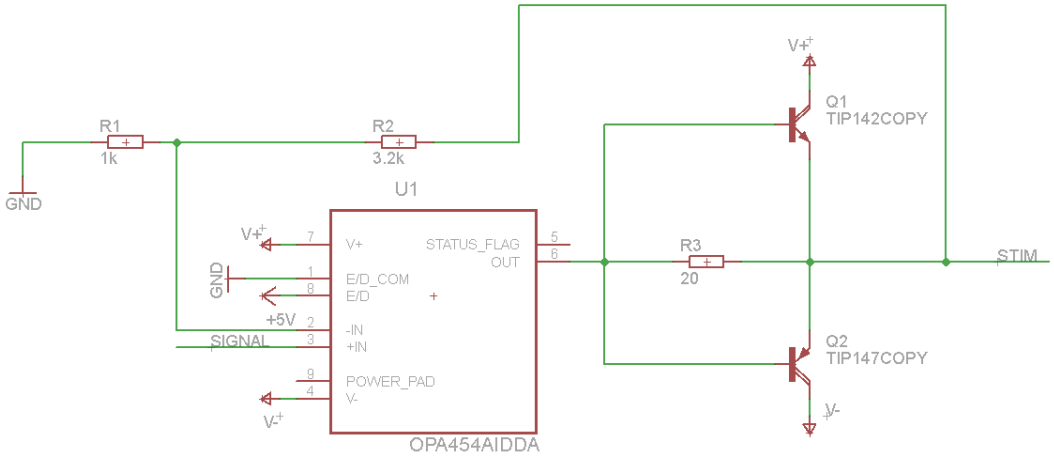

Ilustración 16. Etapa de amplificación de potencia

De esta forma, a la salida de ésta etapa tenemos un estímulo bifásico cuya amplitud se encuentra comprendida en el rango $\pm 40V$, con un ancho de pulso programable, y con un periodo de aplicación que podemos configurar a voluntad, tal y como queríamos. En los test comprobaremos que efectivamente, el estimulador puede suministrar 1A tal y como piden las especificaciones del proyecto.

4.1.1.8 Relé.

Una de las especificaciones del proyecto es que los electrodos y el sistema estén aislados del tejido a estimular en todo momento salvo cuando se aplique el estímulo. Esto se hace para evitar corrientes entrantes desde el corazón al sistema.

Para conseguir esto, entre la salida de la etapa de amplificación de potencia y los conectores de los electrodos, pondremos el relé C347S de Coto Technology.

Éste relé permite una tensión de carga de 80V, una corriente de 1A, y en el caso de corriente de pico, admite hasta 3A [22], por lo que es compatible con nuestra señal de estímulo.

Como podemos observar en la Ilustración 17, este relé consiste en un circuito integrado con dos optoacopladores. Para cada optoacoplador individual hay los siguientes pines:

- 1, 3: Ánodo del LED.
- 2, 4: Cátodo del LED.
- 5, 6, 7, 8: Terminales de los optoacopladores.

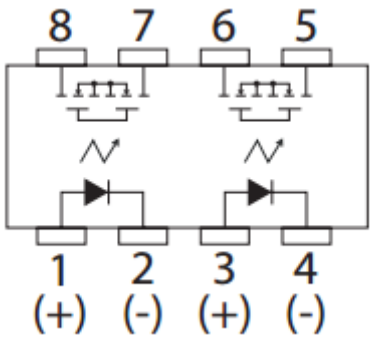

Ilustración 17. Patillaje del C347S

Para cada uno de los dos optoacopladores, cuando la tensión ánodo – cátodo es superior a 1.5V y la corriente que circula por el LED es superior a 1.2mA, el LED emitirá luz, y el MOSFET conducirá. En caso contrario el LED estará apagado y el MOSFET estará en corte [22].

En nuestro caso, el cátodo irá conectado a masa y el ánodo a una señal de control procedente del microcontrolador, que se pondrá a nivel alto únicamente cuando se vaya a realizar el estímulo. Por otra parte, un MOSFET conectará la señal del estímulo a un electrodo del estimulador y el otro la masa al otro electrodo del estimulador, de forma que el estimulador se pueda aislar completamente el corazón cuando no esté estimulando (Ilustración 18).

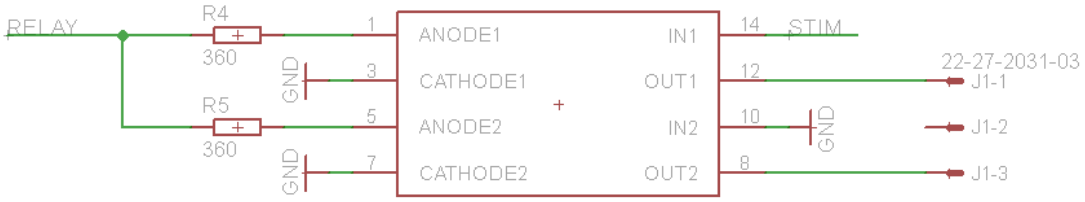

Ilustración 18. Conexionado de los relés del C347S.

4.1.1.9 Circuitería para la parada del estimulador.

Las especificaciones de nuestro proyecto indican que el estimulador debe poder deternerse de forma inmediata en caso de emergencia mediante un pulsador.

Para ello emplearemos la interrupción externa del pin 2 del Arduino.

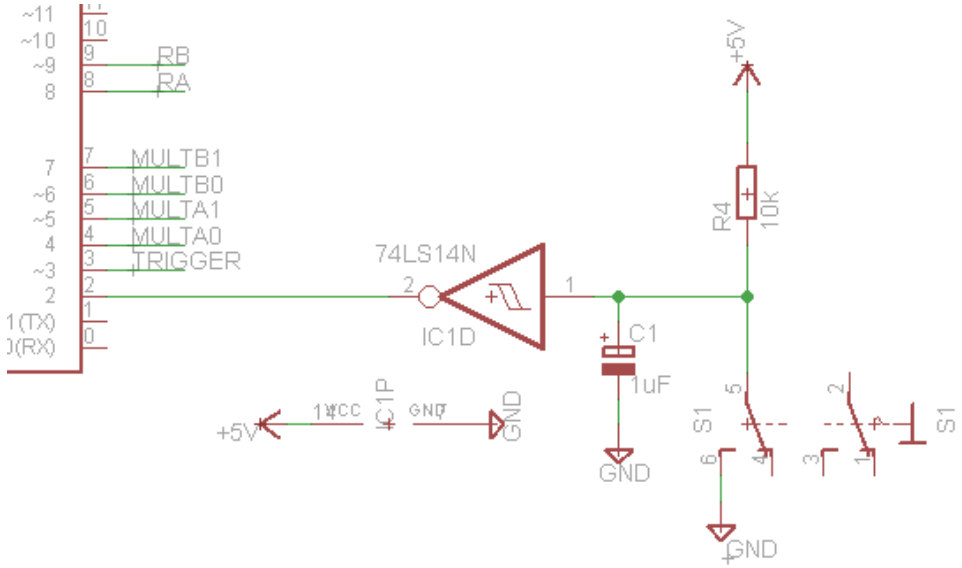

Ilustración 19. Circuitería para detener el estimulador

Como podemos ver en la Ilustración 19, el pulsador actúa de manera que al no estar pulsado, a la entrada del 74LS14N tenemos una tensión de 5V, en cambio, cuando se pulsa dicho pulsador, conecta la masa y en la entrada del 74LS14N tenemos 0V.

Para filtrar los rebotes en la señal debidos a la pulsación, ponemos un condensador de 1μF y un circuito inversor trigger-schmitt.de forma que a la salida dispongamos de una señal filtrada sin rebotes. De esta forma, a la entrada del pin de interrupción, tendremos una señal de 0V cuando el pulsador está en su posición normal, y de 5V cuando lo pulsamos.

4.1.1.10 Alimentación del sistema

Para alimentar nuestro sistema dispondremos de tres fuentes de alimentación:

- Una fuente de alimentación de 24V para la parte digital y de generación del estímulo.

- Dos fuentes de alimentación de 48V y 1A que nos proporcionarán en conjunto ±48V, para alimentar la etapa de potencia.

En las etapas digitales y de señal no necesitamos 24V, sino que necesitamos ±12V para las etapas analógicas, 9V para alimentar la placa del Arduino y 5V para alimentar el resto de circuitos integrados digitales.

Para obtener los ±12V a partir de la alimentación de 24V emplearemos el circuito integrado VAWQ3-Q24-D12, un convertidor DC-DC que con 24V de entrada, genera una salida de ±12V de 3W [23].

Dicho convertidor posee los siguientes pines:

- +VIN1, +VIN2: Pines conectados a la tensión continua de entrada.
- -VIN1, -VIN2, COM1, COM2: Pines conectados a masa.
- +Vo: Tensión de salida de +12V.
- -Vo: Tensión de salida de -12V.

En la Ilustración 20 podemos observar cómo se ha realizado el conexionado del VAWQ3 para suministrar una tensión de ±12V a partir de una única alimentación de 24V.

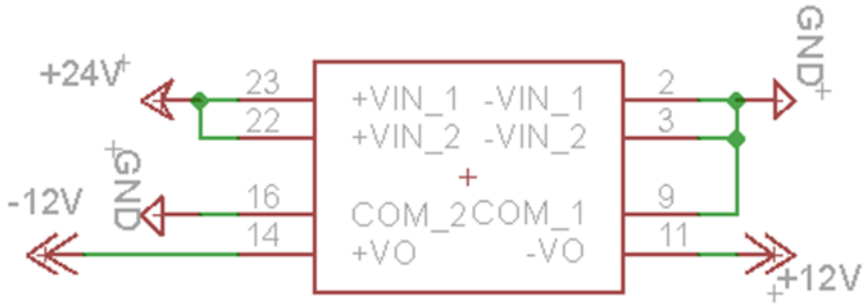

Ilustración 20. Conexionado del VAWQ3.

Como se ha resumido en la Tabla 1 del el apartado 4.1.1.2, la placa del Arduino tiene que alimentarse con una tensión comprendida en el rango de 7-12V. Para ello, emplearemos un regulador lineal LM7809, que a partir de la alimentación de 24V nos generará una tensión de salida de 9V y hasta 1A de corriente. El regulador 7809 tiene los siguientes tres pines, y podemos ver como se han conectado en la Ilustración 21.

- IN: Tensión de entrada de 24V.
- OUT: Tensión de salida regulada a 9V.
- GND: Pin conectado a masa.

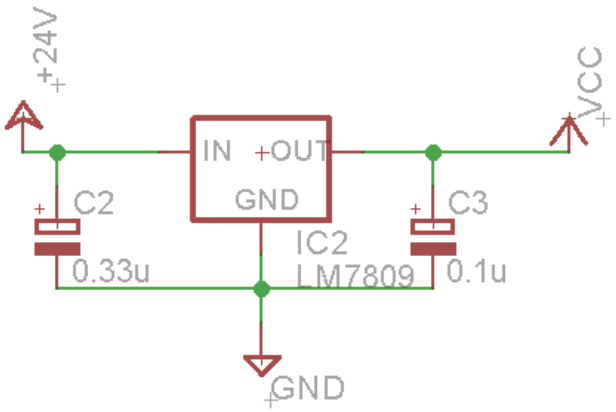

Ilustración 21. Conexionado del LM7809

Ya sólo nos queda la alimentación de 5V para la circuitería digital. Tal y como se explicó en el apartado 4.1.1.2, la placa del Arduino se alimenta con una tensión de 9V, con la que un regulador presente en la placa del Arduino genera una tensión de 5V que alimenta el microcontrolador y los componentes de los que dispone la placa del Arduino. Además, dicha tensión de alimentación va a un pin del Arduino, permitiendo su uso para alimentar toda la circuitería digital externa al Arduino que disponemos en nuestro sistema.

4.1.1.11 Diseño de la PCB.

A continuación describiremos el diseño final de la PCB de nuestro sistema.

Una de las condiciones de nuestro proyecto es que la etapa de potencia estuviese separada de las etapas digitales y de acondicionamiento de la señal, de forma que si había algún fallo o sobretensión en la etapa de potencia, afectase lo menos posible al resto de componentes y se pudiese sustituir con facilidad.

Diseñaremos por tanto dos tipos de PCB para nuestro proyecto, las cuales serán fabricadas mediante el proceso de fresado.

Placa principal (ver Ilustración 22). Esta placa actúa a modo de placa base y contiene los elementos básicos de nuestro sistema, incluyendo los subsistemas de alimentación, de generación y el acondicionamiento de la señal de ambos canales de estimulación y la placa Arduino Dispone de:

- Dos conectores de alimentación, uno para la alimentación de 24V y otro para la alimentación de ±48V.
- Regulador lineal 7809 que a partir de los 24V de alimentación obtiene los 9V para alimentar el Arduino.
- Convertidor DC-DC VAWQ3 que proporciona a partir de la alimentación de 24V obtiene los ±12V para alimentar los circuitos analógicos de acondicionamiento de señal.
- Socket para conectar la placa del Arduino.
- Pulsador para detener la estimulación.
- Inversor Trigger – Schmitt 7414.
- Led para indicar si la estimulación está habilitada.
- Dos convertidores DAC MCP4726, uno para cada canal de estimulación.
- Cuatro amplificadores operacionales LT1097, dos para cada canal.
- Dos multiplexores analógicos ADG408, uno para cada canal.
- Dos conectores hembra de pines cuadrados para poderle conectar la placa de potencia.

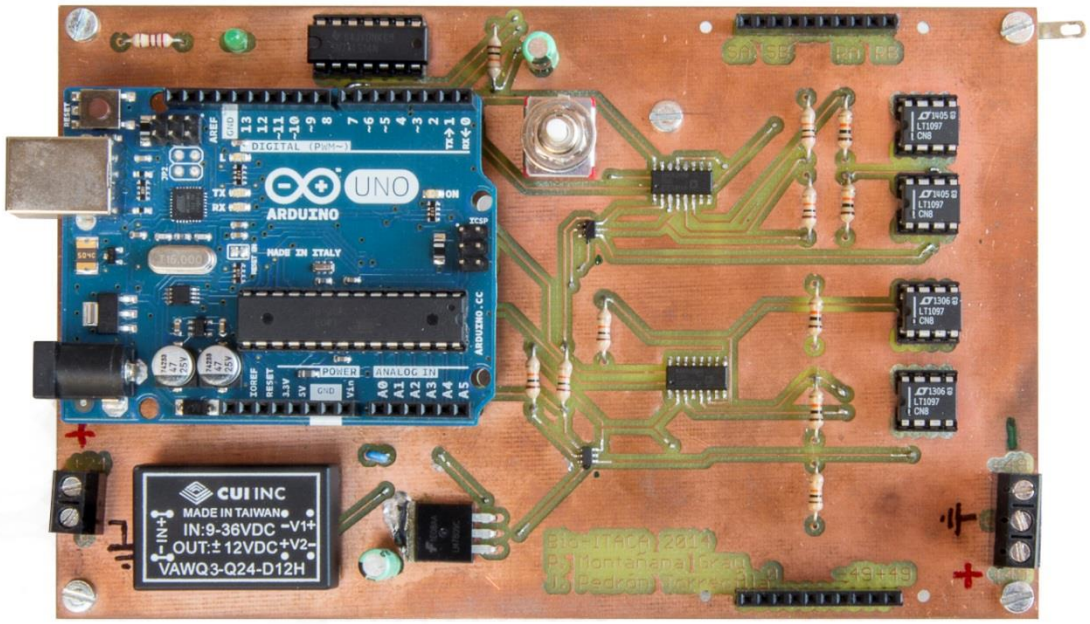

Ilustración 22. Placa principal con la placa del Arduino conectada a ella.

La placa de potencia tiene un menor número de componentes ya que únicamente lleva la circuitería de potencia de uno de los dos canales (ver Ilustración 23). Hemos realizado un diseño modular que permite que una misma placa de potencia pueda servir para el canal 1 o el canal 2 indistintamente, programándola mediante Jumpers para seleccionar un canal u otro.

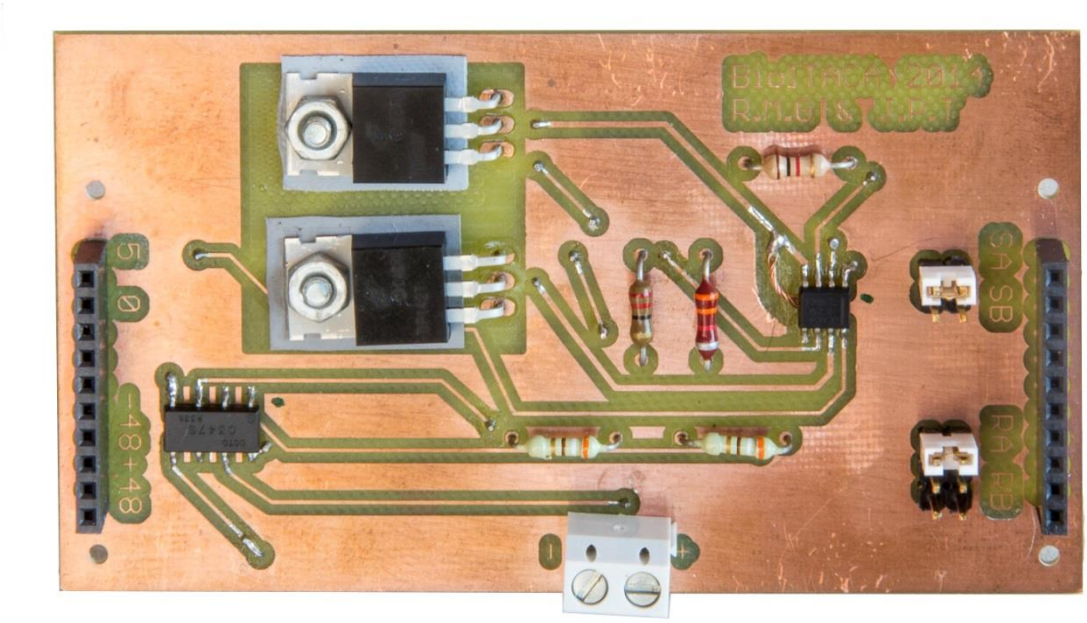

Ilustración 23. Placa de potencia.

Sus componentes son:

- Conectores hembra-macho de pines cuadrados para conectarla a la placa principal y conectar otra placa de potencia sobre esta placa (ver Ilustración 24)
- Jumper de dos posiciones que permite elegir si a la etapa de potencia entra la señal correspondiente al canal 1 o al canal 2.

- Jumper de tres posiciones que permite seleccionar como señal de activación del relé la correspondiente al canal 1, al canal 2 o mantener el relé desactivado.
- Amplificador operacional de alto voltaje OPA454.
- Etapa push-pull con los amplificadores TIP147 y TIP142.
- Relé optoacoplador de dos canales C347S.
- Un conector para conectar la salida del estimulador al corazón.

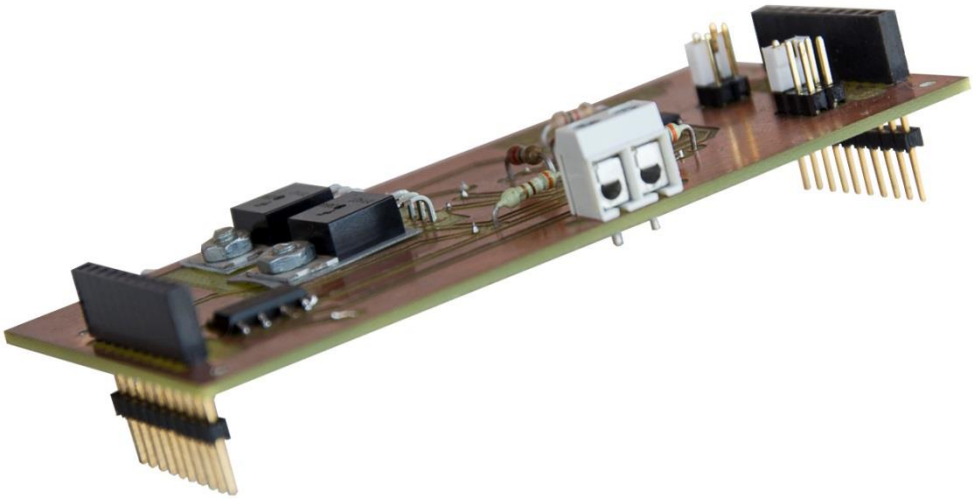

Ilustración 24. Conectores hembra-macho de la placa de potencia.

Así pues, finalmente hemos realizado un diseño que separa la etapa digital y de acondicionamiento de la señal de la etapa de potencia,

Además el diseño que hemos realizado es muy modular, ya que si queremos añadir un nuevo canal, bien sea del A o del B, únicamente hay que añadir una PCB de potencia adicional (ver Ilustración 25 e Ilustración 26); y si queremos cambiar de un canal a otro, solo hay que cambiar la posición de dos jumpers.

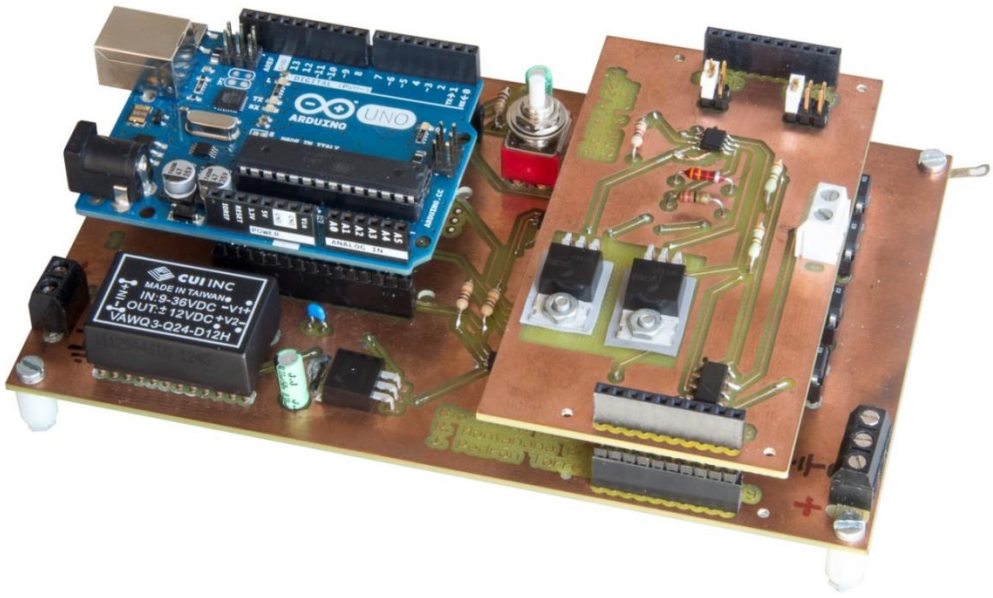

Ilustración 25. Estimulador con únicamente una salida de potencia.

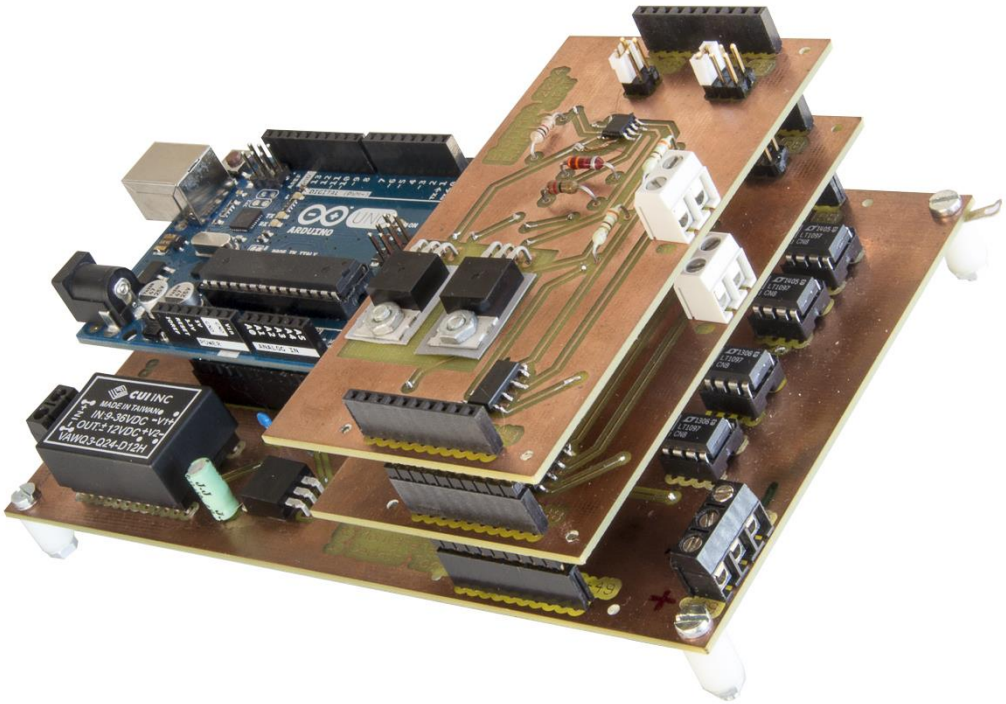

Ilustración 26. Estimulador con dos salidas de potencia.

Comentar también la facilidad de sustitución de la placa de potencia en caso de fallo, ya que cada placa son únicamente cuatro componentes, aparte de las resistencias de realimentación.

4.1.2 Código del firmware del Arduino.

En esta sección describiremos el código del firmware que controla el comportamiento del Arduino y por lo tanto de todo nuestro sistema.

Nuestro estimulador consta de dos canales independientes entre sí, pero cuyos parámetros configurables son los mismos, por ello, para ordenar mejor las variables y que sus nombres sean más comprensibles, agruparemos dichos parámetros en la estructura de la Ilustración 27.

```
typedef struct
{
  unsigned int tension_dac;//Valor que enviamos al DAC
  unsigned long tension_dac_ant;//Valor binario que tenía el DAC en la iteracion anterior
  float periodo;//periodo que queremos, parametro de entrada
  unsigned long ancho;//Ancho de pulso que queremos, parametro de entrada
  unsigned long contador_interrupciones;//Numero de interrupciones que llevamos realizadas
  unsigned long cont_estimulo;//Numero de estimulos del protocolo que ya se han realizado
  char estimular;
  unsigned long periodo_final;//Periodo entre estimulos al final del protocolo
  unsigned long longitud;//Longitud del vector de estimulos
  unsigned long periodos_protocolo[MAX_NUM_PROT];
  char aplicar_protocolo;
}ESTIMULO;
```

Ilustración 27. Definición de la estructura tipo ESTIMULO

De esta manera, únicamente tenemos que declarar dos estructuras del tipo *ESTIMULO* para disponer de dos canales independientes, y en caso de querer más canales, sólo hay que declarar tantas estructuras de este tipo como canales se quieran.

Lo primero que ejecuta el Arduino es la función “setup()” (Ilustración 28), que únicamente se ejecuta en el arranque y realiza la inicialización de las variables y la configuración de los periféricos. Concretamente realiza las siguientes tareas:

- Inicialización de la UART a 9600 baudios.
- Configuración de los DAC del canal 1 y del canal 2.
- Configuración de los pines como entrada o salida.
- Configuración del pin 2 como una interrupción por flanco de subida.
- Inicialización de los pines de salida.
- Configuración de la interrupción de timer para que salte cada 10ms y cuyo handler es la función “est_temporizado()”.

```
void setup()
{
  Serial.begin(9600);
  Wire.begin(); //Nos unimos al bus I2C como master
  //Configuramos el DAC
  configurar_dac(B1100000);//configurar DAC 1
  configurar_dac(B1100001);//configurar DAC 2

  init_variables(&canal1);
  init_variables(&canal2);
  //Configuramos los pines
  pinMode(ledpin,OUTPUT);
  pinMode(RELE_A,OUTPUT);
  pinMode(RELE_B,OUTPUT);
  pinMode(MULT_A0,OUTPUT);
  pinMode(MULT_A1,OUTPUT);
  pinMode(MULT_B0,OUTPUT);
  pinMode(MULT_B1,OUTPUT);
  //Configuramos las interrupciones
  //Interrupcion de pausa del estimulador asociada al flanco de subida del PIN 2
  attachInterrupt(0,pausa,RISING);
  //Estado inicial de los pines
  digitalWrite(ledpin,LOW); //LED indicador de pausa, encendido
  //Inicialmente el multiplexor A saca S1 (masa)
  digitalWrite(MULT_A0,LOW);
  digitalWrite(MULT_A1,LOW);
  digitalWrite(MULT_B0,LOW);
  digitalWrite(MULT_B1,LOW);
  //Configuramos la interrupcion para el estimulo temporizado
  MsTimer2::set(10,est_temporizado);
}
```

Ilustración 28. Función setup (Inicialización y configuración de los periféricos).

A continuación se ejecutará permanentemente la función “loop()”. Lo primero que se hace en esta función es guardar las tensiones que ha sacado el DAC en la iteración anterior, de forma que si no se cambia en esta iteración, no será necesario reenviar el comando de cambiar tensión al DAC (Ilustración 29).

```
void loop()
{
  //Almacenamos el valor enviado al DAC en la iteracion anterior
  canal1.tension_dac_ant=canal1.tension_dac;
  canal2.tension_dac_ant=canal2.tension_dac;
```

Ilustración 29. Almacenamos la tensión almacenada en la iteración anterior.

La función está continuamente escuchando el puerto serie a la espera de recibir un nuevo comando desde el PC. Dicho comando provocará un cambio en la tensión, el ancho, el periodo del pulso o cargará un nuevo protocolo de estimulación; en cualquiera de los dos canales, canal 1 o canal 2 (Ilustración 30).

```
String comando= "";
//Si hay datos en la UART los recibimos y los tratamos
if (Serial.available()>0)
{
  //Concatenamos todas los caracteres que nos llegan para formar el comando
  while(Serial.available()>0)
  {
    comando.concat(char(Serial.read()));
    delay(5);//Sin este delay no concatena los caracteres
  }
  Serial.println(comando);
  if (comando.substring(0,4)=="EST1") //Modificamos el canal 1
  {
    if (comando.substring(4,7)=="TEN") //Cambiamos la tension
    {
      //Decodificamos el valor de la nueva tension
      word tension=(comando.substring(7)).toInt();
      calc_tension(&canal1, tension);
    }
    if (comando.substring(4,7)=="PER") //Cambiamos el periodo
    {
      //Decodificamos el periodo en decimas de segundo
      float periodo=(comando.substring(7)).toInt();
      calc_periodo(&canal1, periodo);
      calc_num_interrupciones(&canal1);
    }
    if (comando.substring(4,7)=="ANC") //Cambiamos el ancho de pulso
    {
      canal1.ancho=(comando.substring(7)).toInt();//Decodificamos el ancho del pulso
    }
    if (comando.substring(4,7)=="PRO") //Cambiamos el protocolo de estimulacion
    {
      carga_protocolo(&canal1);
      calc_num_interrupciones(&canal1);
    }
  }

  if (comando.substring(0,4)=="EST2") //Modificamos el canal 2
  {
    if (comando.substring(4,7)=="TEN") //Cambiamos la tension
    {
      //Decodificamos el valor de la nueva tension
      word tension=(comando.substring(7)).toInt();
      calc_tension(&canal2, tension);
    }
    if (comando.substring(4,7)=="PER") //Cambiamos el periodo
    {
      //Decodificamos el periodo en decimas de segundo
      float periodo=(comando.substring(7)).toInt();
      calc_periodo(&canal2, periodo);
      calc_num_interrupciones(&canal2);
    }
  }
}
```

Ilustración 30. Cambio de los parámetros de estímulo a través de comandos enviados desde el PC

Por último la función "loop()" hará una llamada a la función encargada de cambiar en caso necesario la tensión del DAC. Luego comprobará si el estimulador está pausado o activo, en caso de que esté activo, y haya pasado el periodo de estímulo, se llamará a la función encargada de realizar dicho estímulo (Ilustración 31).

```
  //Llamamos a la funcion para cambiar las tensiones de salida de los DAC
  cambiar_tension_DAC(&canal1, B11000000);
  cambiar_tension_DAC(&canal2, B11000001);

  //Comprobamos si hay que realizar un estimulo
  if(encendido==true)
  {
    if(canal1.estimular==true)
    {
      //Ejecutamos el estimulo
      estimulo(1);
      canal1.estimular=false;
    }
    if(canal2.estimular==true)
    {
      estimulo(2);
      canal2.estimular=false;
    }
  }
}
```

Ilustración 31. Cambio de las tensiones de los DAC y realización del estímulo en su caso.

Una vez explicada la función "loop()", pasaremos a explicar la función "init_variables()". Esta función tiene como parámetro de entrada una estructura de tipo ESTIMULO e inicializa todas las variables de dicha estructura (Ilustración 32).

```
void init_variables(ESTIMULO *canal)
{
  canal->tension_dac=0;
  canal->tension_dac_ant=0;
  canal->periodo=0;
  canal->ancho=0;
  canal->contador_interrupciones=0;
  canal->cont_estimulo=1;
  canal->estimular=false;
  canal->periodo_final=0;
  canal->longitud=0;
  canal->aplicar_protocolo=false;
}
```

Ilustración 32. Función init_variables

Cuando la función "loop()" recibe un comando de cambiar la tensión del estímulo, llama a la función "calc_tension()" (Ilustración 33). Dicha función tiene dos parámetros de entrada: una estructura ESTIMULO correspondiente al canal del que se quiere cambiar su tensión de estímulo, y empleando la ecuación (4.1) calcula el valor binario que se le tiene que enviar al DAC para obtener dicha tensión a la salida del estimulador a partir de la otra entrada "tension".

```
//Funcion para calcular el valor binario de la tension DAC para cada canal
void calc_tension(ESTIMULO *canal, unsigned long tension)
{
  // Calculamos el valor a enviar al DAC, redondeado con el +0.5
  canal->tension_dac=(tension*102.375)+0.5;
}
```

Ilustración 33. Funcion calc_tension

Disponemos de una función llamada “cambiar_tension_DAC()” que nos permite cambiar la tensión de salida de los dos DAC que dispone la placa. A dicha función le pasamos como parámetros el canal del que deseamos cambiar la tensión, y la dirección I^2C del correspondiente DAC.

La función comprueba si la tensión de dicho canal ha cambiado respecto a la iteración anterior, y en caso afirmativo, genera el comando para cambiar la tensión de salida del DAC, y envía la trama I^2C (Ilustración 34).

```
//Funcion para cambiar la tension de salida del DAC
void cambiar_tension_DAC(ESTIMULO *canal, uint8_t direccion)
{
  if (canal->tension_dac_ant != canal->tension_dac)
  {
    // Generamos la trama para escribir en el registro DAC
    word trama_tension = 0x0000;
    trama_tension = trama_tension | canal->tension_dac;

    //Generamos los dos bytes a enviar a partir de la trama
    //Los dos bytes que tenemos que enviar al DAC para cambiar el registro DAC
    byte bytes_tension[2];
    //Nos quedamos con el byte alto de la palabra
    bytes_tension[0] = (trama_tension >> 8) & 0xFF;
   //Nos quedamos con el byte bajo de la palabra
    bytes_tension[1] = trama_tension & 0xFF;

    //Enviamos por I2C
    Wire.beginTransmission(direccion);
    Wire.write(bytes_tension,2);
    Wire.endTransmission();
  }
}
```

Ilustración 34. Función cambiar_tension_DAC

Cuando el comando que se recibe es el de cambiar el periodo entre estímulos se ejecuta la función “calc_periodo()”.

La función “calc periodo()” únicamente convierte el periodo que hemos transmitido a través del comando de milisegundos a segundos, y deshabilita el estímulo por protocolo (Ilustración 35).

```
//Funcion para calcular el periodo
void calc_periodo(ESTIMULO *canal, float periodo)
{
  //Calculamos el periodo en segundos
  canal->periodo=periodo*0.01;
  //Al cambiar el periodo de estimulo, desactivamos el estimulo por protocolo
  canal->aplicar_protocolo=false;
}
```

Ilustración 35. Función calc_periodo

Cuando se envía el comando indicando que se desea estimular a través de un protocolo de estimulación, se ejecuta la función “carga_protocolo()” (Ilustración 36). Esta función tiene como parámetro de entrada la estructura con las variables asociadas al canal que se desea modificar. La función reinicia el contador de estímulos del protocolo de estimulación que ya se han realizado y a continuación obtiene los siguientes parámetros necesarios para la estimulación por protocolo a través de la UART:

- Periodo que se aplicará una vez acabe el protocolo de estimulación.
- Longitud del protocolo de estimulación.
- Cada uno de los diferentes tiempos entre los estímulos que forman parte del protocolo y que guardaremos en el vector "periodos_protocolo()"
- La amplitud y el ancho del pulso se configuran aparte a través de sus propios comandos.

Por último activa el estímulo por protocolo.

```
//Funcion para cargar un protocolo
void carga_protocolo(ESTIMULO *canal)
{
  byte periodo_final_bajo;
  byte periodo_final_alto;
  byte longitud_bajo;
  byte longitud_alto;
  //Reiniciamos el contador de estimulos del protocolo realizados
  canal->cont_estimulo=1;
  //Espero a tener nuevos datos en la UART
  while(Serial.available()==0)
  {
  }
  while(Serial.available()>0)
  {
    periodo_final_bajo=Serial.read();
    while(!Serial.available());
    periodo_final_alto=Serial.read();
    //Periodo una vez acabe el protocolo
    canal->periodo_final=(periodo_final_alto<<8) | periodo_final_bajo;
    while(!Serial.available());
    longitud_bajo=Serial.read();
    while(!Serial.available());
    longitud_alto=Serial.read();
    //Longitud del protocolo
    canal->longitud=(longitud_alto<<8) | longitud_bajo;
    byte periodo_protocolo_bajo;
    byte periodo_protocolo_alto;
    int i;
    //Guardamos los distintos periodos del protocolo
    for(i=1;i<=canal->longitud;i++)
    {
      while(!Serial.available());
      periodo_protocolo_bajo=Serial.read();
      while(!Serial.available());
      periodo_protocolo_alto=Serial.read();
      canal->periodos_protocolo[i]=(periodo_protocolo_alto<<8) | periodo_protocolo_bajo;
    }
    //Al seleccionar un protocolo de estimulo, activamos el estimulo por protocolo
    canal->aplicar_protocolo=true;
  }
}
```

Ilustración 36. Función carga_protocolo

Tanto si hacemos un estímulo con un periodo fijo o un estímulo por protocolo, necesitaremos calcular el número de interrupciones del timer que tienen que ocurrir para que el estímulo se realice con el periodo deseado.

Para eso, empleamos la función "calc_num_interrupciones()" a la que le pasamos como parámetro de entrada la estructura del canal que queremos modificar (Ilustración 37). Esta función tiene dos comportamientos diferentes:

- Si estamos trabajando con un periodo fijo, la función calcula cuantas interrupciones de 10ms tienen que ocurrir y lo guardamos en la primera posición del vector "periodos_protocolo()".
- Si estamos trabajando con una estimulación por protocolo, calcula el número de interrupciones que deben ocurrir para cada uno de los periodos almacenados en el vector "periodos_protocolo" y lo almacena en la correspondiente posición del vector.

Una vez ha calculado el número de interrupciones habilita el timer y la interrupción.

```
//Funcion para calcular el numero de estimulos
void calc_num_interrupciones(ESTIMULO *canal)
{
  float periodo;
  if(canal->aplicar_protocolo==false)
  {
    canal->periodos_protocolo[0]=word(canal->periodo/0.01);
  }
  else //Calculamos el numero de interrupciones necesarias para cada periodo del estimulo
  {
    canal->periodo_final=canal->periodo_final*0.001;
    canal->periodos_protocolo[0]=word(canal->periodo_final/0.01);
    for (int i=1; i<=canal->longitud; i++)
    {
      periodo=canal->periodos_protocolo[i]*0.001;//Calculamos el periodo en segundos
      canal->periodos_protocolo[i]=word(periodo/0.01);
    }
  }
  MsTimer2::start();
}
```

Ilustración 37. Función calc_num_interrupciones.

La interrupción del timer saltará cada 10ms y es atendida por la función "est_temporizado()" (Ilustración 38), la cual llama a la función "comprobar_temporización()" (

Ilustración 39).

```
//Handler de la interrupcion del contador
void est_temporizado()
{
  comprobar_temporizacion(&canal1);
  comprobar_temporizacion(&canal2);
}
```

Ilustración 38. Funcion est_temporizado.

La función "comprobar_temporizacion()" también tiene dos comportamientos diferentes:

- En el caso de que tengamos configurado un periodo fijo, añadiremos uno al contador de cuantas interrupciones han ocurrido. En el momento en que llegamos al número de interrupciones necesarias, se reinicia el contador y se habilita el flag de que se realice un estímulo.
- En el caso de un estímulo por protocolo, se aumentará en uno el contador de cuantas interrupciones se han realizado. Cuando llegamos al número de interrupciones necesarias para un estímulo determinado del protocolo, se realizará un estímulo y se aumentará en uno el contador del número de estímulos que se han realizado, con lo que cambiamos la temporización para el siguiente estímulo a realizar; reiniciamos el

contador del número de interrupciones y habilitamos el estímulo. Cuando se han realizado todos los estímulos del protocolo, reiniciamos el contador de estímulos que se han realizado y deshabilitamos el estímulo del protocolo, de forma que pasamos a estimular con un periodo fijo.

```
void comprobar_temporizacion(ESTIMULO *canal)
{
  if(canal->periodos_protocolo[0]==0)//No hay periodo de estimulo
  {
  }
  else
  {
    canal->contador_interrupciones++;//Hemos hecho una interrupcion
    if(!canal->aplicar_protocolo)
    {
      //Hemos superado o igualado el numero de interrupciones requeridas
      if(canal->contador_interrupciones >= canal->periodos_protocolo[0])
      {
        canal->estimular=true;
        canal->contador_interrupciones=0;//Reiniciamos el contador de interrupciones
      }
    }
    else
    { //Se realizan los estimulos del protocolo, y al final se fija un periodo de protocolo
      if(canal->contador_interrupciones >= canal->periodos_protocolo[canal->cont_estimulo])
      {
        canal->cont_estimulo++;
        canal->estimular=true;
        canal->contador_interrupciones=0;
        if(canal->cont_estimulo>canal->longitud)
        {
          canal->cont_estimulo=1;
          canal->aplicar_protocolo=false;
        }
      }
    }
  }
}
```

Ilustración 39. Función comprobar_temporizacion.

La función “estimulo()” es la que se encarga de generar el estímulo. Lo primero que realiza es activar el relé de salida, a continuación modifica las señales de control del multiplexor analógico para que tengamos una salida de +10V. Después del tiempo marcado por la variable “ancho”, modificamos el control del multiplexor de forma que obtenemos a la salida -10V. A continuación conmutamos el multiplexor para que saque masa y desactivamos los relés para aislar los electrodos del estimulador (Ilustración 40).

```
//Funcion estimulo
void estimulo(byte num_canal)
{
  byte i;
  //Activamos los reles seleccionados
  switch (num_canal)
  {
    case 1:
      digitalWrite(RELE_A, HIGH);
      break;
    case 2:
      digitalWrite(RELE_B, HIGH);
      break;
  }
  if(num_canal==1)
  {
    //Sacamos por el multiplexor A S3 (Señal positiva)
    //digitalWrite(ledpin,HIGH);//BORRAR
    digitalWrite(MULT_A1,LOW);
    digitalWrite(MULT_A0,HIGH);
    //Esperamos el ancho de pulso
    delay(canal1.ancho);
    //digitalWrite(ledpin,LOW);//BORRAR
    //Sacamos por el multiplexor A S2 (Señal negativa);
    digitalWrite(MULT_A1,HIGH);
    digitalWrite(MULT_A0,LOW);
    //Esperamos el ancho de pulso
    delay(canal1.ancho);
    //Sacamos masa por el multiplexor A
    digitalWrite(MULT_A1,LOW);
  }
  else if(num_canal==2)
  {
    digitalWrite(MULT_B1,LOW);
    digitalWrite(MULT_B0,HIGH);
    //Esperamos el ancho de pulso
    delay(canal2.ancho);
    //Sacamos por el multiplexor A S2 (Señal negativa);
    digitalWrite(MULT_B1,HIGH);
    digitalWrite(MULT_B0,LOW);
    //Esperamos el ancho de pulso
    delay(canal2.ancho);
    //Sacamos masa por el multiplexor A
    digitalWrite(MULT_B1,LOW);
  }
  //Desasctivamos todos los reles
  switch (num_canal)
  {
    case 1:
      digitalWrite(RELE_A, LOW);
      break;
    case 2:
      digitalWrite(RELE_B, LOW);
      break;
  }
}
```

Ilustración 40. Función estimulo

La función que atiende a la interrupción del pin 2 es la función “pausa()”. Cuando se pulsa el pulsador, entramos en esta función, si el estimulador está activado, se pausará y viceversa. La función controla también un LED que nos informa de que el estímulo está activado al estar encendido (Ilustración 41).

```
//Handler de la interrupcion pausa
void pausa()
{
  if (encendido==true)//El estimulador estaba en marcha cuando se ejecuta la interrupcion
  {
    encendido=false;//Indicamos que el estimulador pasa a estar en pausa
    digitalWrite(ledpin,HIGH);//Apagamos el led de estado
  }
  else if(encendido==false)//El estimulador estaba en pausa cuando se ejecuta la interrupcion
  {
    encendido=true;//Indicamos que el estimulador pasa a estar encendido
    digitalWrite(ledpin,LOW);//Encendemos el led de estado
  }
}
```

Ilustración 41. Función pausa

4.1.3 Desarrollo de la interfaz de usuario.

Para configurar los parámetros del estimulador, se realizó un programa para Windows que sirva como una interfaz de usuario que permita comunicarse con el Arduino y modificar su configuración.

Se realizó el programa en el lenguaje de programación C#, el cual es un lenguaje orientado a objetos que se encuentra en el framework .NET de Microsoft.

.NET es un componente de software para el sistema operativo Windows. La versión más reciente que es la 4.5.1 viene incluida en Windows 8.1 y Windows Server 2012 R2 [24].

El framework permite la programación en varios lenguajes entre los que se encuentran Visual Basic.NET, C++ y el ya mencionado C#, ya que la herramienta de desarrollo en realidad hace una compilación del código escrito por el programador a un código intermedio, llamado CIL (Common Intermediate Language), y posteriormente el framework dispone de un compilador just-in-time, llamado CLR (Common Language Runtime), que genera el código máquina, que es el que se ejecuta realmente en el ordenador. Al conjunto CIL+CLR se le llama CLI (Common Language Infrastructure) [25]. (Ver Ilustración 42).

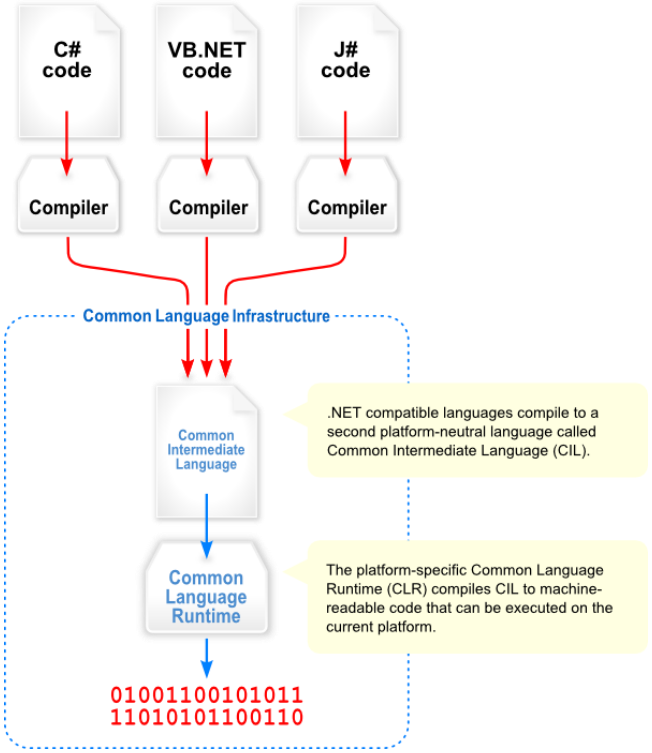

Ilustración 42. Compilación y ejecución de un programa .NET

El CLI del framework .NET está estandarizado por ECMA e ISO, por lo que otros fabricantes, pueden realizar sus propias implementaciones de .NET aparte de la de Microsoft y ser reescritas para otras plataformas distintas a Windows. [26].

Algunas de las implementaciones alternativas del framework son las siguientes:

- .NET Micro Framework: Es un framework .NET para dispositivos con escasos recursos y orientado a los sistemas embebidos. Incluye una versión reducida del CLR y permite la programación en C# y hacer debugging por emulación software o por hardware [27].
- Mono: Es una implementación que incluye compiladores para C# y Visual Basic.NET, además de soporte para ASP.NET, ADO.NET y Windows Forms para una gran variedad de arquitecturas y sistemas operativos, entre ellos, Windows, Linux, y OsX [28].

Se ha empleado la API Windows Forms, la cual da acceso a los elementos gráficos de la API de Windows, para realizar el entorno gráfico.

Nuestro programa dispone de una pantalla principal donde configurar los parámetros de la estimulación.

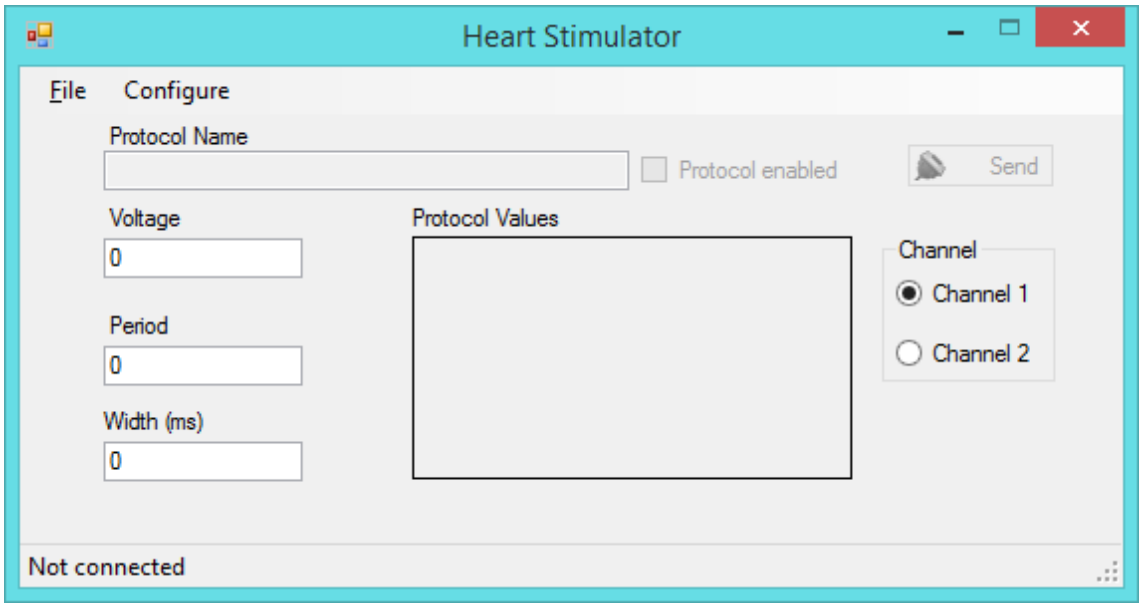

Ilustración 43. Pantalla principal del interfaz de usuario.

Como podemos ver en la Ilustración 43, la pantalla principal nos permite configurar la amplitud del estímulo, el periodo entre estímulos y el ancho del pulso que deseamos.

En el caso de que para el canal que tenemos seleccionado hayamos cargado un protocolo de estimulación, el nombre de dicho protocolo aparecerá en el recuadro “Protocol Name”, la casilla “Protocol enabled” pasará a habilitarse y en “Protocol Values” aparecerá una tabla con los valores de dicho protocolo. Pulsando en “Protocol enabled”, el usuario podrá cambiar entre un estímulo por periodo fijo o por protocolo de estimulación en dicho canal.

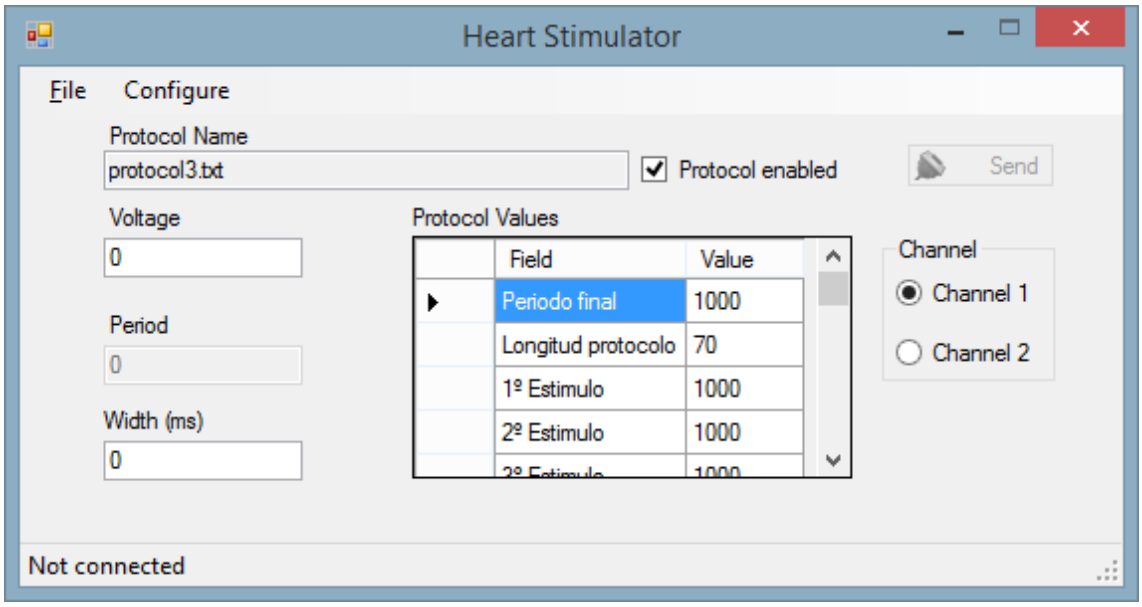

Ilustración 44. Canal con protocolo de estimulación cargado

Podemos ver además en la Ilustración 44 que si la casilla "Protocol enabled" está activada, se deshabilita la casilla para elegir el periodo fijo, ya que el periodo vendrá dado por los valores del protocolo.

Para cargar un protocolo se ha habilitado un botón dentro del menú "File" (Ilustración 45).

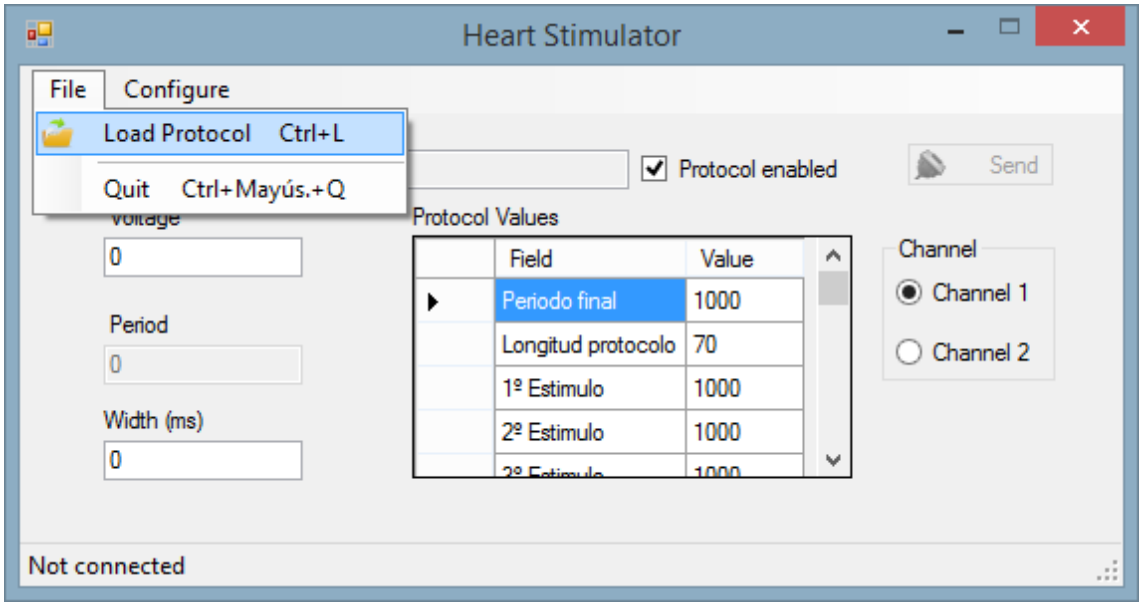

Ilustración 45. Botón para cargar un protocolo de estimulación.

Al pulsar sobre dicho botón, saldrá un dialogo para abrir un fichero, donde se podrá elegir el fichero de texto que contiene el protocolo que deseamos.

Para configurar el puerto serie con el cual deseamos comunicarnos con el Arduino, deberemos entrar al menú Configure y pulsar en Serial Port (ver Ilustración 46), al hacerlo, nos aparecerá una nueva ventana donde podremos elegir entre todos los puertos COM que están disponibles en nuestro PC, tal y como podemos ver en la Ilustración 47.

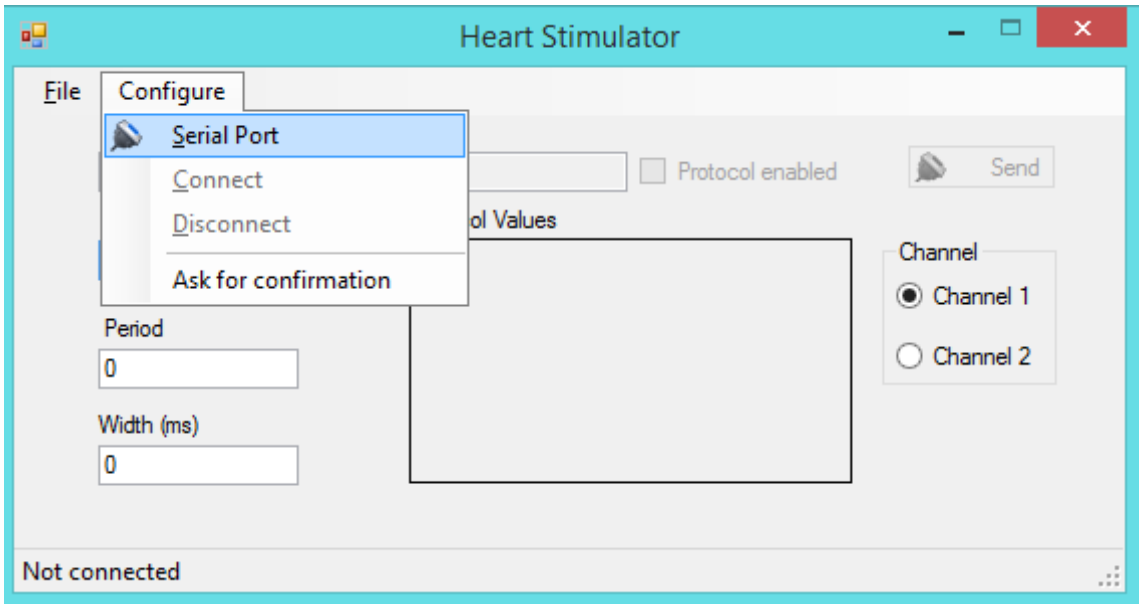

Ilustración 46. Menú para configurar el puerto serie.

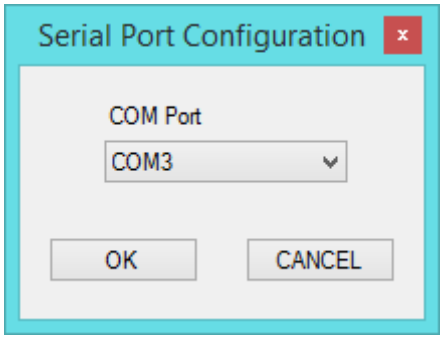

Ilustración 47. Configuración del puerto serie.

Una vez configurado el puerto serie, podemos abrir la conexión, para ello se ha habilitado un botón en el menú Configure (Ilustración 46). Una vez establecida la conexión el botón "Send" se habilitará y ya se podrán enviar las nuevas configuraciones al Arduino pulsando dicho botón.

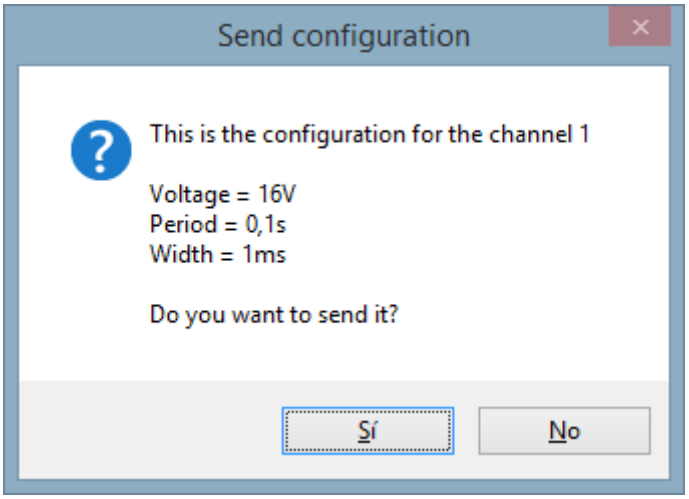

Ilustración 48. Solicitud de la confirmación de la configuración del estímulo

Hay una opción en el menú de configuración que permite que cuando se le pulse al botón "Send", aparezca una ventana de confirmación informando de la configuración del estímulo que se va a enviar al Arduino y permite confirmar o cancelar la configuración (Ilustración 48). Para

ello, únicamente hay que pulsar en Configure → Ask for confirmation y quedará habilitada la opción, para deshabilitarla solo hay que volver a pulsar (Ilustración 49).

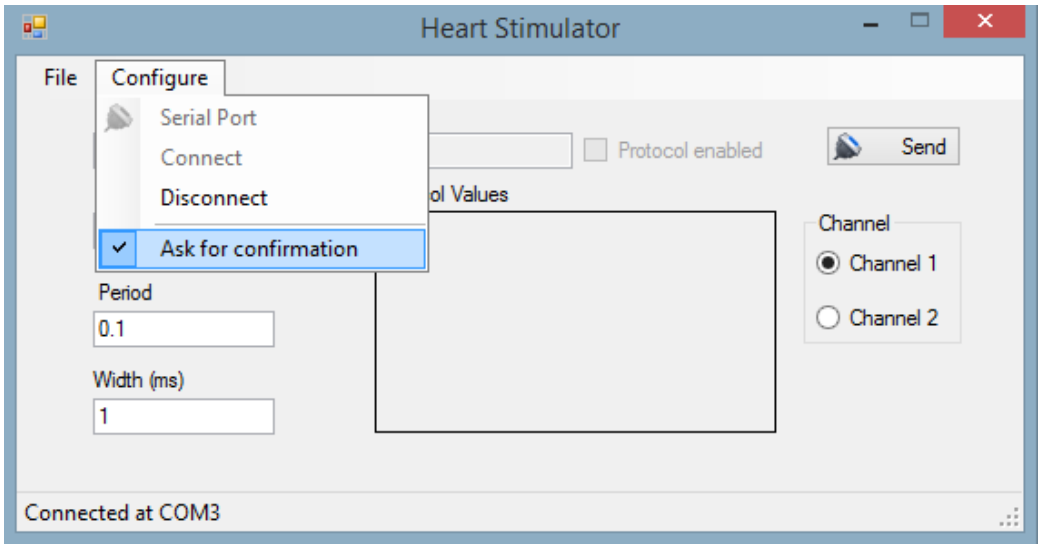

Ilustración 49. Configurar confirmación de la configuración del estímulo.

4.2 Resultados.

En esta sección se pondrán y se analizarán distintas pruebas que se han realizado con nuestro estimulador para comprobar que cumple con las condiciones de diseño estipuladas.

En primer lugar se pondrán casos de estímulo con un único canal y se comprobará que se pueden cambiar los tres parámetros de estímulo, y que se llega como mínimo a los valores impuestos por el pliego de condiciones.

En segundo lugar se repitieron dichas pruebas con los dos canales activados, y se comprobó que ambos canales son completamente independientes y que se pueden emplear simultáneamente.

En tercer lugar se comprobó que el estimulador permite la aplicación al miocardio de los protocolos de estimulación S1-S2 y de frecuencias crecientes, comprobando que el tener los dos canales habilitados no influye en su correcta estimulación.

Por último, se tuvo el estimulador una hora estimulando para realizar un test de persistencia de la temporización y ver que no tiene derivas con el tiempo.

Estas pruebas se hicieron bajo condiciones límites de la placa, es decir, o bien aplicando la amplitud máxima de estímulo, 40V, o bien aplicando la corriente máxima que tiene que soportar el estimulador cardiaco, es decir 1A.

Se testeó la condición de corriente máxima hasta con un régimen de trabajo del 10%, más concretamente, con un ancho de pulso de 20ms y un periodo de 200ms, durante una hora, comprobando que las pistas y los componentes soportan dicha condición límite

Una prueba adicional fue comprobar que sucedía si la salida del estimulador cardiaco se cortocircuitaba a masa, situación probable en la aplicación real, comprobando que las fuentes de alimentación limitaban la corriente, y ni ellas ni ninguno de los componentes de la placa resultaron dañados.

4.2.1 Pruebas con un canal.

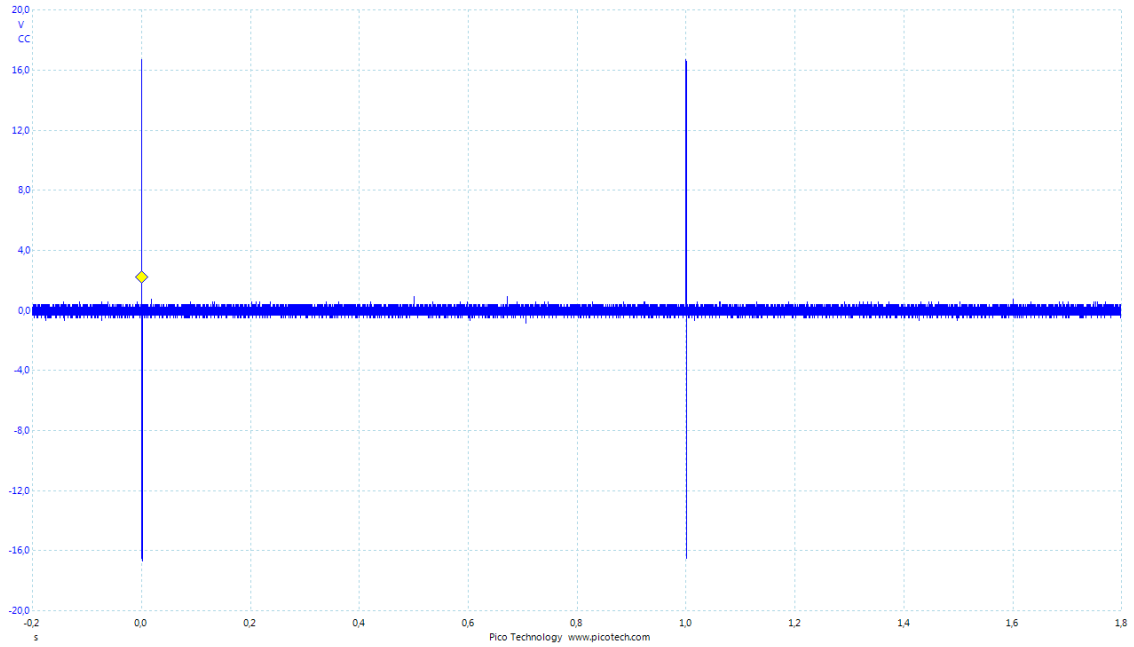

Ilustración 50. Estímulo de un único canal con 16V de amplitud, 2ms de ancho y 1s de periodo entre estímulos.

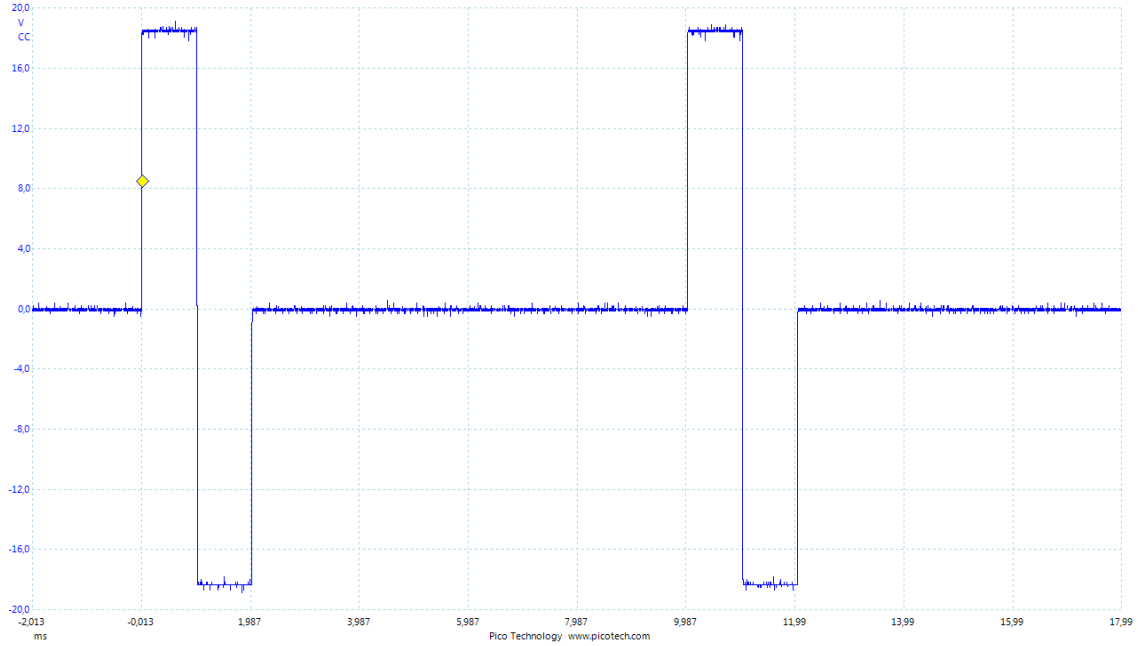

Ilustración 51. Estímulo de un único canal con 18V de amplitud, 2ms de ancho y 10ms de periodo entre estímulos.

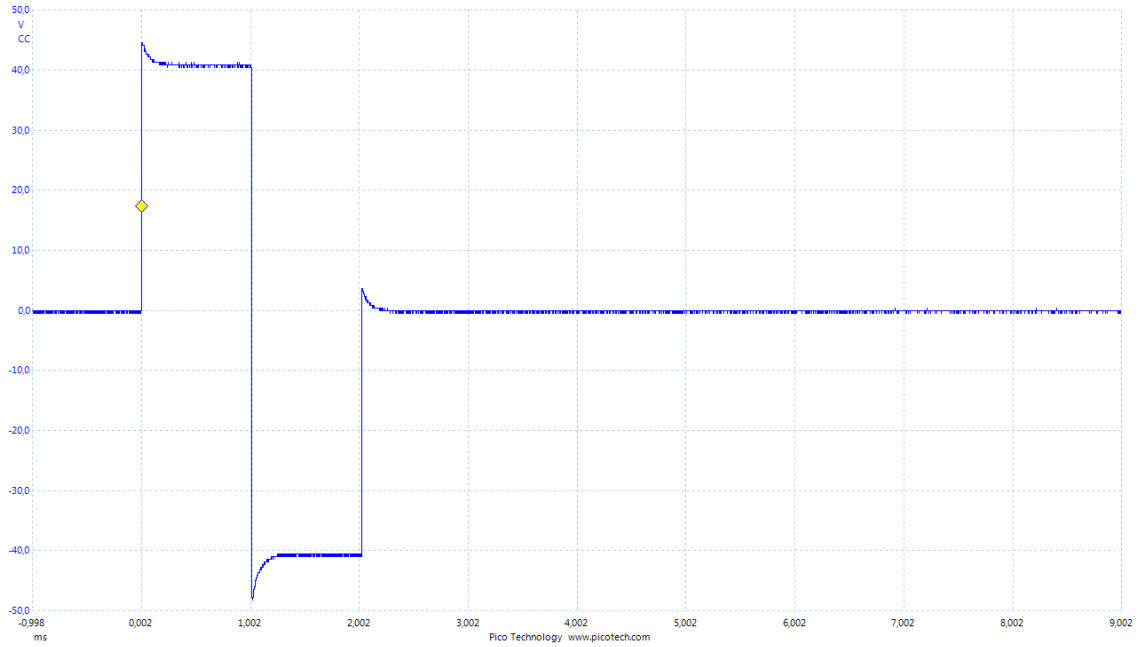

Ilustración 52. Estímulo de un canal con una amplitud de 40V, 2ms de ancho de pulso y un periodo de 1s entre estímulos. Los picos observados en los cambios de tensión se deben a la capacidad que introduce la sonda atenuadora.

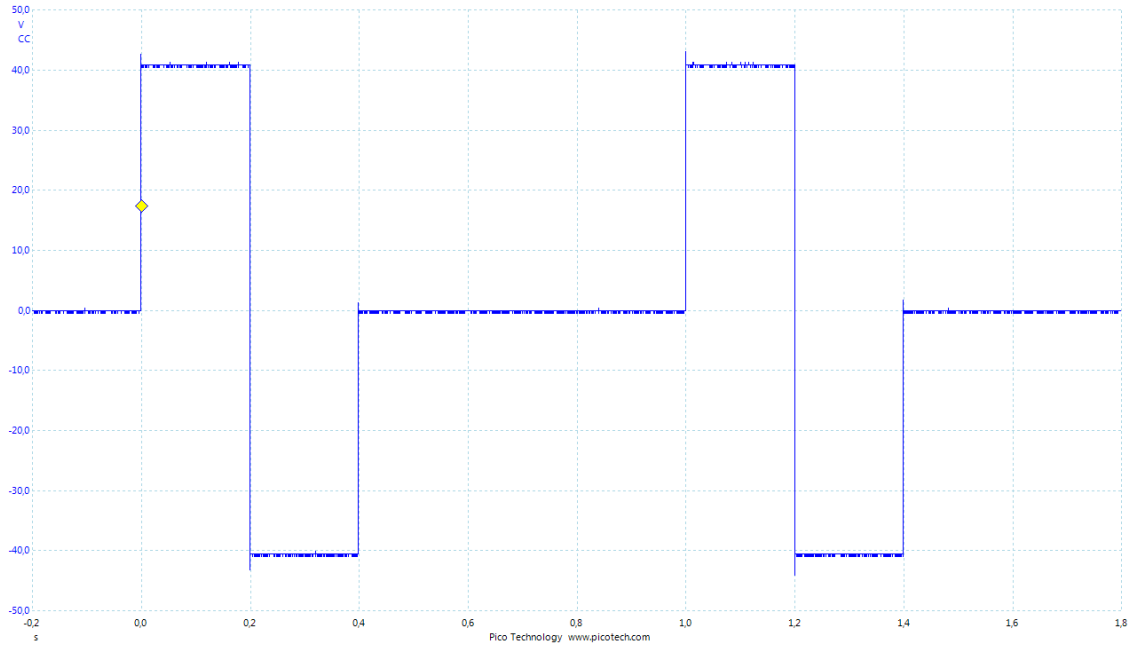

Ilustración 53. Estímulo de un canal con una amplitud de 40V, un ancho de pulso de 400ms y un periodo entre estímulos de 1s. Los picos que se observan en los cambios de tensión se deben a la capacidad que introduce la sonda atenuadora.

En las imágenes anteriores hemos podido comprobar diferentes casos de estímulo que se han aplicado con nuestro estimulador.

En la Ilustración 50 podemos observar el periodo mínimo entre estímulos que permite nuestro sistema, el cual es 10ms. Por otra parte, en la Ilustración 52 y en la Ilustración 53 podemos observar dos estímulos con la amplitud máxima que se exigía en las condiciones que son 40V. En el último caso podemos comprobar también como el ancho del pulso puede ser del orden de milisegundos o del orden del periodo del estímulo.

4.2.2 Pruebas con dos canales habilitados.

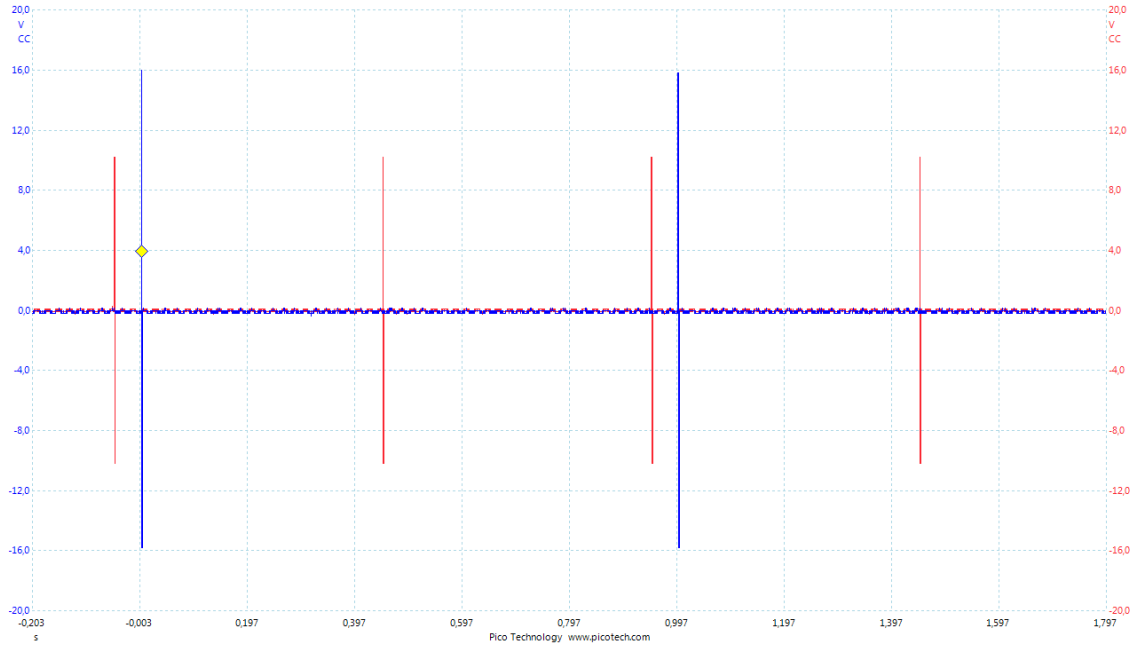

Ilustración 54. Estímulo con dos canales habilitados. El canal azul está configurado a 16V de amplitud con 1s de periodo entre estímulos y 2ms de ancho de pulso. Por otro lado, el canal rojo está configurado a 10V de amplitud, con un periodo de 0.5s entre estímulos y 10ms de ancho de pulso.

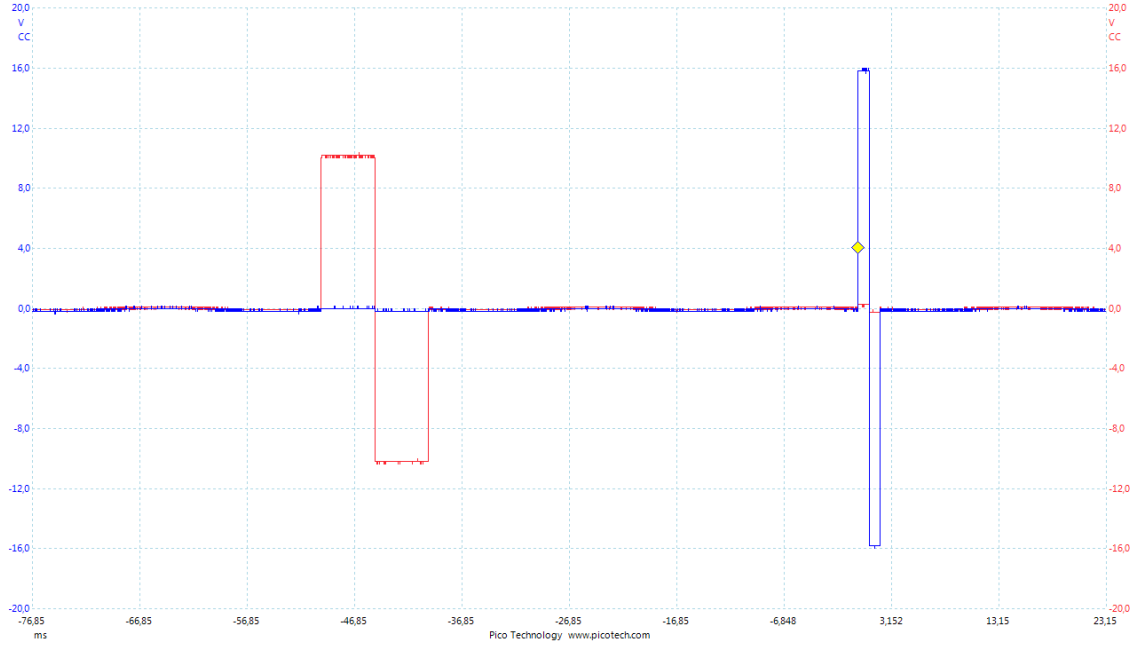

Ilustración 55. Estímulo con dos canales habilitados. El canal azul tiene configurada una amplitud de estímulo de 16V con 2ms de ancho de pulso, mientras el canal rojo tiene configurada una amplitud de estímulo de 10V con un ancho de pulso de 10ms.

La Ilustración 54 nos muestra como nuestro estimulador es capaz de tener dos estímulos independientes con diferentes amplitudes y frecuencias, pero debido a la limitación de la resolución del osciloscopio no es posible apreciar los diferentes anchos de pulso de los dos canales. Por ello, en la Ilustración 54 se ha aumentado la resolución temporal del osciloscopio, para que, a pesar de no poder apreciar el periodo entre estímulos, podamos apreciar los anchos de pulso.

4.2.3 Aplicación de protocolos de estimulación.

En este apartado aplicaremos dos tipos de protocolos de estimulación para comprobar que nuestro estimulador cumple con esa condición de diseño.

En primer lugar realizaremos un protocolo de frecuencias crecientes, en el cual se aumenta la frecuencia de estimulación progresivamente de un estímulo al siguiente (ver Ilustración 56).

Ilustración 56. Protocolo de estimulación de frecuencias crecientes.

Para probarlo en nuestro sistema se implementará un protocolo de frecuencias crecientes que permita ver de forma clara el aumento de la frecuencia entre estímulos. Una vez se empiece a aplicar el estímulo, realizaremos únicamente 5 estímulos y por lo tanto tendremos cuatro periodos de estimulación distintos, que serán:

Tabla 2. Periodos del protocolo de estimulación de frecuencias crecientes.

1er periodo	500ms
2º periodo	250ms
3er periodo	120ms
4º periodo	60ms

Los resultados obtenidos al aplicar dicho protocolo de estimulación con nuestro estimulador se puede observar en la Ilustración 57.

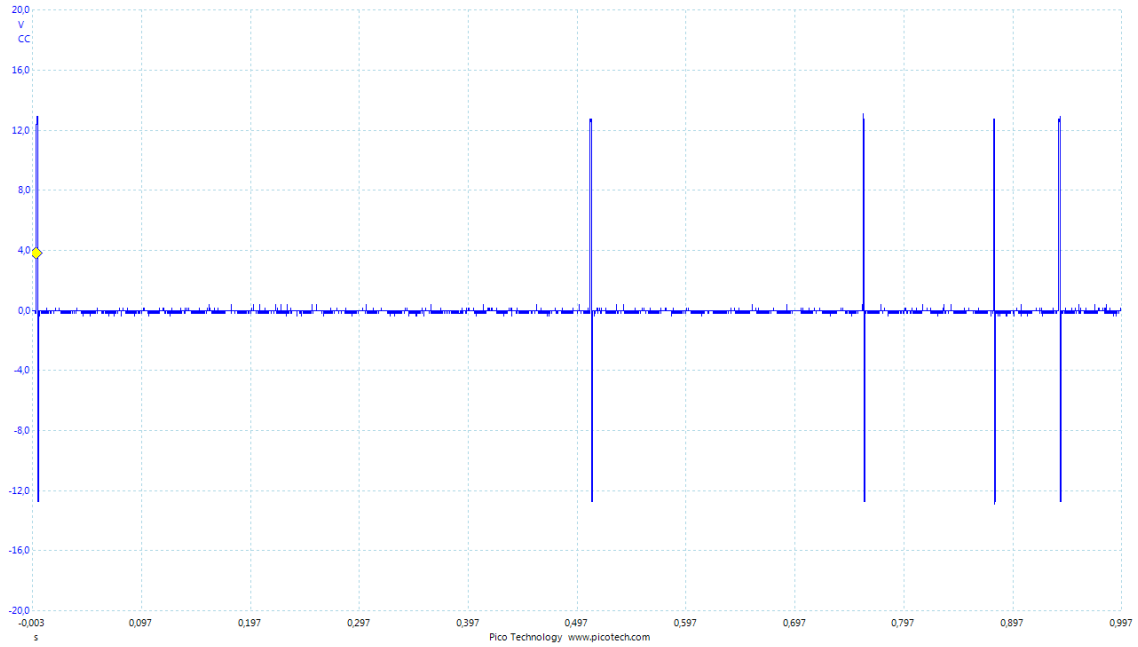

Ilustración 57. Estímulo de un único canal con 13V de estímulo, 2ms de ancho de pulso y con el protocolo de estimulación de frecuencias crecientes de la Tabla 2

A continuación se comprobará que nuestro estimulador puede aplicar un protocolo de estimulación S1-S2. El protocolo S1-S2 consiste en un protocolo en el que se aplica un tren de estímulos espaciados con un periodo S1 constante, y a continuación se aplica un último estímulo tras un periodo S2 diferente a S1 (ver Ilustración 58).

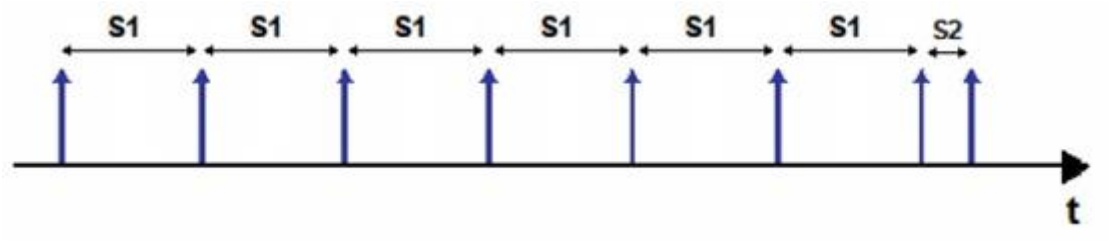

Ilustración 58. Protocolo de estimulación S1-S2.

Para probarlo en nuestro sistema se implementará un protocolo S1-S2 en el que aplicaremos un tren de estímulos espaciados 250ms seguido de un último estímulo separado 120ms (ver Ilustración 59).

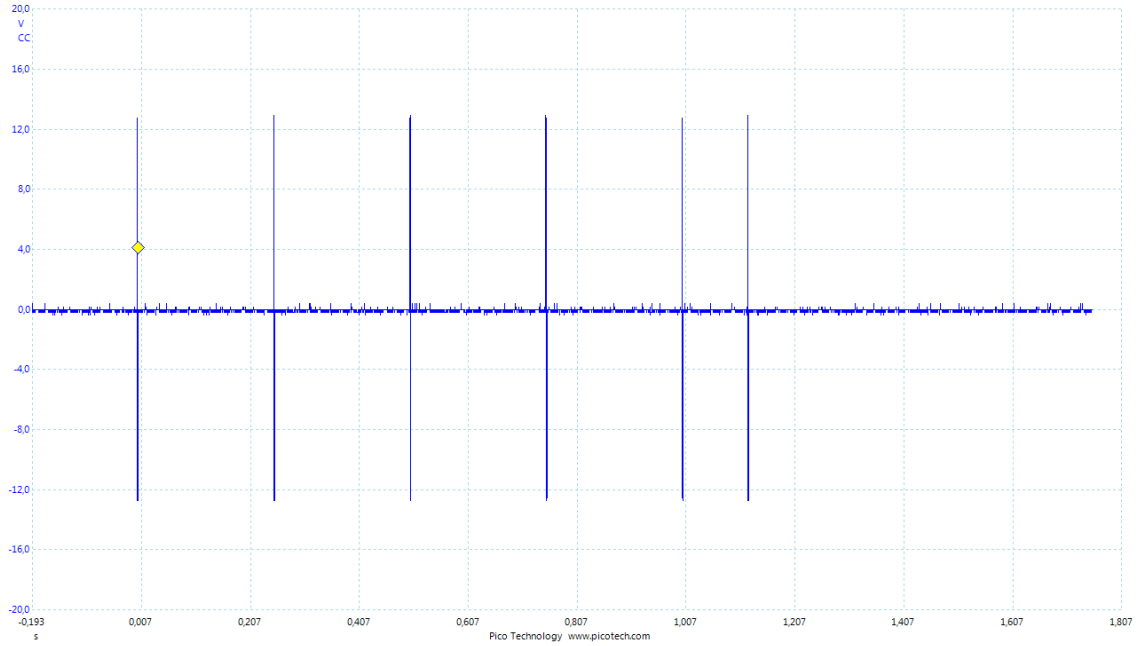

Ilustración 59. Aplicación en un único canal de un protocolo S1-S2 con una amplitud de 13V y un ancho de pulso de 2ms. El protocolo consiste en un tren de estímulos separados 250ms seguido de un último estímulo espaciado 120ms.

4.2.4 Análisis de persistencia de la temporización.

En esta sección tendremos el estimulador funcionando durante una hora y comprobaremos que la temporización del ancho y del periodo se mantienen estables con el tiempo.

Este análisis se realizará empleando una función de nuestro osciloscopio digital, en la que realiza capturas por flanco de subida que son mantenidas en la pantalla. El osciloscopio colorea de rojo los puntos que más veces ha adquirido a lo largo del tiempo, y de azul los que menos.

En la Ilustración 60 se ve una acumulación de todas las capturas del estímulo durante una hora de funcionamiento. En ella podemos apreciar que tras una hora estimulando, todos los pulsos han sucedido en el mismo espacio de tiempo, y no hay desviaciones de unos estímulos a otros.

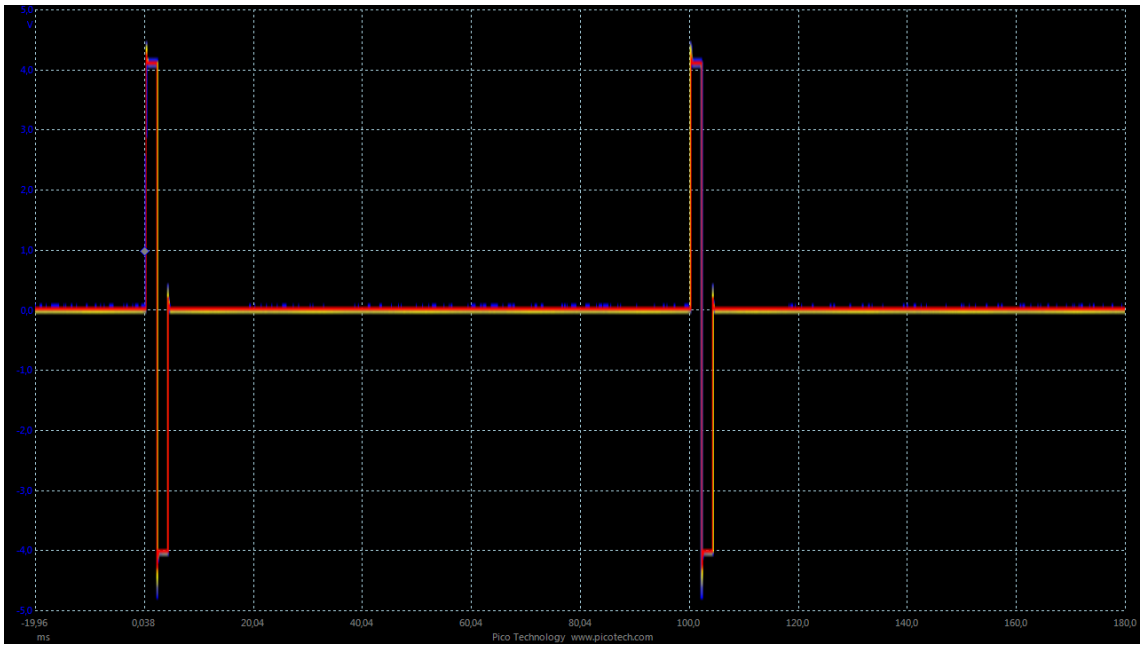

Ilustración 60. Test de persistencia de la temporización con un estímulo de 40V, con un ancho de pulso de 6ms y un periodo de 100ms entre estímulos. Aunque en el eje Y se observa que los valores van de 0 a 4v, esto es debido a que se empleó una sonda atenuadora x10. Los picos observados se deben a la capacidad que introduce la sonda atenuadora.

Capítulo 5. Presupuesto.

Qty	Value	Device	Package	Precio unitario	Precio
1		Pulsador	320-938	6,12 €	6,12 €
4	ESQ-110-14-G-S	Conectores		3,02 €	12,08 €
4	ESW-110-14-G-S	Conectores		1,21 €	4,84 €
1	ESW-110-14-G-D	Jumpers		2,42 €	2,42 €
1	0.1u	Condensador electrolítico.	E2,5-6	0,23 €	0,23 €
1	0.33u	Condensador electrolítico.	E2,5-6	0,15 €	0,15 €
11	10k	Resistencia	0309/10	0,03 €	0,33 €
1	120	Resistencia	0309/10	0,01 €	0,01 €
1	1uF	CPOL-EUE2.5-6	E2,5-6	0,17 €	0,17 €
2	MKDSN2,5/2-5.08	Bloque de terminal de tornillo de 2 vías		0,66 €	1,32 €
1	MC000047	Bloque de terminal de tornillo de 3 vías		0,89 €	0,89 €
1	74LS14N	74LS14N	DIL14	2,01 €	2,01 €
2	ADG408BR	Multiplexor analógico.	SOIC127P600X175-16N	4,89 €	9,78 €
1	ARDUINO_UNO	Placa Arduino	ARDUINO_UNO	24,20 €	24,20 €
1	Led verde	Led 5mm	LED5MM	0,19 €	0,19 €
1	LM7809	Regulador de tensión de 9V	78MXXS	0,56 €	0,56 €
4	LT1097	Amplificador operacional de precisión	DIL08	2,99 €	11,96 €
2	MCP4726	DAC 12 bits	SOT23-6	0,71 €	1,42 €

1	VAWQ6	Convertidor DC-DC dual	DIL24-6	17,34 €	17,34 €
2	1k	Resistencia	0309/10	0,7 €	0,14 €
2	20	Resistencia	0309/10	0,5 €	0,10 €
2	3.3k	Resistencia	0309/10	0,03 €	0,06 €
4	360	Resistencia	0309/10	0,03 €	0,12 €
2	C347S	Relé optoacoplador	SOP14	6,63 €	13,26 €
2	OPA454AIDDA	Amplificador operacional de alta tensión y corriente	SOIC127P600X170-9N	7,51 €	15,02 €
2	TIP142	Transistor Darlington NPN	TO220AH	1,20 €	2,40 €
2	TIP147	Transistor Darlington PNP	TO220AH	0,88 €	1,76 €
	TOTAL				128,88 €

Tabla 3. Presupuesto del estimulador cardiaco.

Capítulo 6. Pliego de condiciones.

Las condiciones que nuestro proyecto debía de cumplir son las siguientes:

- Posibilidad de comunicación con una interfaz de usuario en el PC.
- La interfaz de usuario tiene que estar basada en un entorno gráfico.
- El estimulador tiene que funcionar aun estando desconectado del PC.
- Dos canales completamente independientes en su configuración.
- Posibilidad de programar tensión, frecuencia y ancho del pulso.
- Rango de tensiones de -40 a 40V.
- Tiene que disponer de un sistema de relés que, excepto en los momentos en que se apliquen los pulsos de estimulación, aíslen el estimulador del corazón.
- Dichos relés tienen que aguantar tensiones de 40V y corrientes de 1A.
- Tiene que disponer de un pulsador que permita detener la estimulación.
- Las pistas de la PCB tienen que soportar un régimen de pulsos a 1A durante intervalos largos de tiempo.

Capítulo 7. Conclusiones y propuestas de trabajo futuro.

Este proyecto tenía como objetivos generales el diseño y realización de un estimulador electrónico programable de dos canales para corazones bioartificiales.

Para llegar a cumplir dicho objetivo se dividió el proyecto en las distintas tareas descritas en la sección 3.2, y ahora analizaremos si hemos cumplido los objetivos propuestos.

1. Se han estudiado estimuladores eléctricos programables ya desarrollados con anterioridad, analizando sus deficiencias y posibles mejoras, así como que partes del diseño se podían reutilizar.
2. Se ha realizado un diagrama de bloques que nos permite tener una visión general de que subsistemas debe de tener nuestro estimulador, como deben estar conectadas y su función dentro del conjunto.
3. Una vez se ha hecho un diagrama de bloques, se han elegido los distintos componentes para realizar cada uno de los subsistemas y se ha diseñado el esquemático de nuestro estimulador.
4. Se ha realizado la programación del firmware del Arduino. Dicho firmware permite que a través de comandos enviados por el puerto serie del PC, podamos reconfigurar la amplitud, la frecuencia y el ancho de dos estímulos independientes.
5. Se ha diseñado un programa de PC que, mediante un interfaz gráfico, nos sirve de interfaz de usuario para reconfigurar el estímulo de cada uno de los dos canales, todo ello de forma transparente al usuario.
6. Se ha diseñado y montado una PCB de doble cara por el método de fresado, que nos permite bajo un diseño modular, disponer de cualquiera de los dos canales, replicado las veces que queramos.
7. Se ha realizado un test conjunto, software, firmware, hardware, comprobando que la intercomunicación entre las tres partes del proyecto funciona correctamente. Se ha comprobado que el estimulador funciona correctamente bajo las condiciones límite, es decir, con 40V de amplitud de estímulo, y con 1A de corriente, comprobando que los componentes y las pistas de la PCB soportan dichas condiciones. También se ha testeado el uso de protocolos de comunicación, que era condición de diseño. Por último se ha testeado también que durante una hora de funcionamiento, ninguno de los tres parámetros a configurar, amplitud, frecuencia, o ancho, posee deriva.

Basándonos en los puntos mencionados, podemos concluir que el diseño e implementación de nuestro estimulador ha sido satisfactorio, habiéndose cumplido los objetivos y pliego de condiciones impuestos.

Aun así, hay ciertas ideas que podrían implementarse en una versión futura de nuestro estimulador:

1. La realización de la PCB será enviada a fabricar a un fabricante de PCB que incluya máscara de soldadura y vías metalizadas.
2. El montar el Arduino sobre la placa principal con los pines de expansión hacia arriba, nos permite poder conectar otros módulos de Arduino directamente en nuestro sistema y reprogramar el firmware para utilizarlos. Un módulo que sería de gran utilidad sería el módulo ZigBee de Arduino, el cual conecta un chip transceptor de ZigBee al Arduino por medio del puerto serie. De esta forma, de manera tan sencilla como la escritura de comandos a través del puerto serie, que es la misma filosofía empleada en nuestro proyecto con el USB, podemos mandar y recibir comandos de manera inalámbrica. Conectando un dongle transceptor de ZigBee al PC, se podrá controlar los diferentes dispositivos del laboratorio (estimulador, LEDs y cámaras para Optical Mapping, bombas de perfusión, etcétera) desde un mismo PC.
3. Se procederá al diseño que una caja especialmente diseñada para contener el estimulador. Los conectores destinados a la alimentación, los electrodos o comunicación USB estarán ensamblados en la caja, así como el LED indicador y el botón de emergencia que será de muy fácil acceso en la parte exterior de la caja.

Capítulo 8. Bibliografía

[1] Bayes-Genis A, et al. Human progenitor cells derived from cardiac adipose tissue ameliorate myocardial infarction in rodents. *J Mol Cell Cardiol* 49, p. 771 -80. 2010

[2] Houtgraaf JH, et al. First experience in humans using adipose tissue -derived regenerative cells in the treatment of patients with ST -segment elevation myocardial infarction. *JAm Coll Cardiol* 59, p.539 -40. 2012

[3] Le Blanc K, et al. HLA expression and immunologic properties of differentiated and undifferentiated mesenchymal stem cells. *Exp Hematol* 31, p.890 -6. 2003

[4] Vunjak-Novakovic G, et al. Challenges in cardiac tissue engineering. *Tissue Eng Part B Rev* 16, p.169 -87. 2010

[5] Perez-Ilzarbe M, et al. Characterization of the paracrine effects of human skeletal myoblasts transplanted in infarcted myocardium. *Eur J Heart Fail* 10, p. 1065-72. 2008

[6] Pelacho B, et al. Multipotent adult progenitor cell transplantation increases vascularity and improves left ventricular fuction after myocardial infarction. *J Tissue Eng Regen Med*, 1, p. 51-9. 2007

[7] Aranguren XL, et al. Multipotent adult progenitor cells sustain function of ischemic limns in mice. *J Clin Invest* 118, p. 505-14. 2008

[8] Uemura R. et al. Bone marrow stem cells prevent left ventricular remodeling of ischemic heart throught paracrine signaling. *Circ Res* 98, p.1414-21. 2006

[9] Hou D, et al. Radiolabeled cell distribution after intramyocardial, intracoronary, and interstitial retrograde coronary venous delivery: implications for current clinical trials. *Circulation* 112, p.I150 -6. 2005

[10] Eschenhagen T, et al. Three-dimensional reconstitution of embryonic cardiomyocytes in a collagen matrix: a new heart muscle model system. *FASEB J* 11, p. 683-94. 1997

[11] Zimmermann WH, et al. Tissue engineering of a differentiated cardiac muscle construct. *Circ Res* 90, 223-30. 2002

[12] Arduino, "Arduino Board Uno". http://arduino.cc/en/Main/ArduinoBoardUno [online]

[13] Atmel Corporation: "ATmega48A; ATmega48PA; ATmega88A; ATmega88PA;

ATmega168A; ATmega168PA; ATmega328; ATmega328P Datasheet" p1.

[14] Arduino, "Arduino Playground – MsTimer2".

http://playground.arduino.cc/Main/MsTimer2 [online]

[15] Microchip: "MCP4706/MCP4716/MCP4726 Datasheet" p1.

[16] Microchip: "MCP4706/MCP4716/MCP4726 Datasheet" p.46.

[17] Wikipedia: “I2C”. http://en.wikipedia.org/wiki/I%C2%B2C [online]

[18] Robot Electronics: “I2C Tutorial”.

http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html [online]

[19] Linear Technology Corporation: “LT1097 Datasheet” p1.

[20] Texas Instruments: “OPA454 Datasheet” p1.

[21] On Semiconductor: “TIP140, TIP141, TIP142, (NPN); TIP145, TIP146, TIP147, (PNP) Datasheet” p1.

[22] Coto Technology: “C247S/C347S Datasheet” p2.

[23] CUI INC: “VAWQ3 Datasheet” p1.

[24] Microsoft: “.NET System Requirements”.

http://msdn.microsoft.com/en-us/library/8z6watww(v=vs.110).aspx [online]

[25] Wikipedia: “Common Language Infrastructure”

http://en.wikipedia.org/wiki/Common_Language_Infrastructure [online]

[26] Wikipedia: “.NET Framework” http://en.wikipedia.org/wiki/.NET_Framework [online]

[27] Microsoft: “What is the .NET Micro Framework – NETMF” http://www.netmf.com/what-is-the-net-micro-framework.aspx [online]

[28] Mono: “Mono”. http://www.mono-project.com/Main_Page [online]

Capítulo 9. Anexos

9.1 Esquemáticos.

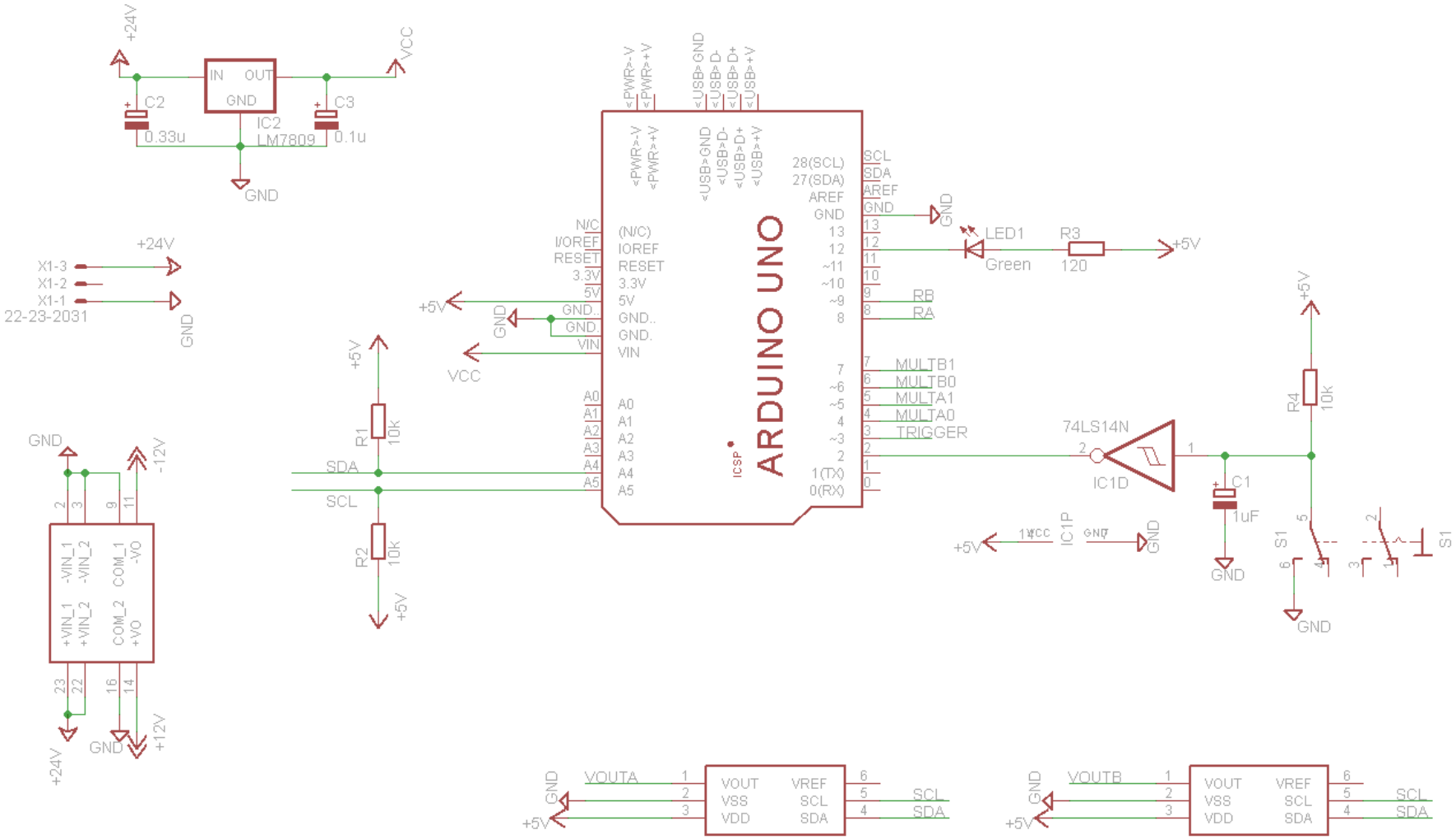

Ilustración 61. Esquemático de la placa principal. Página 1 de 2.

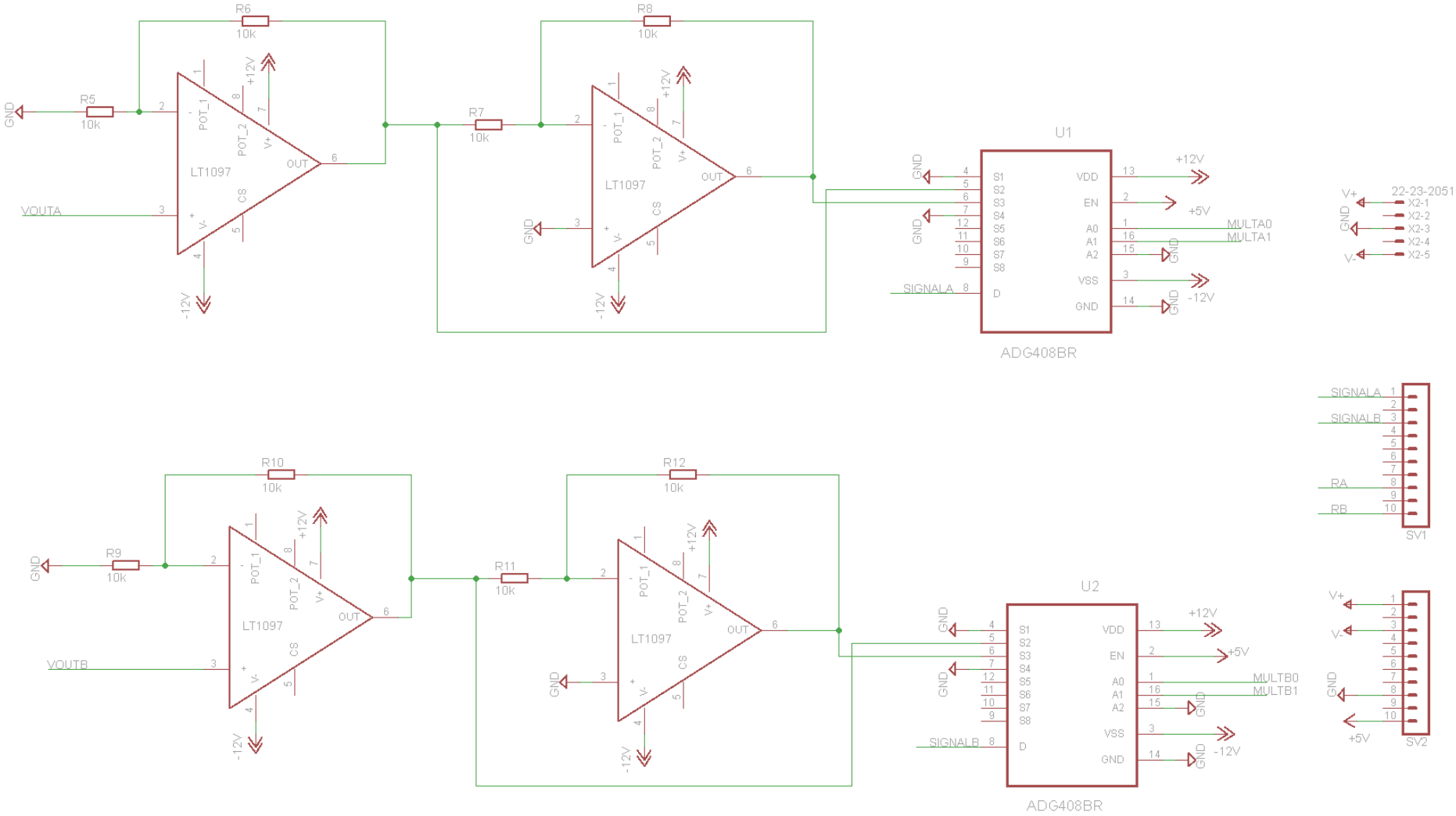

Ilustración 62. Esquemático de la placa principal. Página 2 de 2

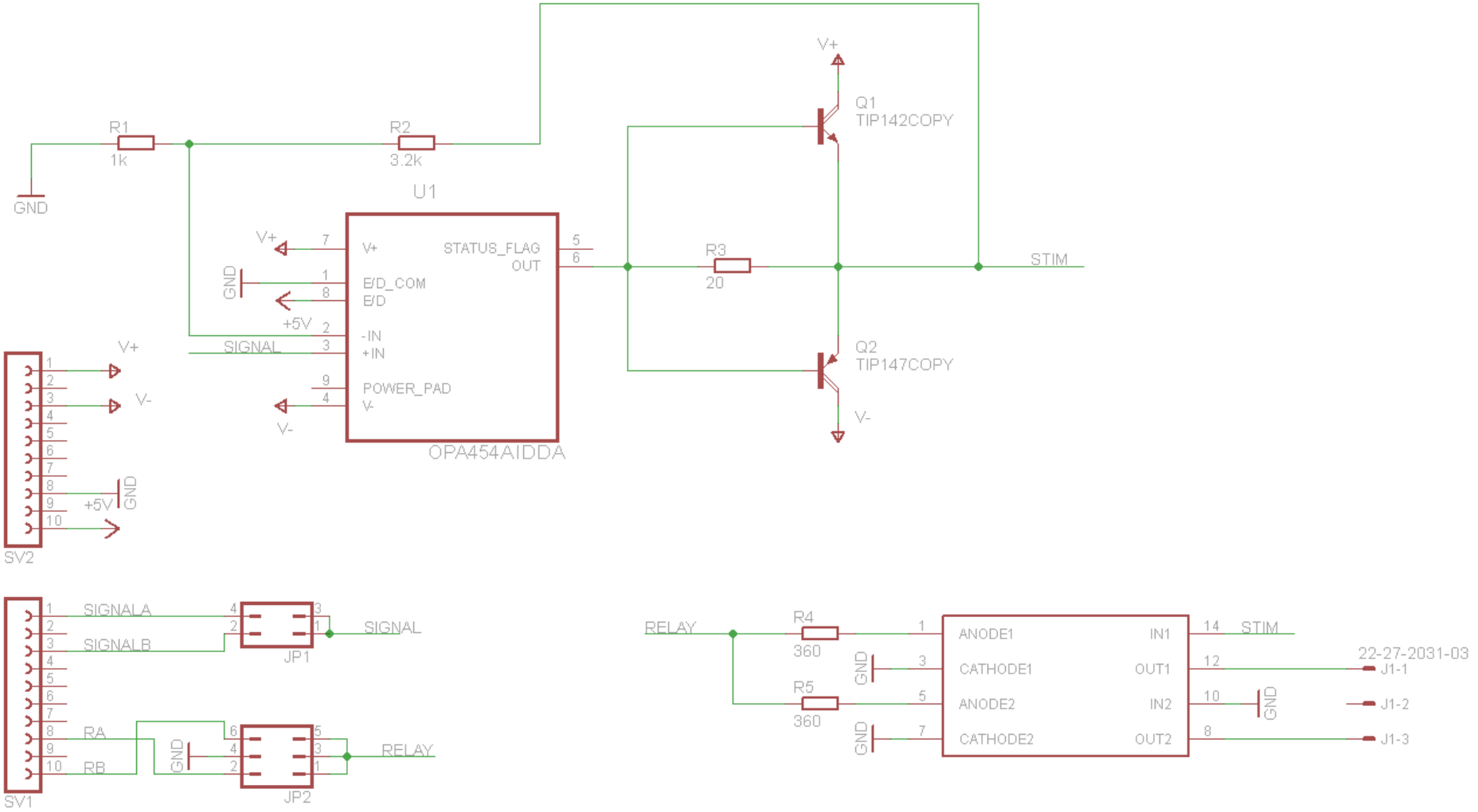

Ilustración 63. Esquemático de la placa de potencia.

9.2 Capas de las PCB.

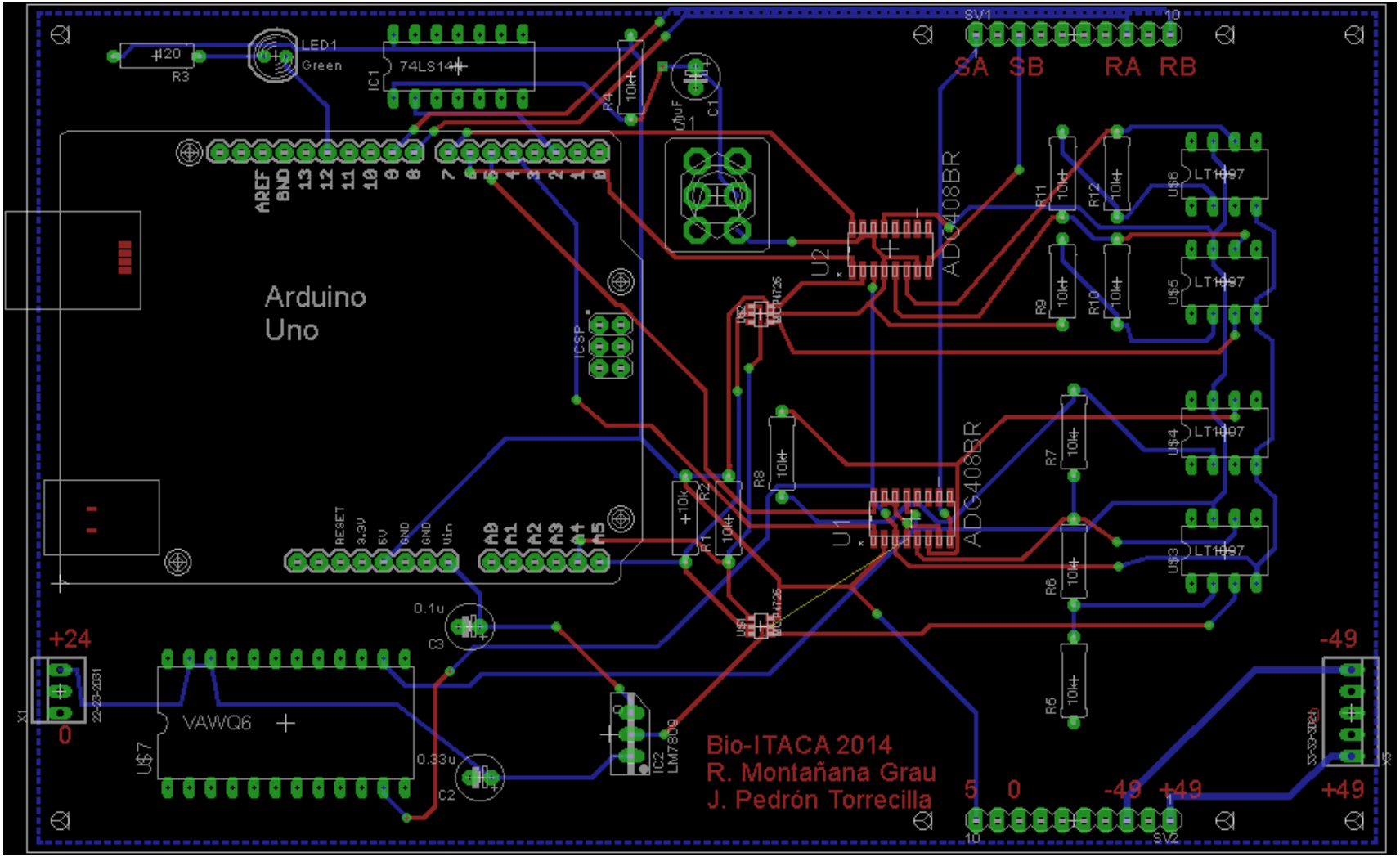

Ilustración 64. PCB de la placa principal.

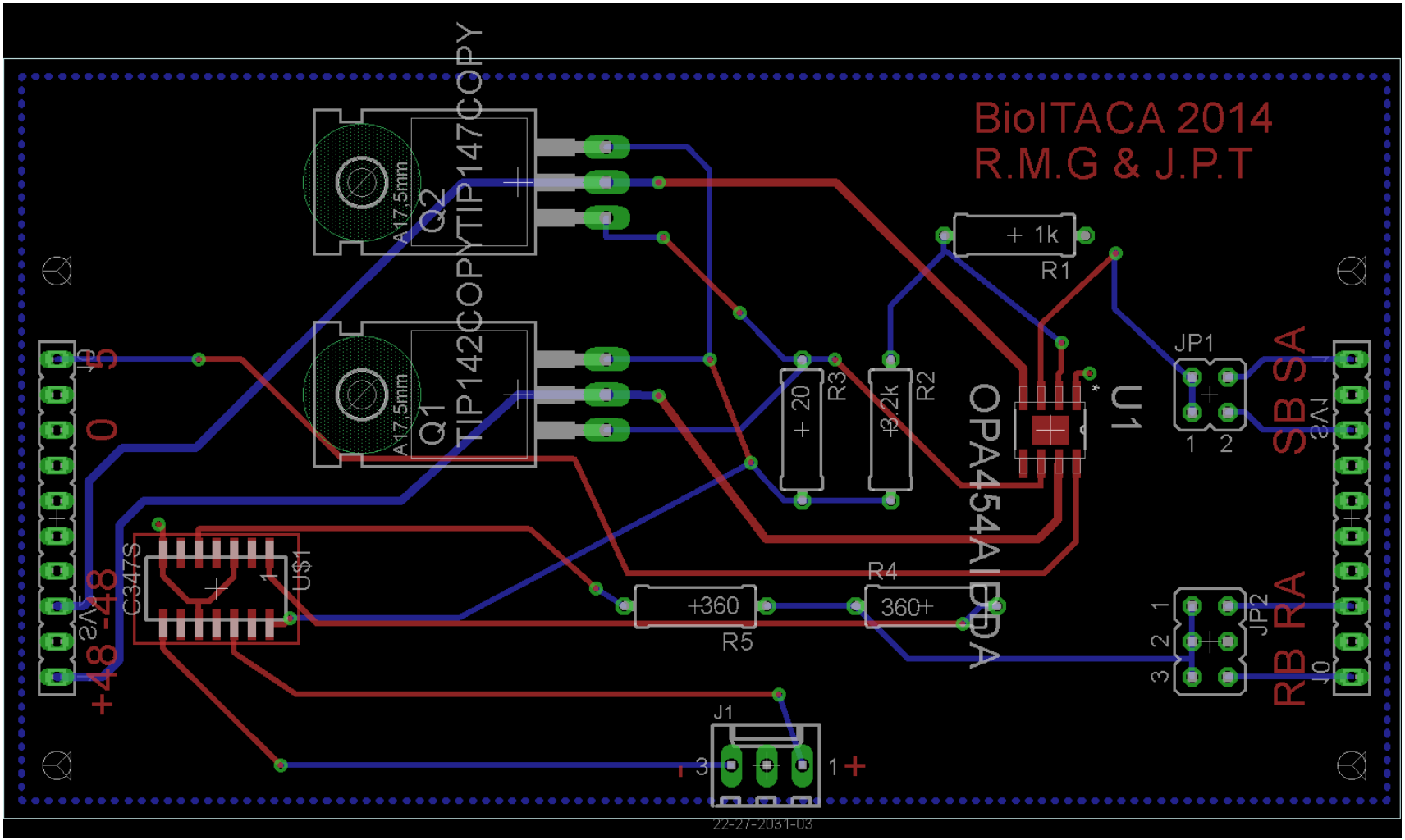

Ilustración 65. PCB de la placa de potencia