industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Generador de señales usando microcontroladores

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Pablo Ares Paredes**
Director: Manuel Sánchez Alonso

Cartagena, 2020

Agradecimientos

Quiero agradecer a mis padres por todo el apoyo que me han dado durante los años de carrera, sin ellos no hubiera sido posible. Gracias.

A mi tutor de proyecto, por las horas que ha dedicado a conseguir que este trabajo llegase a buen puerto.

Y por último a mis compañeros en general, pero de una manera especial a Víctor y Ramón.

Resumen

Este TFE tiene como propósito el diseño y desarrollo de un generador de señales. Dicho dispositivo es una herramienta esencial en numerosos ámbitos. En general todas las vertientes de la electrónica se benefician enormemente de su uso. También se utiliza en áreas como la física, la química y la música por poner sólo unos pocos ejemplos. Un generador de señales es útil en todas las fases de la vida de un dispositivo electrónico: investigación, desarrollo, depuración, construcción, mejoras y actualización, mantenimiento y reparación.

Para alcanzar los objetivos propuestos se usarán microcontroladores. Se requerirá comunicación entre ellos además de trabajo propio de cada uno. Por otro lado, la señal digital que se obtenga del microcontrolador deberá ser adaptada a las necesidades, por lo que será necesario diseñar y construir un circuito DAC que la convierta en analógica. Finalmente, para poder obtener una señal variable se usará circuitos amplificadores que permitan modificar amplitud y offset.

Índice

1. Prólogo

El transcurso de la mayoría de este TFE ha sucedido durante la pandemia del COVID y los confinamientos. Esto no intenta ser una excusa que justifique posibles errores sino una explicación a criterios de diseño y tomas de decisiones en la manera de haber llevado a cabo el trabajo.

Adapté el TFE para poder realizarlo desde casa con las cosas que disponía y las que compré. Así que, aunque desde junio se podía ir a la universidad para poder realizar trabajos de fin de grados, me hubiera supuesto rehacer muchas cosas y sin saber si volverían a haber restricciones que impidiesen ir a la universidad, lo que hubiera supuesto volver adaptarlo. Por todo esto, decidí seguir el trabajo como si sólo pudiese realizarlo desde casa.

2. Introducción

2.1. El mundo actual y la electrónica

El mundo en el que vivimos cada vez tiene más tecnología, y sin lugar a duda, la electrónica es una de las ramas que más impacto ha tenido en los últimos tiempos.

Inicialmente la mayoría de los dispositivos que se desarrollaban eran totalmente analógicos, en cambio hoy en día cada vez se le da más peso a la parte digital. No obstante, la electrónica analógica sigue siendo imprescindible entre otras cosas por cómo nos comunicamos los seres vivos con el entorno (nuestros sentidos son analógicos).

Al ser la electrónica un aspecto tan importante en la actualidad, el estudio y comprensión, así como su desarrollo y uso, son vitales.

Para poder trabajar con esta tecnología es necesario herramientas que nos permitan pasar de la teoría a la práctica. Hay muchas herramientas que dependiendo de la rama en la que nos centremos serán necesarias e imprescindibles o aconsejables. Pero, por lo general, todas coinciden en algunos instrumentos y/o dispositivos:

- Fuente de alimentación: Dispositivo que nos permite proveer de energía eléctrica a nuestro circuito. Se pueden categorizar de muchas maneras, pero las más relevantes depende de si la electricidad varía respecto del tiempo, serán fuentes de alterna si lo hacen y de continua si no. Otra clasificación viene dada de su capacidad para regularse. Hay modelos con control de corriente y de voltaje. Su precio dependerá del rango de voltaje y corriente que sean capaz de dar, así como de su exactitud a la hora de darlo.
- Multímetro: Es una herramienta de medición cuyo objetivo principal es poder medir voltaje, corriente y resistencia, de ahí su nombre. Actualmente viene con más funciones como pruebas de continuidad y diodos o termómetro entre otras. Pero todas se basan en medir una de las tres magnitudes primeras. Dependiendo de la exactitud con las que den las mediciones su precio variará, pero un multímetro para un uso normal suele ser asequible.
- Osciloscopio: Es una herramienta que nos permite visualizar la forma de onda que tiene el voltaje. Esto proporciona una cantidad enorme de información, que sumado a que los osciloscopios modernos también miden, hace que sean una herramienta muy potente para el estudio de la electrónica. Son dispositivos relativamente caros.
- Generador de señales: Un generador de señales es una herramienta que nos permite producir una onda de las características deseadas para introducirlas en un circuito y ver cómo se comporta. Esta herramienta es la compañera ideal del osciloscopio ya que con ambas se pueden estudiar la gran mayoría de situaciones que nos encontremos en el mundo de la electrónica.

Estos cuatro dispositivos son útiles en casi todos los aspectos de la electrónica y en los lugares donde se trabaja con la electrónica (talleres, laboratorios, fábricas, etc.…).

2.2. Generador de señales

Centrándonos en el generador de señales, que es el dispositivo que se va a tratar en el TFE, podemos añadir que dependiendo de su rango de características puede usarse tanto en todas las ramas de la electrónica como en otras áreas como son la música, la física o la química entre otras.

Un generador de señales se usa para producir una onda de unas características determinadas y conocidas por los usuarios. Estas características son principalmente frecuencia y/o periodo, amplitud y offset. También es interesante poder establecer otros valores como el ciclo de trabajo y el desfase respecto a otra onda. Antes de continuar detallando las cualidades de un generador de señales, procederemos a describir las características de una onda, pues será fundamental conocer que significan para poder manejar un generador de señales de manera apropiada.

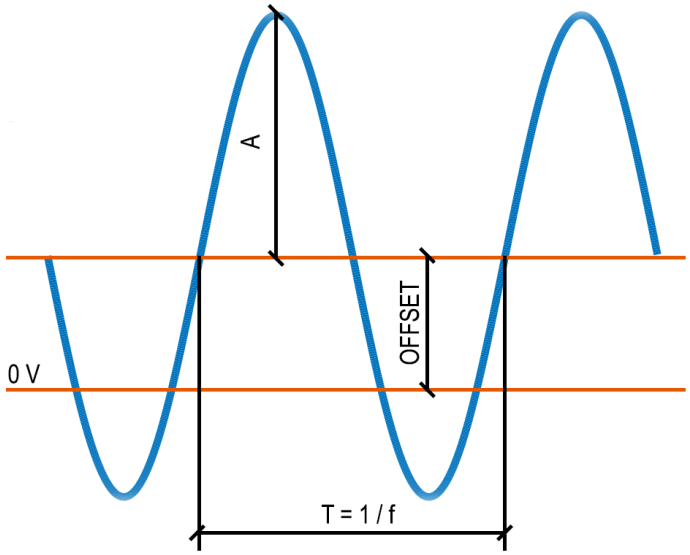

Fig. 1. Onda y sus características.

- Frecuencia (f): número de veces que la onda se repite por una unidad de tiempo. Es la inversa del periodo.
- Periodo (T): tiempo que tarda la onda en completar un ciclo. Es inversa de la frecuencia.
- Amplitud (A): valor que representa la magnitud de voltaje que alcanza la onda en su punto más alejado respecto de su centro.
- Offset: valor que representa el desplazamiento del centro de la onda respecto a la tierra del circuito (nivel cero). Otra manera de verlo es el nivel de continua que tiene la onda.
- Ciclo de trabajo: es un valor porcentual que indica la relación entre el tiempo que se encuentra activa, o en valor alto, la onda respecto al periodo total de la misma.

  Es la relación entre el tiempo que se encuentra la actividad, de señales que influyen en la onda, respecto al periodo total.
- Desfase: distancia entre el inicio de dos ondas.

El rango de estas características en un generador de señales determinará las áreas dónde se podrá usar. Por ejemplo, en el ámbito musical/auditivo, al tener los seres humanos un umbral de audición de hasta los $20\ kHz$, generadores de señales que superen por mucho ese valor son innecesarios. Por esta misma razón, muchos otros dispositivos que tienen señales alternas evitan entrar en el rango de frecuencia audible, sobre todo si son de potencia.

Sucede lo mismo si nos vamos al terreno de la electrónica, dependiendo del área en la que queramos trabajar, las características de un generador de señales pueden ser superfluas, necesarias o imprescindibles. Si queremos simular la red eléctrica necesitaremos frecuencias de 50-60 Hz. En el ámbito de las fuentes de alimentación se suele trabajar por encima de los 50Khz. En la radiofrecuencia necesitaremos desde los cientos de kHz a los GHz. Por lo tanto, los generadores de señales son dispositivos se pueden aplicar a motores, luz, radio, circuitos amplificadores, aplicaciones especiales, etc.

Los semiconductores tienen un papel fundamental en la electrónica, especialmente los transistores. Se usan en casi cualquier aplicación y dependiendo del tipo de tecnología a la que pertenezcan tendrán unas características que los destinarán a unas determinadas aplicaciones. El voltaje que son capaces de soportar y la corriente que pueden resistir son dos de esas características, pero la que nos interesa es la frecuencia a la que son capaces de conmutar, ya que es la que será capaz de aportar un generador de señales. Los semiconductores más comunes son: transistores BJT (transistor de unión bipolar), transistores FET (transistor de efecto de campo), transistores IGBT (bipolar de puerta aislada) y tiristores con mención especial a los GTO (tiristor desactivado por compuerta). Dentro de los transistores FET los más usados son los JFET (transistor de efecto de campo de unión) y los MOSFET (transistor de efecto de campo metal-óxido-semiconductor). Todos estos semiconductores son los que más se emplean en la electrónica ya sea de consumo o industrial.

Al fijarnos en la clase de semiconductor, interesa que podamos trabajar con el mayor tipo posible. Observando la Fig. 2, podemos ver que con $100\ kHz$ ya cubriríamos la mayoría de los semiconductores, y con ello, las aplicaciones donde se usan.

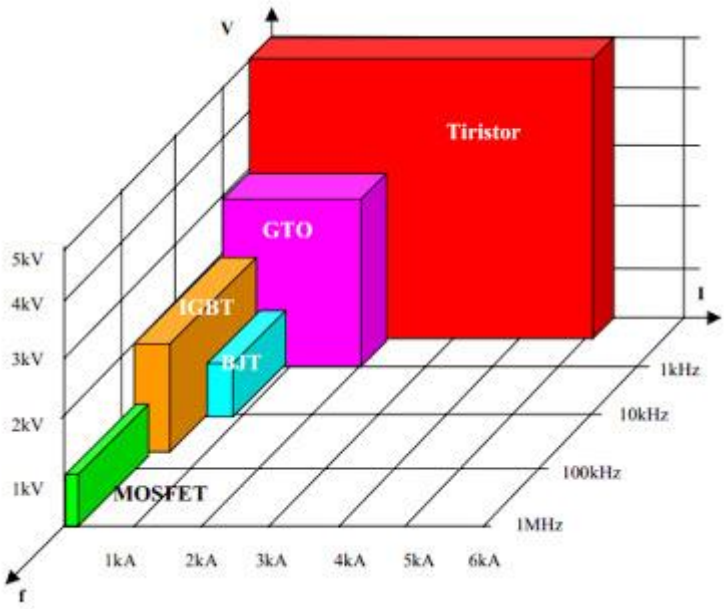

Fig. 2. Semiconductores y su aplicaciones.[1]

[1] Fig. 2. Véase la referencia [1].

2.3. Tipos de generadores de señales

La primera manera de clasificar los generadores de señales es atendiendo a la manera en la que consiguen formar la onda. Al fijarnos en esa característica encontramos:

- Analógicos: Mediante componentes discretos y/o integrados se construyen osciladores que consiguen producir señales básicas como cuadrada, diente de sierra y triangular. Para generar las formas de onda disponen de un circuito especial para cada una de ellas que las genere en su totalidad o transformar una onda ya generada. Esto complica tener muchos tipos de ondas en un dispositivo e incluso imposibilita la formación de ondas complejas. Actualmente han sido desplazados por los digitales.

  Como ejemplo básico de funcionamiento podemos ver como se obtendrían la onda cuadrada y la triangular en este tipo de generadores. Normalmente consiguen primero la onda cuadrada conmutando entre un valor alto y uno bajo de voltaje. Luego, integrando esa onda mediante un amplificador operacional, en forma de integrador, se consigue la señal triangular.

- Digitales: Se basan en ir dando valores de voltaje en el orden preciso para obtener la forma de onda deseada. Este tipo de generadores facilita mucho la construcción de diversas ondas tanto simples como complejas. Una manera de conseguir esto es empleando bucles en los que usando operaciones matemáticas se consiguen los valores que conformaran la onda. Otra manera es almacenar los valores de la señal en una tabla e ir sacándolos para formar la onda. Posteriormente las ondas generadas se modifican si es preciso como en los casos de modulación de amplitud (AM) o de frecuencia (FM).

Si los clasificamos según en lo que podemos encontrar en el mercado [2]:

- Generador arbitrario de ondas: Este tipo de generadores permite generar casi cualquier onda periódica, sólo es necesario muestrear la onda en cuestión, y almacenarla en un espacio de la memoria del microcontrolador, obteniendo lo que podemos llamar tabla, de manera que pueda acceder a ella. Mediante un bucle se sacan, en el momento preciso, los valores de dicha tabla en orden, de esta manera obtenemos la onda deseada.

- Generador de audio: Genera ondas para uso en aplicaciones de audio, por lo que el rango de frecuencia se encuentra entre los 20 Hz y los 20 kHz. Normalmente generan sólo ondas senoidales con muy baja distorsión armónica y una respuesta muy plana, es decir, que la salida no se vea alterada en amplitud a lo largo de su rango de frecuencia.

- Generador de funciones: Es un tipo de generador centrado en ondas simples. Normalmente cuadrada, triangular, diente de sierra y senoidal. Tienen el inconveniente de que no suelen llegar bien a las bajas frecuencias. Antiguamente estos tipos de generadores eran construidos de forma analógica. Hoy en día se fabrican de manera digital y posteriormente se convierte en analógica.

- Generador de pulso: Su uso y diseño está enfocado en generar pulsos y el tiempo deseado entre ellos. Las aplicaciones más comunes de estos dispositivos suelen ser para sistemas digitales, aunque también pueden ser útiles en los analógicos.

- Generador de señales de RF: Tienen el objetivo de obtener señales de radiofrecuencia (RF). Su característica principal es la modulación. Las técnicas de modulación que

sean capaces de trabajar determinarán aplicaciones concretas en las que serán sumamente útiles.

La característica de modular se consigue de varias maneras dependiendo del tipo de modulación que se requiera, pero básicamente se basa en dos señales, la portadora (la señal base) y la moduladora (la señal que tiene la información que se quiere transmitir). Los tipos más comunes de modulación son amplitud (AM), frecuencia (FM) y fase (PM), pero también existen muchos más como modulación en cuadratura (QM), modulación combinada, codificación entrelazada (TCM), cancelación de eco, etc.

- Generador de señales vectorial: Es un generador de señales cuyo propósito es el de modular señales de radiofrecuencia. El objetivo es muy similar a los generadores de RF, pero usa técnicas más avanzadas y complejas. Se utilizan para generar ondas que se usan en telecomunicaciones modernas, como Wifi o 5G.

2.4. Objetivo

El objetivo es construir un prototipo de generador de señales que sea lo más versátil posible y a la vez asequible. Está destinado, inicialmente, a ser un dispositivo que se pueda usar en la mayor cantidad de situaciones que se pueden encontrar en el grado de Ingeniería Industrial Electrónica y Automática, tanto en la realización de prácticas como en circuitos descritos en teoría que se deseen probar. Pero las características con las que contará serán suficientes para usos generales en muchos ámbitos de la enseñanza y laborales, siempre siendo consciente de que el presupuesto se intentará que sea económico.

Tras revisar asignaturas vistas en la carrera, preguntar algunos profesores y haber buscado los usos generales de un generador de señales se proponen conseguir los siguientes objetivos mínimos para intentar que sea un dispositivo asequible pero útil:

- Variable en frecuencia, amplitud, y offset.
- Tipo de ondas: cuadrada, triangular, diente de sierra, seno e introducida por el usuario.
- Una frecuencia mínima de $50\ Hz$.
- Una frecuencia máxima al menos de $200\ kHz$, con al menos 10 muestras.
- Salida máxima de $15\ V_{pp}$.
- Resolución de voltaje de máximo $100\ mV$.
- Control del dispositivo directo e indirecto. Una mediante un LCD, un encoder y unos potenciómetros. La otra será una aplicación para PC.

3. Marco teórico

El proyecto se dividirá en cuatro bloques: generador de la onda, adaptación de la onda, interfaz y comunicación y alimentación. A lo largo del resto del informe se irán estudiando estos bloques en las diversas secciones pertinentes.

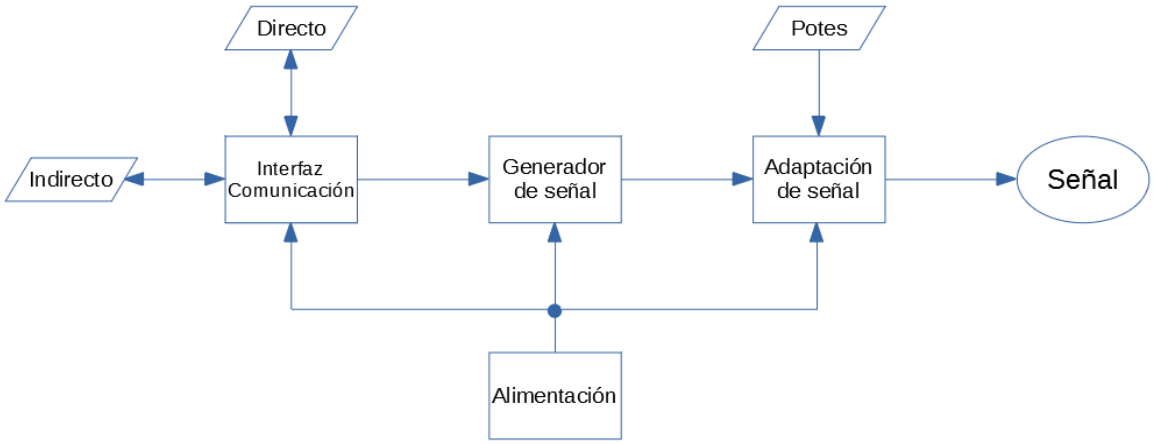

Fig. 3. Diagrama de flujo general del generador de señales.

El funcionamiento general del dispositivo se muestra en la Fig. 3. Empezamos con el bloque de interfaz y comunicación con la que el usuario podrá controlar las características de la onda que desee conseguir y otros ajustes. Por otra parte, el bloque generador de señal será el encargado de crear los valores de voltaje necesarios. Estos valores son digitales, por lo que serán acondicionados por el bloque de adaptación de señal para obtener la onda analógica deseada. Además, se conseguirá amplificar el valor querido. Por último, el bloque de alimentación proporcionará los voltajes necesarios a las demás partes del dispositivo.

3.1. Bloque generador de señal

El bloque generador de señal es la parte del dispositivo que se encarga, cómo su propio nombre indica, de generar la onda. El método para obtener una onda se denomina DDS (Direct Digital Synthesis) [3]. La idea de este método se basa en tener en la memoria del microcontrolador los valores que tomaría la onda, a esto le llamaremos tabla. Luego lo que hay que hacer es ir sacando en orden los valores de dicha tabla, y si entre cada valor que saquemos, dejamos pasar el tiempo preciso, la onda tendrá la frecuencia deseada. Una manera de verlo es como en la Fig. 4. Imaginemos que los valores de la tabla están situados en un círculo y cada valor de la tabla en un grado. Tenemos una variable, a la que llamaremos $punteroTabla$, cuya función es saber en que posición del círculo nos encontramos. Solo nos queda por saber cuánto tenemos que girar cada vez. Esto lo guardaremos en una variable a la que llamaremos $incrementoPuntero$. Para saber el valor que tendrá esta variable tenemos que saber varias cosas: frecuencia deseada ($Fout$), los valores totales de la tabla ($Estados$) y la frecuencia a la que sacamos una muestra ($F_{muestra}$).

$$incrementoPuntero = \frac{Estados * Fout}{F_{muestra}}$$

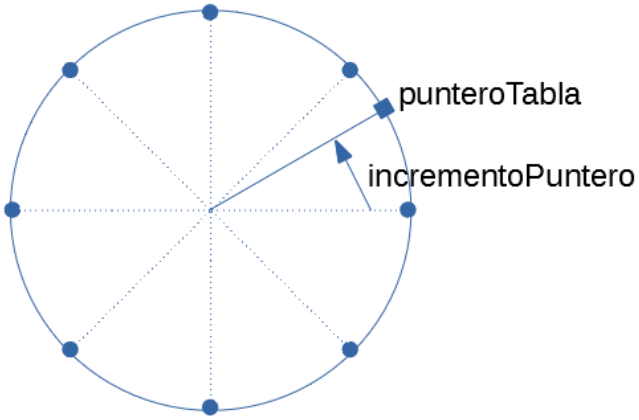

Fig. 4. Funcionamiento DDS.

Dependiendo del microcontrolador la $F_{muestra}$ será distinta, porque depende de la frecuencia de instrucción a la que trabaje. También influirá el número de instrucciones que necesitemos para sacar las muestras.

Este bloque se construirá con un microcontrolador y los componentes que se necesitarán para funcionar correctamente: resistencias, condensadores y un cristal de cuarzo. Las características que se buscan en este bloque son:

- Alta frecuencia de funcionamiento. Está directamente relacionada con la frecuencia final que obtendremos.
- Que trabaje en 16 bits, lo cual proporcionará una resolución de salida muy alta.
- Que disponga de una puerta con los 16 bits del registro con salida el exterior.
- Memoria suficiente para poder guardar todas las tablas que almacenen las ondas.
- Posibilidad de programarlo en lenguajes de más alto nivel que ensamblador.
- Bus de comunicación, al menos UART. Permitirá comunicarse con el bloque interfaz.
- Funciona a 5 voltios, lo que simplifica el desarrollo y el esquemático al juntarlo con Arduino.

3.2. Bloque adaptación de señal

Este bloque se divide en 2 partes. La primera es un DAC (Digital Analogic Converter) que convertirá la señal digital proveniente del bloque generador de onda en una analógica. La segunda parte es un circuito con AO (amplificadores operacionales) que se encargará de amplificar o atenuar la señal a los valores deseados, así como añadir offset a la señal. También se utilizará con el objetivo de hacer que la onda mantenga sus parámetros.

3.2.1. DAC

Para convertir una señal digital en una analógica se pueden usar varias técnicas. Estas se pueden agrupar en métodos con circuitos integrados y en métodos con componentes discretos.

Los primeros se basan en un circuito integrado que proporciona un valor analógico de voltaje dada una señal digital. Dependiendo del integrado la señal digital puede venir en un bus serie (SPI, I2C, UART…) o paralelo, que es como se implementará en este caso.

La manera elegida para realizar el DAC es mediante una red de resistencias R-2R. Esto no es más que una red compuesta por resistencias de dos valores distintos (un valor cualquiera R y el doble de ese valor, 2R) que consigue ponderar unas entradas digitales de N bits a su valor correspondiente analógico. El funcionamiento de este tipo de DAC se detalla a continuación y las explicaciones se refieren a la Fig. 5.

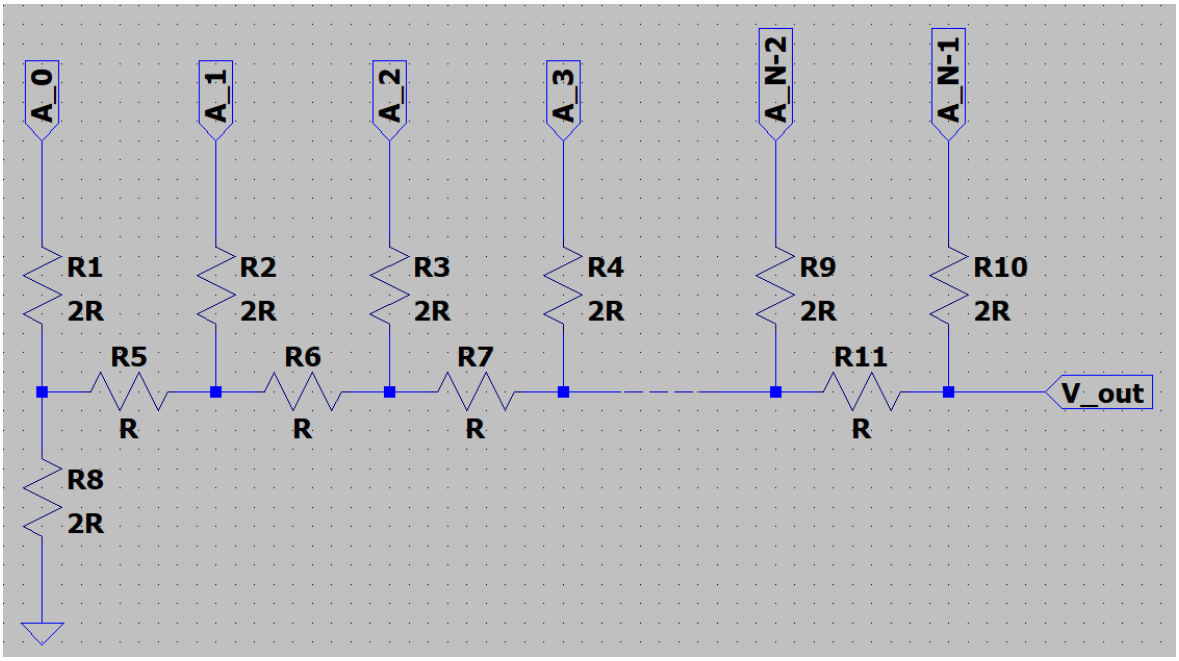

Fig. 5. DAC con red R-2R de N bits.

Cada entrada digital (A_i) se encuentra siempre con una resistencia 2R y un circuito a su izquierda con un valor equivalente de 2R, por lo que entre las dos se tendrá la mitad del voltaje de la entrada digital A_i. A esto hay que sumarle el voltaje de cada bit anterior (de menos peso) con lo que se irá acumulando el valor real del número digital que se haya puesto.

De manera matemática se puede ver como un sumatorio de todos los bits por el valor de voltaje que alcanzan al estar en un nivel alto (V_{ref}), divididos por su peso. N es el número de bits.

$$V_{out} = \left(\frac{A_0}{2^{N-0}} + \frac{A_1}{2^{N-1}} + \frac{A_2}{2^{N-2}} + \frac{A_3}{2^{N-3}} + \cdots + \frac{A_{N-2}}{2^{N-(N-2)}} + \frac{A_{N-1}}{2^{N-(N-1)}}\right) * V_{ref}$$
$$= \sum_{i=0}^{N-1} \frac{\mathrm{A_i}}{2^{\mathrm{N-i}}} * V_{ref}$$

Otra manera de verlo si trabajamos con el valor decimal del número digital:

$$V_{out} = V_{ref} * \frac{Valor_{dec}}{2^N}$$

El uso de una red de resistencias R-2R como DAC tiene una serie de ventajas:

- Simple de construir o fabricar. Solo se necesitan dos valores de resistencia, uno si las resistencias 2R se consiguen colocando en serie dos resistencias de valor R o

poniendo en paralelo dos de 2R para conseguir las resistencias de valor R. Al ser siempre los mismos componentes facilita su compra.

- Económico. De la misma manera, al ser siempre los mismos componentes será más barato al comprarlo en cantidades mayores. En caso de mandarlo a fabricar, será más económico en pequeñas tiradas, puesto que muchas veces se encarece más por usar distintos componentes que por usar muchos si son el mismo. Esto se debe a que en la mayoría de las fábricas donde se montan, cobran por cada tipo distinto de componente, además del precio del mismo componente (que puede ser insignificante respecto al primer coste).
- Rapidez. Al ser componentes discretos, y además puramente resistivos, el retardo de propagación es mínimo.
- Rango de voltaje amplio. Al ser las resistencias componentes que aguantan al menos varias docenas de voltios, pueden trabajar en cualquier voltaje que envíe el microcontrolador.

Por el contrario, también presenta desventajas frente a otros tipos de DAC:

- Muchos componentes en comparación a un integrado que haga la misma función. Si se monta a mano puede ser fatigoso en comparación a un integrado.
- Dependiente de la tolerancia. La desviación del valor nominal de las resistencias afecta mucho, sobre todo en los bits de más peso. Cuantos más bits tenga el DAC, más importante será el usar resistencias de baja tolerancia, al menos del 1%.

3.2.2. Circuito AO

Conseguir obtener una señal analógica no es suficiente si queremos que el generador de señales sea útil. Aunque obteniendo una señal analógica del DAC ya tendría muchas utilidades, el poder regularla en amplitud y offset le otorgaría al generador más versatilidad, ya que para muchas situaciones es imprescindible.

Este bloque tiene un doble objetivo. El primero es dotar de la posibilidad de regular la amplitud de la señal y de añadirle un offset. El segundo es procurar que la señal no se deforme ni realizarle los ajustes antes mencionados, ni al conectarla a diversas cargas que saturasen la red R-2R. Para poder realizar estas labores se optará por un circuito de amplificadores que constará de varias fases:

- Eviten saturaciones y deformaciones provenientes del DAC.
- Amplifiquen la señal hasta los máximos planteados en los objetivos.
- Añada el offset deseado a la señal.
- Eviten saturaciones y deformaciones debido a la carga a la que se le inyecte la onda.
- Filtren la alta frecuencia si es necesario. La misma placa atenuará las altas frecuencias que provengan de la conmutación digital del microcontrolador, pero puede ser necesario un filtro que haga un trabajo más exhaustivo.

El amplificador operacional deberá poder cumplir con el ancho de banda de las señales de salida, y para que la atenuación sea inapreciable, debería ser varias veces la frecuencia máxima de las señales que se vayan a generar.

Al tener como objetivo una salida de hasta $15\ V_{pp}$, interesa que, para señales en las que cambie de manera muy brusca de extremo a extremo (por ejemplo, una señal diente de

sierra), el slew rate sea de al menos de $15\ V/\mu s$, a ser posible del doble. Cuanto más alto sea el slew rate, mejor seguirá los cambios de onda que vienen del DAC.

3.3. Bloque interfaz y comunicación

El poder controlar el dispositivo nos da la posibilidad de cambiar el tipo de onda y la frecuencia, entre otras características. Esto es lo que da versatilidad al generador de señales y el poder usarlo en infinidad de aplicaciones y situaciones. El control del dispositivo se podrá realizar de dos modos distintos. El primero es un método directo, por medio de un control o mando, como pueden ser los botones o los encoder, y un modo de visualización, como podría ser una pantalla o un LCD. El segundo es un modo indirecto, que se realizará con un programa para ordenador que nos permitirá comunicarnos con el dispositivo por UART mediante un USB. Ambas comunicaciones las recibirá, por separado, un microcontrolador que las gestionará y se las transmitirá a otro microcontrolador (el del bloque generador de señal) que proporcionará la onda con las características deseadas.

3.3.1. Interfaz directa

El manejo del dispositivo de manera independiente es un aspecto muy importante, ya que proporciona a la herramienta el poder usarla en cualquier sitio sin necesidad de tener que disponer de otros instrumentos. Al hablar de una interfaz directa nos referimos a la forma de manejar el dispositivo de manera independiente. Es lo contrario a un método indirecto, dónde con otro equipo podemos controlar al generador. El método directo es esencial porque da la posibilidad de poder trabajar solamente teniendo el generador de señales, sin depender de otros dispositivos que pueden estar estropeados, o simplemente no estar disponibles.

El uso directo debe poder cumplir el intercambio de la información necesario. Por un lado, debe ser capaz de mostrar los datos referentes a las opciones que se pueden seleccionar. Para este cometido, puede dar buen resultado un LCD. La elección del LCD habrá que tener en cuenta que sea cómodo para visualizar la información a la hora de trabajar con él. Por otro lado, tiene que estar preparado para la entrada de la información que el usuario desea introducir. Hay varias opciones para esta función, pero las más relevantes son los botones y los encoder. De estas dos opciones, la elegida influirá en la programación.

3.3.2. Interfaz remota

La posibilidad de poder controla de forma remota el generador de señales es una característica que, aunque no siendo imprescindible, le da una serie de ventajas que lo hacen muy interesante. Aprovechando la comunicación con el PC, no sólo se podrá elegir las características de la onda, sino que además se podrán añadir varias nuevas. Esta última característica potencia la versatilidad que se busca en este dispositivo.

3.4. Bloque alimentación

Por último, el bloque de alimentación se encargará de proporcionar los voltajes que garanticen el correcto funcionamiento. Los valores necesarios serán los de alimentación de los amplificadores operacionales y el de los microcontroladores. Estos voltajes deben de ser lo más estables posibles, ya que de ello dependerá la precisión de las ondas generadas. Para lograr este objetivo, se usarán reguladores de tensión, por lo que se tendrá en cuenta la caída de tensión que habrá en cada uno de ellos.

Tener la posibilidad de alimentarlo de varias maneras puede ser interesante puesto que, esta placa es un prototipo. También debería tenerse en cuenta el añadir algunas protecciones en

este bloque, sobre todo si consta de varias formas de alimentación. Si se dispone de protecciones evitaríamos daños o averías en el caso de que se produzca alguna mala conexión.

El principal problema que se puede dar al alimentar una placa que está en desarrollo es la polaridad inversa, es decir, conectar de manera incorrecta los cables. Este problema aumenta cuantos más cables de distinto voltaje tengamos porque hay más posibilidades de error. En nuestro caso serán al menos tres, por lo que nos conviene tener alguna protección frente a este tipo de fallos. Existen muchas maneras de proporcionar este tipo de protección, la mayoría se basan en al menos un diodo, otros incluyen, además: fusibles, SRC, relés, MOSFET, etc. Incluso hay integrados que se encargan de solucionar este problema. Cualquiera de los modos descritos es válido como sistema de protección, pero el más indicado lo determinará el precio, ya que no tiene sentido que el sistema de protección sea más caro que el circuito que protegen.

Otro problema que se puede presentar en la alimentación es el sobrevoltaje, una tensión mayor de la que puede soportar el dispositivo. Este tipo de problema se tendría bajo control, al menos hasta un cierto valor, debido a que tendremos reguladores de tensión a la entrada de alimentación. Estos componentes pueden llegar a funcionar, dependiendo del modelo, hasta los $40V$, que es más de lo que vamos a necesitar y de lo que normalmente proporcionan las fuentes de alimentación. Por lo que el uso de reguladores de tensión, además de proporcionarnos un voltaje muy estable, protegen al circuito ante sobrevoltajes.

También existe la posibilidad de que se produzcan cortocircuitos debido a varias causas como: manejar el dispositivo de manera incorrecta, colocar mal algún componente o realizar mal alguna medición. Para estos problemas también existen varias maneras de proteger el dispositivo, la más común es un fusible. El fusible interrumpe la alimentación si la demanda de corriente es superior a la que se espera en un funcionamiento correcto del circuito.

4. Diseño de la solución

Una vez vistos los requisitos, los planteamientos de teóricos y algunas de las soluciones con las que podemos trabajar, pasamos a la parte práctica en la que vamos a realizar una serie de pasos para conseguir construir el generador de señales.

Primero buscaremos que componentes necesitamos, para ellos se realizarán los cálculos pertinentes. Posteriormente, como en este trabajo se usarán microcontroladores, escribiremos los programas necesarios para que realicen el trabajo requerido. Una vez tengamos estos dos primeros pasos completados, procederemos a realizar pruebas con la que comprobemos si todo funciona de la manera esperada. Si el comportamiento es el deseado se diseñará una PCB que se mandará a fabricar. Con la PCB final y los componentes colocados comprobaremos que funciona correctamente y de una manera más precisa que en las pruebas.

4.1. Cálculos previos

En este punto se explican los cálculos que se han realizado para determinar los componentes que son necesarios. Para algunos hemos tenido que realizar algunas pruebas previas.

4.1.1. Bloque generador de señal

Los cálculos principales se basan en obtener la frecuencia máxima deseada, esto depende del microcontrolador que se encarga de producir la señal digital y del número instrucciones que necesita para sacar una muestra. Esto se desarrollará más adelante con más profundidad en la parte de programación. La frecuencia de salida depende de los siguientes factores:

- Número de muestras por onda ($n_{muestras}$).
- Número de instrucciones por muestra (n_{cy}).
- Frecuencia de instrucción (F_{cy}).

$$F_{out} = \frac{F_{cy}}{n_{muestras} * n_{cy}}$$

Si queremos una frecuencia máxima de $200\ kHz$ y al menos 10 muestras, y habiendo hecho pruebas con varios microcontroladores usamos unas 20 instrucciones para cada muestra, necesitaremos como mínimo una $F_{cy,min} = 40\ MHz$.

Buscaremos en el catálogo de Microchip y filtraremos la búsqueda con las otras características antes mencionadas: 16 bits, una puerta con 16 bits, UART para poder comunicarse con el otro micro, 5 voltios, encapsulado THT (Through-Hole Technology), memoria suficiente para guardar varias ondas. Con estos datos nos deja entre otros el dsPIC33ev256gm102, que es el que finalmente se usará.

4.1.2. Bloque adaptación de señal

Empezando por la parte del DAC, los valores de la resistencia pueden estar entre los pocos cientos de ohmios hasta los cientos de miles de ohmios, así que elegiremos un valor intermedio de $10\ k\Omega$ para las resistencias R y $20\ k\Omega$ para las resistencias 2R. Para determinar la potencia de las resistencias solo hay que ver a que voltaje soportarán y la corriente que las atraviesa. Realmente al ser una red de resistencia con un valor tan alto resistivo, la corriente que pase a través de ellas será mínima y por tanto la potencia también lo será. Haciendo algunas simulaciones se obtienen valores entorno al medio milivatio.

Continuamos con el circuito de amplificadores operacionales. En la imagen se puede ver las distintas etapas con las que se busca conseguir los objetivos antes mencionados.

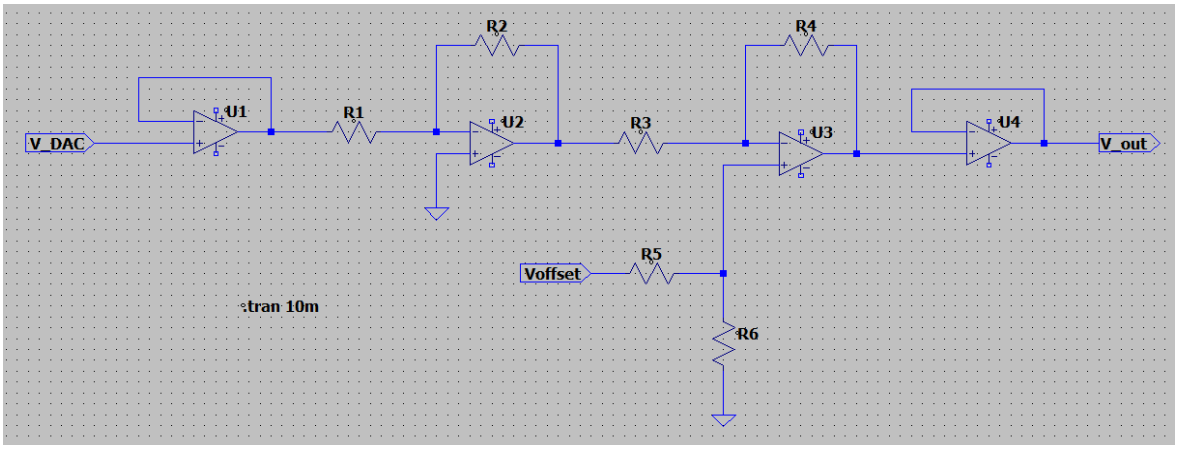

Fig. 6 Circuito de AO.

- U1. Está configurado como seguidor de tensión y se encargará de mantener los valores de la señal que viene del DAC sin que se altere por efecto de las etapas posteriores. No tiene datos que calcular.
- U2. Está configurado como amplificador inversor y tiene como objetivo amplificar la onda desde el mínimo hasta el máximo deseado. Como del microcontrolador vendrá la señal de 0 a 5 V y se quiere amplificar hasta 15 V_{pp}, la amplificación deberá ser de 3. Para evitar que por temas de tolerancia no se pueda llegar a amplificar por 3 la onda, haremos los cálculos para amplificarla por 4 y así ir seguros. Esto se conseguiría al hacer que R2 pueda llegar a ser cuatro veces más grande que R1, la manera que implementaremos será fijando R1 en 20 $k\Omega$ y sustituyendo R2 por un potenciómetro de 100 $k\Omega$. Decidimos que el valor de la resistencia R1 sea 20 $k\Omega$ por la simple razón de que coincide con el valor de las resistencias del DAC, lo que facilitará su compra, como se explicará más adelante.

$$v_{out,U2} = -v_{out,U1} * \frac{R2}{R1}$$

- U3. Cumplirá la tarea de añadir offset a la señal, lo conseguirá mediante una configuración de amplificador restador. La señal que viene de U2 se resta al voltaje V_{offset}. Este voltaje vendrá de un potenciómetro que estará entre los voltajes de alimentación del mismo amplificador, lo que nos permitirá recorrer todo el rango necesario. Los valores de las resistencias R3, R4, R5 y R6 deben tener el mismo valor, pues no buscamos ninguna amplificación, sólo sumar el valor de offset. Un valor de 100 $k\Omega$ es válido.

$$v_{out,U3} = v_{offset} - v_{out,U2}$$

- U4. Para evitar que la onda se deforme ante la carga a la que se conecte, usaremos una etapa de seguidor de tensión, evitando el problema,

*U2 invierte la onda y U3 introduce la onda como sustraendo, por lo que al final la salida no estará invertida.

Si se elige un amplificador cuádruple podríamos tener el circuito completo con solo un integrado, lo que ayudará a que el presupuesto sea ajustado. Independientemente de con

cuantos integrados se forme el circuito los amplificadores U2, U3 y U4 deberán alimentarse con un voltaje dual de al menos $\pm 15\ V$ para que puedan conseguirse que la señal llegue a los valores objetivo. El primer amplificador, al ser un seguidor de tensión, sólo necesitaría alimentarse con un voltaje que permita tener a la salida los 5 voltios. El diseño se simplifica en una manera considerable si todos se alimentan a los mismos voltajes, así que se usarán $\pm 15V$ para los 4 amplificadores.

Con las cuatro etapas propuestas contemplaríamos las necesidades básicas para este bloque y que posiblemente sean suficientes, pero por si fuese necesario añadiremos un filtro paso bajo que elimine la alta frecuencia proveniente de las conmutaciones del bloque generador.

Un filtro paso bajo no realizaría tan bien la función que proporciona el último seguidor de tensión (U4), pero lo haría bastante bien. La opción de combinar ambas con la posibilidad de elegir una frente a otra dependiendo de las necesidades resulta muy interesante, sobre todo porque ayudaría a mantener el número de amplificadores en cuatro (Fig. 7).

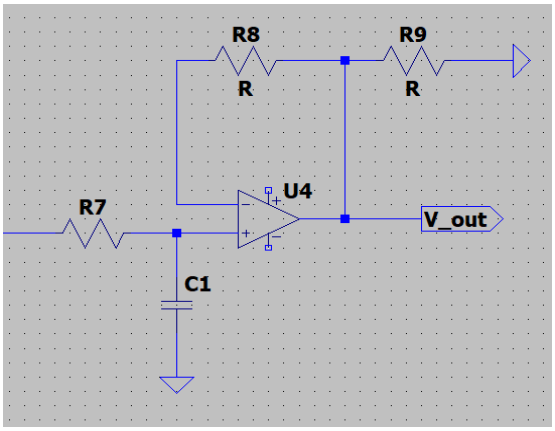

Fig. 7. Circuito AO con filtro pasa bajo.

4.1.3. Bloque interfaz y comunicación

Este módulo más que cálculos tiene algunos requisitos, necesita que tenga UART para poder comunicarse con el microcontrolador del módulo generador de señal y con el conversor UART-USB que se usará para la aplicación en el PC. También I2C para comunicarse con el LCD. Por otro lado, sería muy aconsejable que tuviera interrupciones por flanco o cambio de estado, para poder gestionar de la mejor manera posible el uso del encoder. Todas estas características las podría cumplir perfectamente un Arduino Nano. Una vez visto el microcontrolador necesario, pasamos a ver los elementos que necesitaremos para que el usuario pueda intercambiar la información y controlar el generador de señales.

Empecemos por la manera de visualizar los datos, tras probar los LCD tipo 1602 y 2004 (Fig. 8) elegimos el que más información mostraba. Los dos primeros valores indican los caracteres a lo largo, y los dos últimos, a lo alto. La elección se basa en que el único inconveniente de usarlo sería necesitar que el dispositivo sea lo más pequeño posible, y en nuestro caso, eso no es un problema. Para la conexión LCD-microcontrolador optaremos por un módulo LCD-I2C (Fig. 9).

En cuanto a la entrada de datos, la elección es meramente personal, cualquiera de las dos opciones es igual de válida. Por lo tanto, elegí el uso del encoder simplemente por preferencia personal (Fig. 10).

Fig. 8. LCD 2004.

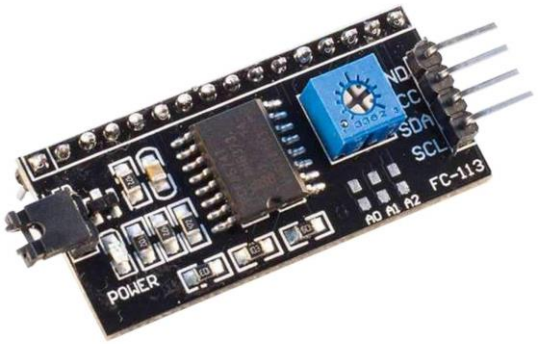

Fig. 9. Módulo LCD-I2C.

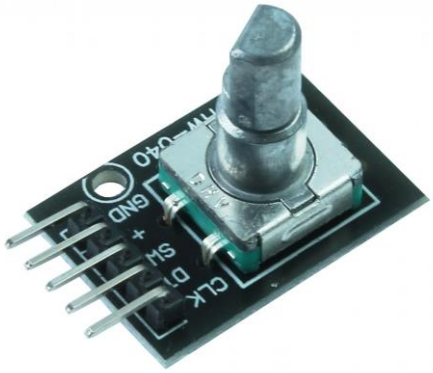

Fig. 10. Enconder HW-040.

4.1.4. Bloque alimentación

Por ser un prototipo, se ha optado por la posibilidad de que sea alimentado de varias maneras, una que nos permita introducir los voltajes con los que finalmente trabaja el dispositivo y otra que sea para voltajes que el mismo dispositivo se encargará de regular a los valores requeridos. Los cálculos solo son necesarios para la segunda manera, ya que en la primera deberá ser el usuario el que introduzca los valores correctos.

Centrándonos en el modo de los reguladores, es interesante recordar los voltajes requeridos: $+5\ V$ para los microcontroladores y $\pm 15\ V$ para los circuitos amplificadores. Esto se traduce en un integrado 7805, un 7815 y un 7915. Necesitaremos algunos voltios más que compensen la caída de voltaje que sucederá en cada uno de ellos. Si miramos las hojas de características de algunos reguladores veremos que lo normal es que tengan una caída cercana a los $2\ V$, por lo que habrá que sumárselo al valor que necesitamos. Por otro lado, hay que tener en cuenta el voltaje máximo que soportan, típicamente es de $30\ V$.

Por último, quedaría por determinar la corriente que suministrarán para poder determinar su encapsulado. Esto lo realizaremos de manera empírica una vez montada una placa de pruebas con todo el circuito.

Buscando en proveedores de componentes electrónicos y filtrando con los datos mencionados, seleccionamos los siguientes componentes:

- L79L15ACUTR. Regulador de voltaje negativo con salida de $-15\,V$, $100\,mA$, $V_{drop} = 1.7V$, $V_{in,\max} = -35\,V$.
- L78M15CDT-TR. Regulador de voltaje positivo con salida de $15\,V$, $500\,mA$, $V_{drop} = 2V$, $V_{in,\max} = 35\,V$.
- L78M05ABDT-TR. Regulador de voltaje positivo con salida de $5\,V$, $500\,mA$, $V_{drop} = 2V$, $V_{in,\max} = 35\,V$.

Por lo tanto, necesitamos unos $\pm 18\,V$de manera que podamos asegurar los $\pm 15\,V$ y por consiguiente no tengamos ningún problema en obtener los +5. La corriente no será un problema ya que se supera de manera holgada la cantidad que necesitamos.

Estos reguladores necesitan de unos condensadores para funcionar de manera correcta. El valor y el tipo de estos condensadores, así como su la situación, vienen indicado en la hoja de características. En la Fig. 11 se muestra un fragmento de la hoja de datos del L79L15ACUTR. En la siguiente imagen, la Fig. 12, hace lo mismo, pero con los reguladores de tensión positivos: L78M15CDT-TR y L78M05ABDT-TR,

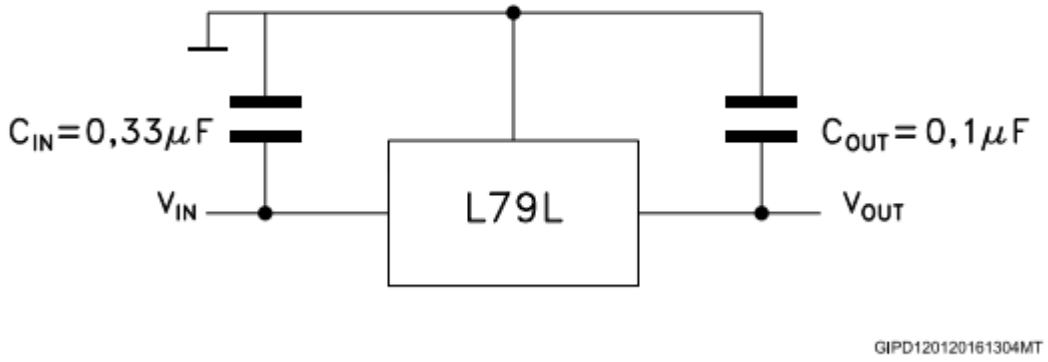

Fig. 11. Regulador de tensión negativo. Configuración.

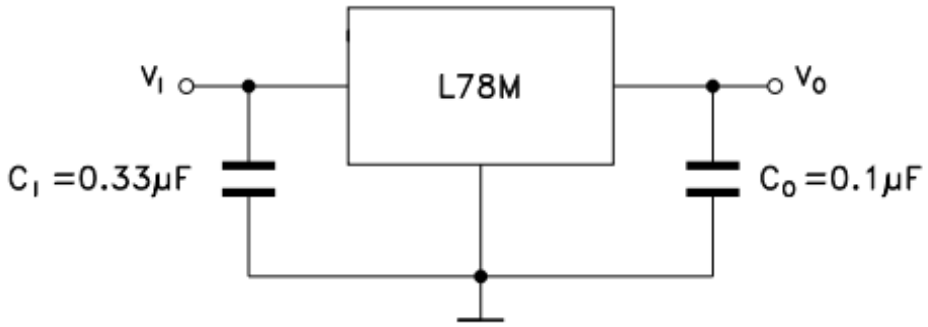

Fig. 12. Regulador de tensión positivo. Configuración.

Para las pruebas se usó como fuente de alimentación un transformador con una salida de $16\,V_{RMS}$, que una vez rectificados y filtrados se convierten en $\pm 21\,V_{DC}$. Con estos valores tendremos suficiente para el correcto funcionamiento de la parte de los reguladores.

Para concluir este bloque tenemos que diseñar y calcular los elementos de protección. El problema de una polaridad inversa lo solucionaremos usando: un diodo en paralelo a la entrada y un fusible en serie. Conectar dos cables de manera incorrecta provocaría que el diodo condujese bajando el voltaje al valor de la caída del diodo, protegiendo así al resto del

circuito. Pero sólo el diodo sería como un cortocircuito, lo que se traduce en un consumo muy elevado de corriente en los cables conectados a la inversa, que provocaría que el diodo terminase destruido. Si el diodo quemado se queda abierto no protegería al circuito, sólo lo haría si se queda en cortocircuito. Pero al tener el fusible esto no llegaría a producirse, ya que bien elegido, se activaría evitando que la corriente subiese tanto como para quemar el diodo.

En nuestro caso, en vez de un fusible normal, se ha optado por una PPTC para no tener que cambiarlo cada vez que la protección se active. Una PPTC es una resistencia que en estado normal tiene un valor muy bajo, pero aumenta hasta comportase como si estuviera abierto al ser atravesada por una corriente superior a su valor de corte. Al activarse la protección, la PPTC no se destruye, por lo que no hay que sustituirlo cada vez.

El consumo de las líneas de $\pm 15\ V$ son de menos de 50 mA, por lo que las PPTC tiene que ser capaz de soportar esa corriente sin activarse el modo de alta resistencia. Cuando se activa esta protección el diodo conduce y, si no queremos que se queme, la PPTC tiene que activarse antes. Por lo tanto, se eligieron los siguientes componentes:

- SMD1206P010TF. PPTC de $60\ V$ y $250\ mA$.
- M7. Diodo de $1\ kV$ y $1\ A$.

Ambos compontes tienen que poder soportar un voltaje más alto del de trabajo normal, para que no se estropeen en caso de que, no solo se conecten en un orden incorrecto, sino que además sea de un valor equivocado. Con que soporten al menos $60\ V$ es suficiente, ya que la protección de sobretensión la realizarán los reguladores de voltaje y solo será hasta los $60\ V$.

El tercer problema que planteamos en el marco teórico fue el de cortocircuitos debidos a varias causas: manejo del dispositivo de manera incorrecta, colocar mal algún componente o realizar mediciones. El fusible, o mejor dicho PPCT, que hemos descrito antes también serviría para este propósito.

4.2. Programación

En este punto se describirán y explicarán los programas que se han escrito para los microcontroladores. Como se dijo en puntos anteriores, un microcontrolador (dsPIC33ev256gm102) se encargará de generar la onda, el bloque generador de señales, y otro gestionará las comunicaciones con el usuario y el microcontrolador de señales. También se expondrá el desarrollo de programa para PC que nos permitirá tanto controlar el dispositivo como añadirle las nuevas ondas.

4.2.1. Bloque generador de señal

En este bloque se alternan dos lenguajes de programación: ensamblador y C. Esto permite, en el caso de ensamblador, tener un control preciso en el flujo de programa, así como del número de instrucciones que se ejecutan, lo que es vital para que la onda que se obtenga tenga los valores deseados de frecuencia. Las ventajas que tiene ensamblador se convierten en inconvenientes a la hora de programar en ese lenguaje la parte de comunicación, es por ello que se usará C para escribir esas partes del código.

El entorno de programación que usé para este bloque fue MPLAB X IDE v5.4, el cual permite trabajar de forma cómoda con el microcontrolador elegido, tanto en ensamblador como en C, ya que tiene editores y compiladores para ambos. Dado que el programa que se le carga al PIC contiene ambos lenguajes, este aspecto es esencial.

El microcontrolador encargado de generar la señal sólo necesita saber el tipo de onda y la frecuencia deseadas. Esta información le llegará desde el microcontrolador de comunicación mediante UART, como se puede ver en la Fig. 13. Una vez recibida dicha información, se usará para saber cuántas muestras dará por ciclo, y con ello el incremento de lectura de la tabla por muestra, y que tipo de onda, lo que implicará ir a la función asociada a la señal que se vaya a generar.

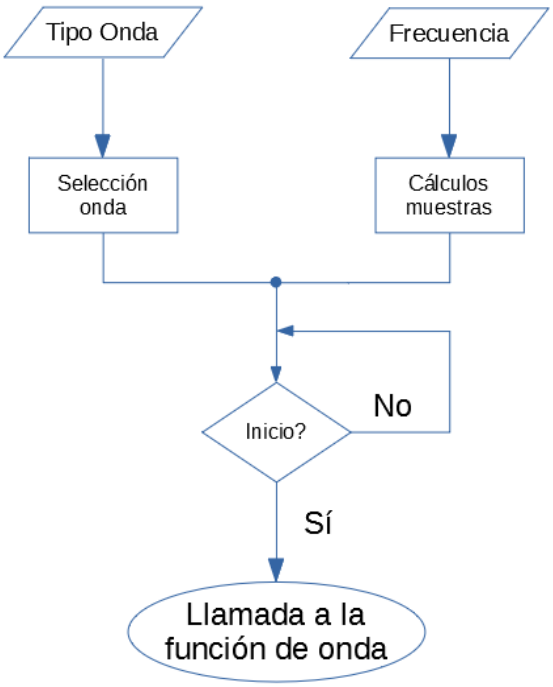

Fig. 13. Diagrama de flujo de la configuración del microcontrolador generador.

El diagrama (Fig. 14) describe el modo de funcionamiento de la formación de ondas que se generan mediante tablas. El programa realiza siempre el mismo trabajo hasta que le pidamos que pare, normalmente para demandarle otra onda. El código se puede ver en el Anexo II. Hay que resaltar que por cómo está implementada y gestionada la memoria de nuestro microcontrolador, tablas de más de 2^{12} posiciones no merecen la pena, porque necesitaríamos instrucciones extra para poder leer toda la tabla.

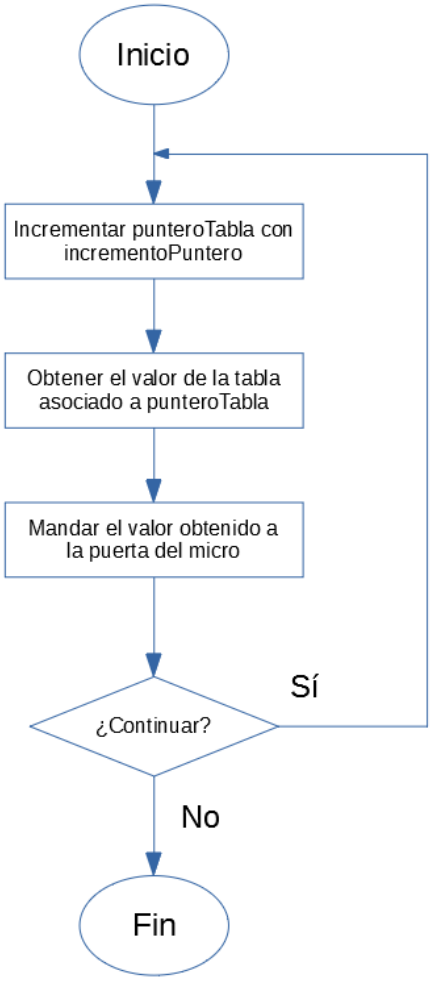

Fig. 14. Diagrama de flujo de la generación de una onda obtenida de tablas.

En nuestro caso particular, $punteroTabla$ se forma con dos registros de 16 bits ($W5{:}\,W4$) y sólo usamos $W5$ para obtener el valor de la tabla. Esto lo hacemos para tener más resolución ya que tenemos en cuenta más decimales. Los $Estados$ pasarían a ser los correspondientes a 32 bits ($2^{32} = 4294967296$) ya que estamos usando dos registros de 16. Por último, tenemos que saber la frecuencia de instrucción (F_{cy}) a la que va a trabajar nuestro microcontrolador que son 70 MHz y el número de instrucciones que necesitamos para sacar una muestra (n_cy) que en este caso son 20. Con estos datos y la siguiente ecuación obtenemos el valor de $incrementoPuntero$ que lo que necesitamos para que la onda deseada tenga la frecuencia requerida.

$$incrementoPuntero = \frac{Estados * Fout * n_{cy}}{F_{cy}}$$

En los casos en los que la onda no se construye mediante tablas, se hará con algoritmos específicos que reducirán el número de instrucciones necesarias por muestra. De esta manera se podrá alcanzar una mayor frecuencia. En la Fig. 15 se puede ver el diagrama de flujo de una onda diente de sierra ascendente y en la Fig. 16 la de una onda triangular.

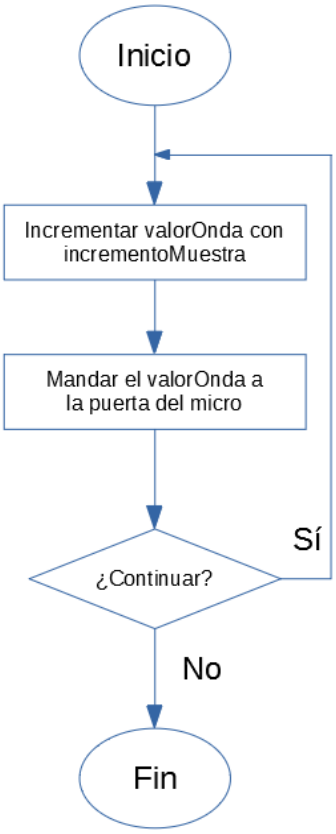

Fig. 15. Diagrama de flujo de la generación de una onda diente de sierra ascendente.

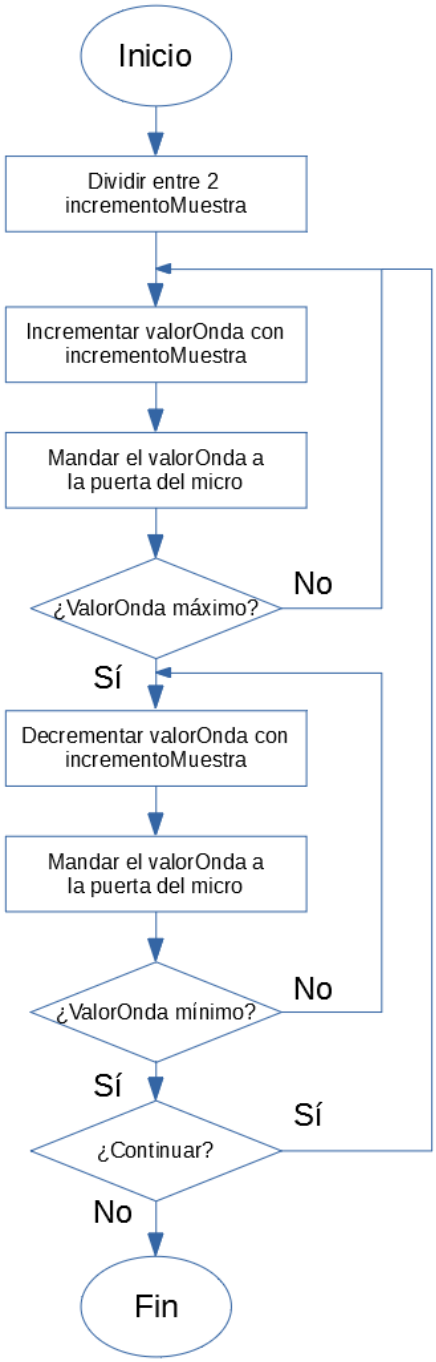

Fig. 16 Diagrama de flujo de la generación de una onda triangular.

4.2.2. Bloque interfaz y comunicaciones

En este bloque veremos el microcontrolador que se va a encargar de las comunicaciones con el bloque generador de señal. Por un lado, recibirá las peticiones que le hagamos desde el método directo mediante el encoder y el LCD. Por otro, se comunicará con el PC mediante un programa que diseñaremos. Estas dos entradas de información se las remitirá al bloque generador de señal.

4.2.2.1. Interfaz directa

Este parte del bloque se programará usando la IDE de Arduino. Se basa en recibir la información del encoder a lo largo de varios estados. En cada uno de ellos podremos elegir el tipo de onda y la frecuencia. Una vez seleccionados se lo enviaremos mediante UART al otro microcontrolador. En la Fig. 17 se puede ver un flujograma de los que acabamos de explicar. Todo el código de este apartado se puede ver en los Anexo III.

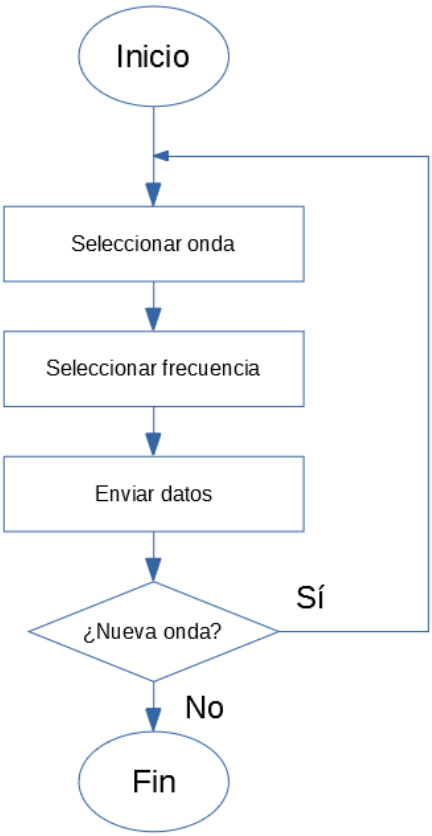

Fig. 17.

4.2.2.2. Interfaz remota

Para esta sección teníamos como objetivo construir un programa para PC que nos permitiese comunicarnos con el generador de señales para poder pedirle las ondas que queramos que genere y la frecuencia a la que oscilen. Para este trabajo usamos Visual Studio y escribimos el código en C#. El diagrama de flujo de este apartado es muy similar al anterior (Fig. 17), puesto que solo necesitamos mandarle la onda que queremos que genere y el valor de frecuencia. En el Anexo III se encuentra el código de este bloque. La interfaz del programa se puede ver en la Fig. 18.

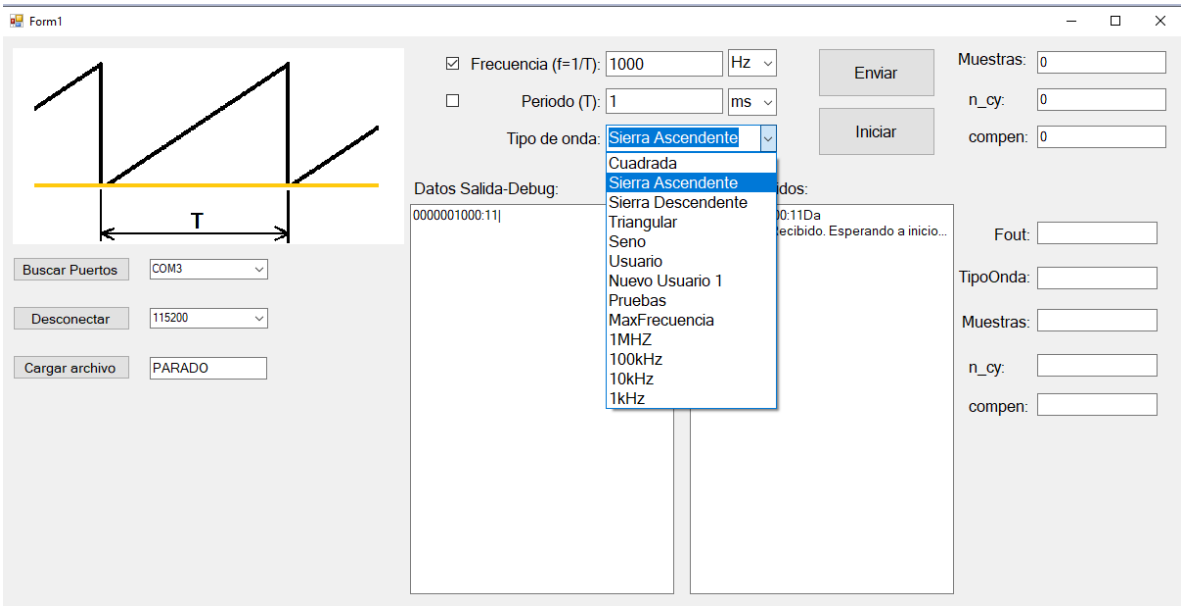

Fig. 18. Interfaz del programa para PC.

Para poder usarlo debemos tener conectada la placa. Cuando la reconozca nuestro ordenador procederemos a buscar el puerto COM que se le haya asignado y lo seleccionaremos. Antes de darle a conectar debemos asegurarnos de que el "baud rate" sea el adecuado, el generador está configurado para trabajar a 115200 baudios. Al conectarnos de manera correcta el botón "enviar" se activará permitiéndonos empezar a mandar las peticiones de onda que estén seleccionadas en las pestañas del centro.

Los dos cuadros de texto que se encuentran en mitad de la aplicación son para llevar un control tanto de lo que se envía como de lo que se recibe, pues el generador devolverá algunos datos que nos permitirán saber si todo va como debería. Hay que recordar que la placa es un prototipo y por consiguiente este programa para PC también lo es, así que toda información que nos ayude a hacer una depuración de lo que hemos programado es muy útil.

Si continuamos describiendo los elementos de la ventana, nos encontramos con los cuadros de texto del lado derecha que también se pueden usar para enviar más información a la placa y ver la que nos devuelve, aunque en la última versión no se llegan a utilizar. Por último, queda por explicar la utilidad del botón "cargar archivo". Su finalidad es la de reescribir la onda usuario por otra que queramos.

Podemos destacar en este programa la facilidad para elegir tanto la onda como la frecuencia. En el primer caso al pinchar en la pestaña de "tipo de onda" se abre un desplegable que nos deja elegir una onda entre las que dispone. Para seleccionar la frecuencia podemos escribir el valor y añadirle la unidad. El recuadro del periodo nos calcula, con la frecuencia que hemos seleccionado, el periodo que debería tener de la onda que le hemos pedido.

4.3. Pruebas prácticas

Durante las pruebas prácticas se mantienen los bloques con los que hemos ido trabajando hasta ahora, pero para comprobar el funcionamiento es necesario mezclarlos. Por ejemplo, es más problemático comprobar el correcto funcionamiento el bloque generador de señal sin el DAC que con él. Por ello la estructura de bloques con la que trabajábamos hasta ahora será modificada ligeramente.

4.3.1. Bloque generador de señal y DAC

Inicialmente empecé trabajando en el bloque generador de señales programando y probando ondas sencillas como la cuadrada o diente de sierra. En este punto comprobé cual era el máximo de frecuencia que podía alcanzar, la cual superaba el objetivo impuesto. Durante los primeros días trabajé con una protoboard y un DAC de 8 bits montado sobre una placa perforada, como se puede ver en la Fig. 19 y Fig. 20.

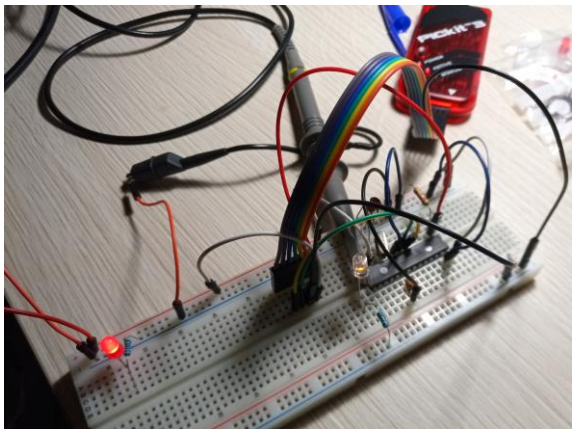

Fig. 19. Circuito de pruebas en una protoboard.

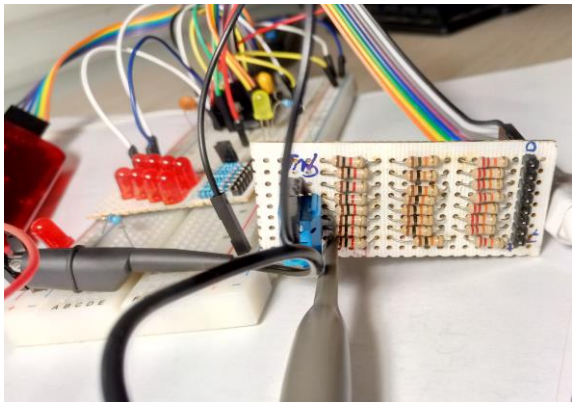

Fig. 20. DAC de 8 bits.

Durante esta primera etapa del proyecto estuve trabajando con un osciloscopio analógico cuya precisión distaba mucho de la deseada para un trabajo como es el diseñar un generador de señales. El inconveniente principal fue el cálculo de la frecuencia, dado que queremos ajustar ese valor lo mejor posible. El trazo en el osciloscopio analógico no permitía saber con exactitud la verdadera frecuencia que estaba dando el prototipo. No fue posible, hasta conseguir un osciloscopio digital, determinar si los cálculos teóricos se cumplían.

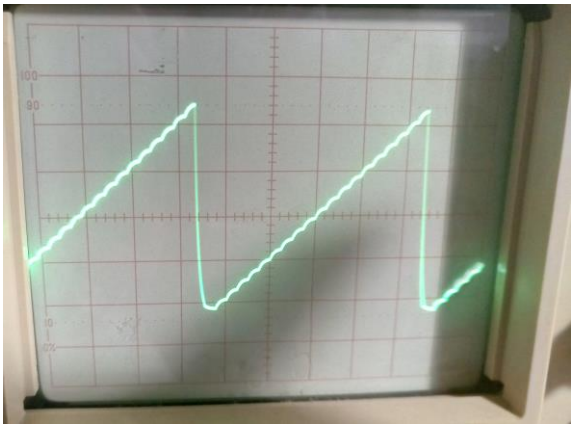

Fig. 21. Onda diente de sierra ascendente con DAC de 8 bits.

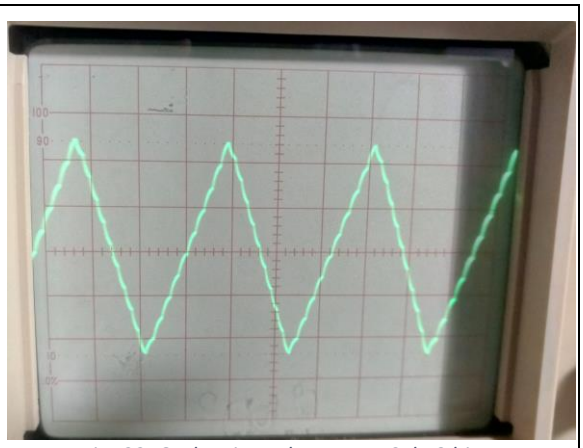

Fig. 22. Onda triangular con DAC de 8 bits.

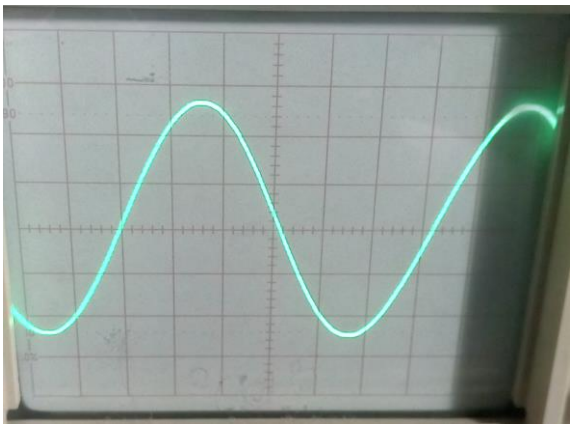

Fig. 23. Onda seno con DAC de 8 bits.

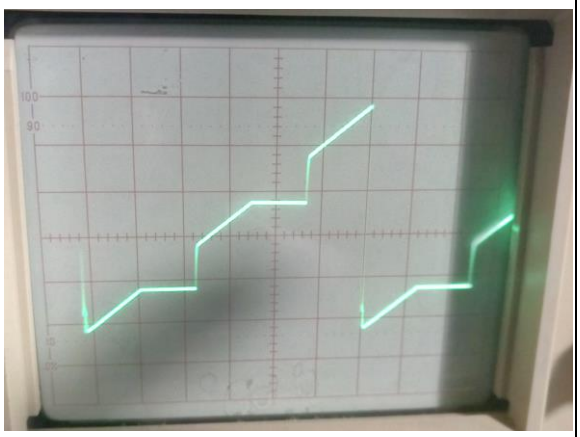

Fig. 24. Onda introducida por el usuario con DAC de 8 bits.

Una vez comprobado el funcionamiento, diseñé y fabriqué un prototipo para el microcontrolador y otra para el DAC, esta vez de 16 bits, como se puede ver en la Fig. 25. Desde entonces estuve trabajando con ambas placas el resto del tiempo de pruebas, hasta pedir la PCB completa. Esto facilitó enormemente el trabajo al no tener tantos cables sin fijar que pudieran dar problemas. No obstante, las resistencias usadas en el DAC eran al 5% de tolerancia, lo que provocaba que las señales que producía tuvieran defectos. Estos desperfectos en las ondas se pueden apreciar sobre todo en torno a los $2{,}5\ V$ (a la mitad de la altura de la onda), que es el momento en el que conmuta el bit de más peso. En menor media, los siguientes bits también acusan esta complicación.

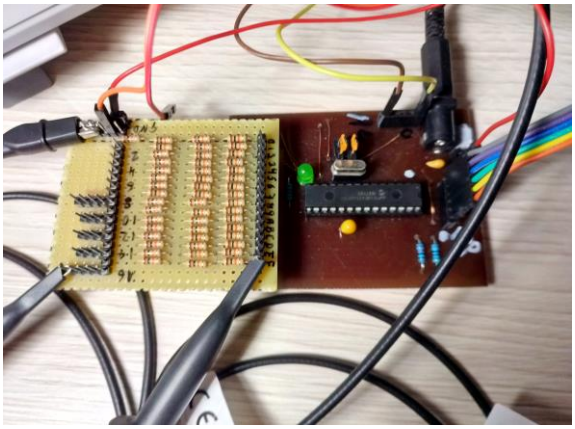

Fig. 25. PCB prototipo del bloque generador y DAC de 16 bits.

El problema mencionado lo podemos apreciar en las siguientes figuras. En la versión final se usarán resistencias de baja tolerancia, por lo que esperamos que este problema se vea resuelto, o al menos minimizarlo. Para ese entonces ya conseguí un osciloscopio digital, el cual me facilitó enormemente el trabajo, principalmente en el aspecto de la frecuencia, tanto por la precisión como en la rapidez de conocer su valor, ya que se muestra en la pantalla junto con la forma de la onda.

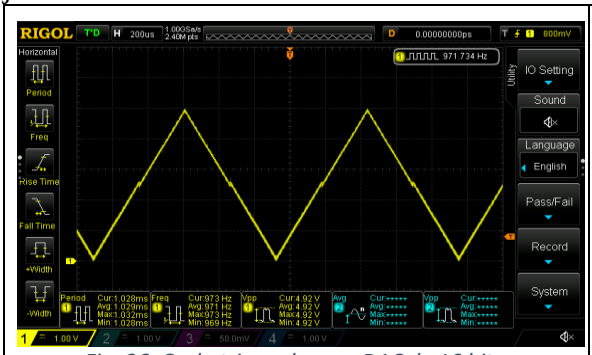

Fig. 26. Onda triangular con DAC de 16 bits.

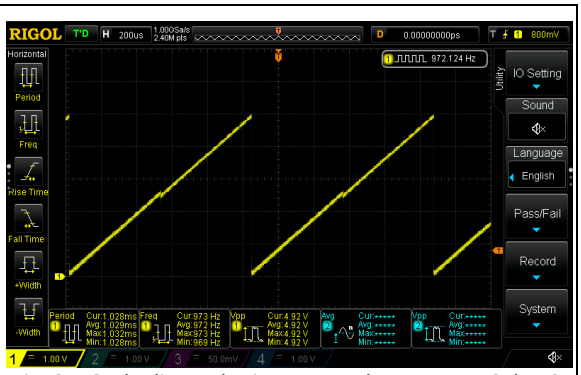

Fig. 27. Onda diente de sierra ascendente con DAC de 16 bits.

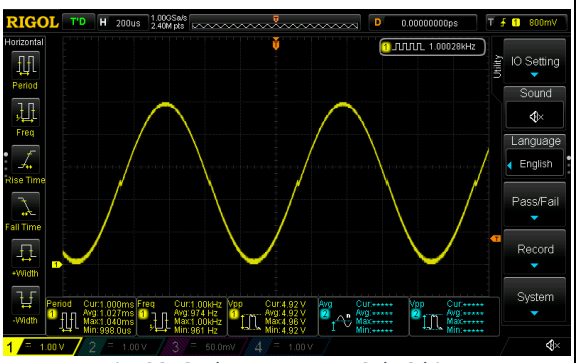

Fig. 28. Onda seno con DAC de 8 bits.

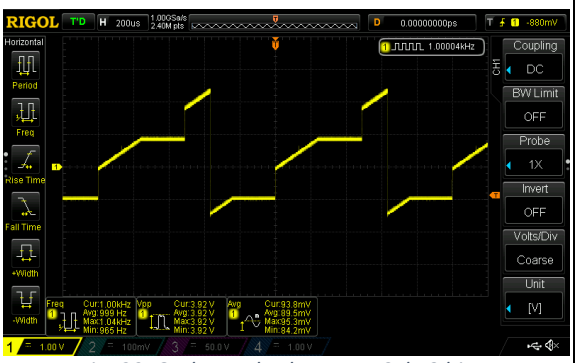

Fig. 29. Onda cuadrada con DAC de 8 bits.

4.3.2. Bloque adaptación de señal: circuito AO

Con una versión del programa que generaba ondas y la placa prototipo, pasamos a trabajar en la segunda parte del bloque adaptación de señal, el circuito de AO. Para las pruebas usamos dos TL072 que, a bajas frecuencias y en ondas que no tuvieran cambios muy bruscos, funcionaba bien. Los problemas llegaban cuando sobrepasábamos los 100 kHz y/o la señal proveniente del DAC presentaba cambios muy grandes en poco tiempo, el bajo slew rate (según la hoja de datos 13 $V/\mu s$) provocaba que la señal se deformase como se puede ver en las figuras de las siguientes figuras. Entonces probé con otro modelo de amplificador, el TLE2142, con un slew rate más alto (según la hoja de datos 35 V/μ) y lo comparé con el TL072 a varias frecuencias.

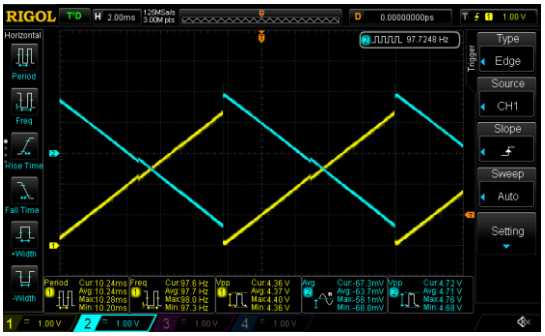

Fig. 30. TL072 100Hz.

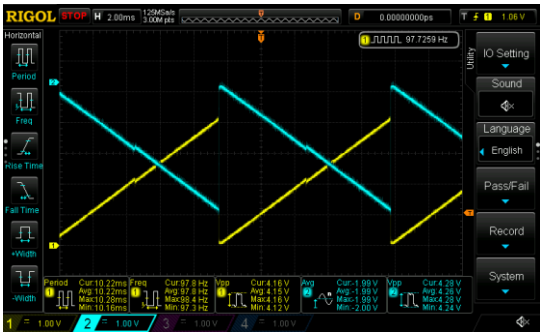

Fig. 31. TLE2142 100Hz.

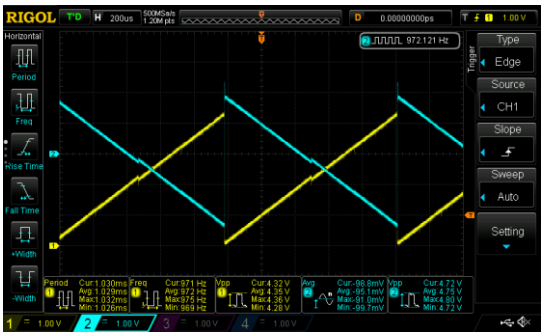

Fig. 32. TL072 1kHz.

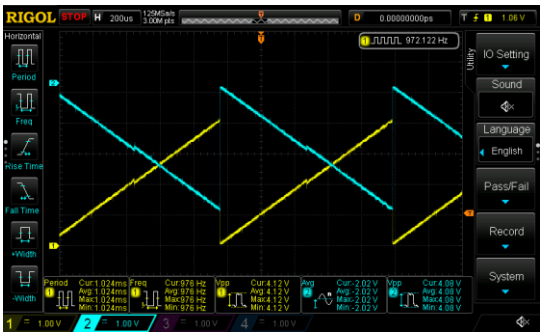

Fig. 33. TLE2142 1kHz.

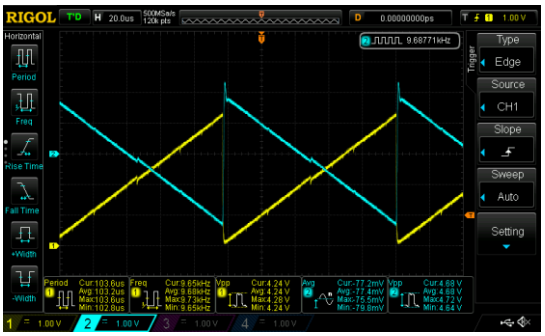

Fig. 34. TL072 10kHz.

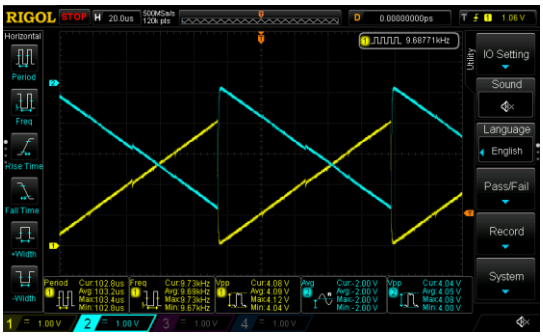

Fig. 35. TLE2142 10kHz.

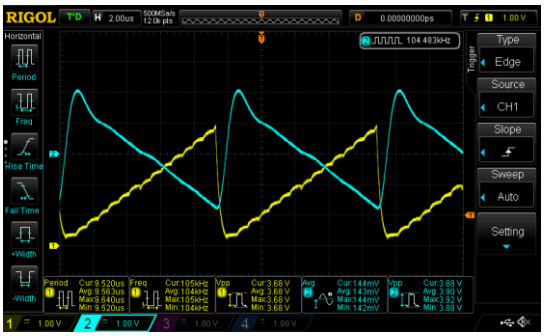

Fig. 36. TL072 100kHz.

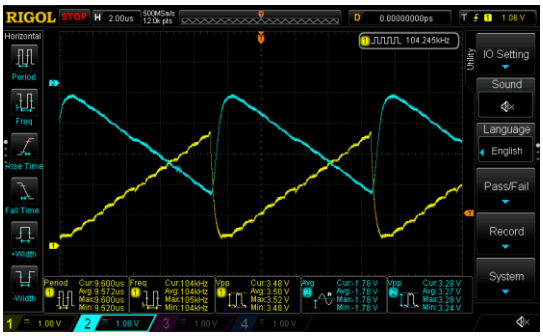

Fig. 37. TLE2142 100kHz.

La configuración del amplificador que se ve en la tabla es la de amplificador inversor con el canal 1 (amarillo) siendo la salida del DAC y en el canal 2 (azul) la salida del segundo amplificador. La ganancia teóricamente es unitaria, pero que, por efecto de la tolerancia de las resistencias, en la práctica resulta algo mayor que uno en el TL072 y menor en el TLE2142. Pero lo interesante es ver cómo, ya a unos 5 V, la señal con el TL072 al pasar del estado más alto al más bajo no es capaz de seguir bien a la onda que sale del DAC. Este

problema se acrecentará a medida que suba la frecuencia o la amplitud de la onda. Es por ello que para la versión final usaremos un amplificador con el slew rate alto, concretamente elegiremos el ISL55004. Este amplificador tiene un slew rate de $300\ V/\mu s$ y un ancho de banda de $200\ MHz$, por lo que no nos hará cuello de botella en ninguno de los aspectos.

Las pruebas del circuito de amplificadores las realicé sobre una protoboard, usando una fuente doble de $\pm 15\ V$ (Fig. 38), que será el voltaje que se usará en la versión final. Las configuraciones probadas fueron varias, pero la que mejor resultado daba fue primero un seguidor de tensión, seguido de un amplificador inversor y de un restador, finalizando con otro seguidor de tensión. El orden amplificador inversor seguido del amplificador restador importa porque, si primero añadimos el offset y luego amplificamos, el offset también lo haría. Esto sería, en principio, un comportamiento indeseado, por lo que usaremos el orden amplificador seguido de restador.

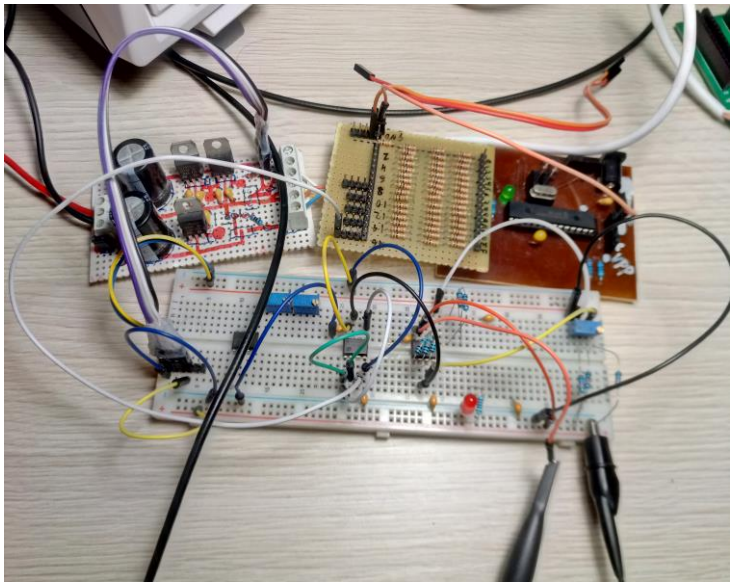

Fig. 38. Placa de pruebas junto con la fuente de alimentación

Pero esta configuración tiene un inconveniente, al amplificar primero la onda, y esta tener un valor de 0 a 5V, no podría llegar hasta los $15\ V_{pp}$ que teníamos como objetivo, porque llegaría al límite que puede dar el amplificador. Para evitar esto se puede buscar un amplificador que tenga un rango de alimentación más amplio, pero como se verá más adelante, el precio nos hará tener que elegir sobre que aspecto nos decantaremos: que podamos amplificar hasta los $15\ V_{pp}$ o que tenga un precio comedido. Creo que es más importante un manejo sencillo y cómodo quedándonos cerca del objetivo de $15\ V_{pp}$, que llegar al objetivo y a la hora de fijar la amplitud y el offset, tengamos dificultades para hacerlo.

Durante las pruebas experimentamos con otras configuraciones, la más reseñable implicaba añadir otro amplificador que se ocupe de quitarle a la onda su componente continua para que pasase a ser completamente alterna. El problema de este añadido es que obliga a usar otro amplificador que excedería los cuatro que vienen normalmente en los integrados y encarecería el dispositivo. De todas formas, al usar el amplificador restador, nos sirve para quitar el offset. Existe la opción de usar un condensador para eliminar la componente continua, pero también provoca que se filtre la señal amortiguando los picos, lo que llega a ser contraproducente. Por lo tanto, optamos por usar solo el amplificador operacional en modo restador para reducir costes sin que afecte significativamente a las prestaciones.

Otro aspecto que se probó fue el de filtrar la alta frecuencia para mejorar la señal, pero en las pruebas no pareció necesario, la misma placa hace de filtro paso bajo (capacidades entre pistas) y los cables (inductancia de las pistas) filtran la alta frecuencia y al menos en las pruebas ese efecto suficiente. No obstante, en el diseño de la PCB final se añadirán los componentes de un filtro paso bajo para, de ser necesario, poder soldarlos.

4.3.3. Bloque interfaz y comunicación

Las pruebas de este bloque se basaron en comprobar que la comunicación funcionaba como debería. Inicialmente probé mediante Termite, un terminal RS232/UART, que es un programa que permite comunicarnos con buses UART. Usando un adaptador UART-USB (Fig. 39) pude mandar, al microcontrolador generador, los datos que enviaría el microcontrolador de interfaz. Una vez verificado que ante los comandos enviados respondía correctamente, pasé a trabajar directamente con el programa de control desde el PC.

En cuanto al control directo, al usar el encoder encontré el problema de los rebotes al cambiar de estado cuando lo giramos, para corregir esta cuestión añadí unos condensadores que eliminaron los rebotes.

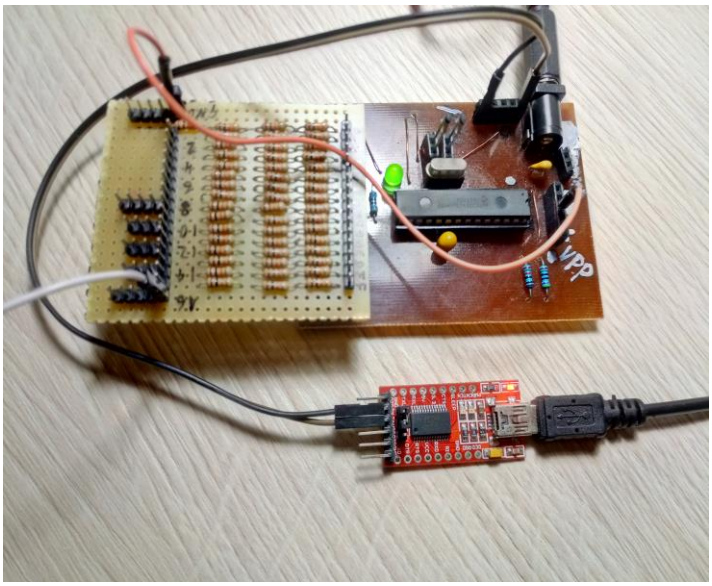

Fig. 39. Placa de pruebas junto con el adaptador UART-USB

4.3.4. Bloque de alimentación

Las pruebas de alimentación se realizaron al construir la fuente con la que se alimentó la placa de pruebas. Al tener todos los bloques conectados y funcionando, se midió la corriente que demandaba, unos 40 mA para las líneas de $\pm 15\ V$ y cerca de 70 mA para los 5 V. Con estos datos comprobamos que los reguladores de tensión elegidos cumplen sobradamente.

4.4. Diseño PCB

Una vez comprobada la viabilidad del diseño, comencé a diseñar la PCB final. Para este trabajo usé el programa KICAD, que permite construir esquemático, y con él, el "layout" de la PCB. Luego de este último archivo se podrán sacar los ficheros necesarios para poder construir la placa, ya sea de manera casera, o mandándola a fabricar a una empresa. Esta última opción será la que elijamos. Para el diseño del esquemático no es necesario saber

cómo se va a terminar fabricando, aunque no estaría de más. Es imprescindible saber cómo se va a fabricar en la fase de del "layout". Esta información es necesaria porque el método de fabricación, así como el fabricante, nos dará las limitaciones que impondremos al diseño para que se puedan fabricar (ancho de pista mínimo, separación entre pistas mínimo, numero de capas, diámetro mínimo de vía, diámetro mínimo de taladro, etc.).

4.4.1. Esquemático

En nuestro caso la PCB se va a mandar a fabricar, así que antes de empezar a diseñar la placa decidí que empresa la construiría, y que limitaciones de fabricación tendría. La elegida fue JLCPCB, porque tiene unos precios que se ajustan al presupuesto y además tiene un servicio de montaje de componentes SMD [4]. Este servicio dispone de un listado de componentes que pueden montar ellos, están divididos en básicos y extendidos. Los primeros son los componentes con las características más comunes, por lo que los tienen en la línea de montaje preparados, y solo tienen un coste por unidad. Los segundos son componentes que no son tan comunes, para montarlos en la PCB tienen que cargarlos en la máquina *Pick-and-place*[2]. Esto repercute en el precio, ya que además del coste por unidad, tienen otro por cada modelo distinto que se solicite. Por lo tanto, al elegir los componentes intentaremos que, en la medida de lo posible, sean del tipo básico y que sean del mismo modelo.

En el diseño del esquemático se tuvieron en consideración las indicaciones que se detallaron tanto en el marco teórico como en las pruebas prácticas. Los planos del esquemático se pueden ver en el Anexo I.

4.4.2. Layout

Una vez completado el esquemático continuamos con el diseño del "layout". El "layout" es el archivo que contiene la posición de las huellas de los componentes en la PCB, así como las pistas que conectan dichos componentes. Además, al trabajar en el "layout", conseguimos generar capas de máscara de soldadura. También incluye otra información correspondiente a la serigrafía para facilitar tanto el montaje de la placa como su posterior análisis de funcionamiento. Facilitando así la implementación y diseño del sistema.

Para empezar, configuramos las opciones de producción del fabricante que podemos ver en su página web [5]. Las opciones más importantes que sacamos son:

Límites de fabricación	Valor
Diámetro de taladro	0,20 *mm* - 6,30 *mm*
Diámetro mínimo del agujero de la vía	0,2 *mm*
Diámetro mínimo de la vía	0,45 *mm*
Distancia mínima entre agujeros	0,5 *mm*
Distancia mínima entre vías (de la misma red)	0,254 *mm*
Distancia mínima entre pads	0,127 *mm*
Ancho mínimo de pista	0,127 *mm*
Distancia mínima entre pistas	0,127 *mm*

[2] *Pick-and-place*: Máquina con la que se colocan los componentes en la cadena de producción.

Una vez introducidas estas restricciones de fabricación en el diseño, nos facilita el trabajo y asegura que el fabricante pueda producir la placa correctamente. Durante la construcción del "layout", al haber introducido los parámetros anteriores, el programa no nos dejará colocar componentes ni trazar pistas que entren en conflicto con la configuración establecida. Una vez finalizado el "layout", realizaremos una última comprobación de posibles errores entre los cuales, se revisará de forma automática que los parámetros de fabricación se cumplen. También se verificarán con el programa que la relación entre componentes y el esquemático es correcta. Las capas del "layout" se pueden ver en el Anexo I.

Por último, a la hora de pedir la PCB al fabricante se eligieron una serie de opciones de las cuales podemos resaltar las siguientes:

Opciones de fabricación	Valor
Capas	2
Grosor del PCB	1,6 *mm*
Color del PCB	Verde
Acabado superficial	LeadFree HASL-RoHS
Cobre	1 oz
Comprobación	Completa
Tipo de material	FR4-Standard Tg 130-140C

4.5. Presupuesto y lista de componentes

A la hora de hablar del presupuesto podemos hacerlo de dos formas. La primera, es cuánto costó llevar a cabo el proyecto y la segunda, por cuánto dinero se podría fabricar una sola palca. El presupuesto de desarrollo no será 100% real, ya que algunos materiales y componentes usados los tenía en casa desde hace tiempo y es complicado saber cuál fue el coste real de los mismo. No obstante, se buscarán y añadirán en las listas como si hubieran sido comprados de manera que podamos hacernos una idea del costo completo. Ambos presupuestos están referenciados a fecha de 1 de diciembre de 2020. Hay que resaltar que debido a que en los distribuidores de componentes electrónicos los LCD tenían un precio muy elevado decidimos comprarlo por Amazon junto con el módulo I2C a un precio de 8 €.

La lista completa por cada PCB de componentes es:

Referencia	Valor	cant
A1	Arduino_Nano_v3.x	1
C1, C2, C14	33u50v	3
C3, C5, C7	0.33u	3
C4, C6, C8-C10, C12, C20-C23, C31, C32	100n	12
C11, C13	27p	2
C19	56p	1

C30	10u tantalum	1
D1	LED	1
D2, D3	diodo 1A	2
F1, F2	Fuse 60V 250mA	2
H1-H4	MountingHole_Pad	4
J1	Barrel_Jack_Switch-36V	1
J2	Conn_01x02_Male	1
J4	Encoder	1
J5	nano-1_15	1
J6	ICSP	1
J7	Pot 100k	1
J8	Pot 100k	1
J3, J9	I2C	2
J10	nano_16-30	1
J11-J14	Conn_01x03_Male	4
J17, J19	Conn_Coaxial	2
J20	+Vdc, GND, -Vdc	1
J21	+15, +5, GND, -15	1
JP1-JP3	SolderJumper_2_Open	3
R1, R3	1k	2
R5-R8	100k	4
R2, R100-R116	10k	18
R4, R9, R10, R201-R216	20k	19
RV1-RV3	R_POT	3
SW1	SW_Push	1
U1	dsPIC33ev256gm102	1
U2	L78M05	1
U3	L78M15	1
U4	L79M15	1
U5	L79L05_SOT89	1
U6	ISL55004IBZ	1
Y1	4MHz	1
Encoder		1
Potenciómetros		2

4.5.1. Presupuesto del desarrollo

Empezando por el presupuesto del desarrollo hay que tener en cuenta que algunos componentes se compraran junto con el PCB y otros, en distribuidores de compontes electrónicos como Mouser [6]. Esto es así para poder reducir los costes y porque antes de mandar a fabricar la PCB ya se pidieron componentes para las pruebas, algunos de los cuales se usaron también en la placa final.

El pedido a JLCPCB fueron 10 PCB. Al pedir 10 PCB el coste era de (PCB + SMT):

$$€5{,}23 \; + \; €46{,}29 = 52{,}15 \; € \, , \qquad Precio/unidad = 5{,}21 \; €/PCB$$

En la tabla se desglosa la cantidad de cada componente, el precio por unidad y el total para las 10 PCB.

Referencia	Valor	Cantidad	Precio unitario	10 PCB	Precio final
C3, C5, C7	0.33u	3	0,0245 €	30	0,73 €
C4, C6, C8-C10, C12, C20-C23, C31, C32	100n	12	0,0188 €	120	2,24 €
C11,C13	27p	2	0,0121 €	20	0,24 €
C19	56p	1	0,0124 €	10	0,12 €
C30	10u tantalum	1	0,1853 €	10	1,85 €
D2, D3	Diodo 1A	2	0,0096 €	20	0,19 €
F1, F2	Fuse 60V 250mA	2	0,0520 €	20	1,04 €
R1, R3	1k	2	0,0057 €	20	0,11 €
R2, R100-R116	10k	18	0,0311 €	180	5,59 €
R5-R8	100k	4	0,0050 €	40	0,20 €
R4, R9, R10, R201-R216	20k	19	0,0276 €	190	5,24 €
U2	L78M05	1	0,1132 €	10	1,13 €
U3	L78M15	1	0,1056 €	10	1,06 €
U5	L79L05_SOT89	1	0,1167 €	10	1,15 €
SW1	SW_Push	1	0,0185 €	10	0,18 €
					21,09 €

El precio de los componentes es de 21,09 €, pero a este valor hay que sumarle el montaje y el precio de las PCB en sí. Todo esto da como resultado un total de 52,15 €. Por último, le añadiremos los gastos de envío que en mi caso fueron 20 €. Así pues, el coste total asciende a 72,15 €.

A este presupuesto nos faltaría por añadirle los componentes que no tenían en JLCPCB. Estos componentes fueron comprados en Mouser. En este caso no compramos material para montar 10 PCB, sino 4, lo que es más que suficiente para permitirnos estropear alguna en las pruebas. La siguiente tabla contiene los componentes que compramos en Mouser.

Referencia	Valor	cant	valor/u	valor/t
A1	Arduino_Nano_v3.x	1	9,82 €	9,82 €
C1, C2, C14	33u50v	3	0,10 €	0,31 €
D1	LED	1	0,15 €	0,15 €
J2	Conn_01x02_Male	1	0,42 €	0,42 €
J4	Encoder	1	- €	- €
J5	nano-1_15	1	- €	- €
J6	ICSP	1	- €	- €
J7	Pot 100k	1	- €	- €
J8	Pot 100k	1	- €	- €
J3, J9	I2C	2	- €	- €
J10	nano_16-30	1	- €	- €
J11-J14	Conn_01x03_Male	4	- €	- €
J17, J19	Conn_Coaxial	2	- €	- €
J20	+Vdc, GND, -Vdc	1	0,38 €	0,38 €
J21	+15, +5, GND, -15	1	0,44 €	0,44 €
U1	dsPIC33ev256gm102	1	3,18 €	3,18 €
U6	ISL55004IBZ	1	7,62 €	7,62 €
Y1	4MHz	1	0,34 €	0,34 €
Encoder		1	0,64 €	0,64 €
Potenciometros		2	1,22 €	2,44 €
			sin IVA	25,73 €
			con IVA	31,13 €

Las cantidades están con relación a una sola PCB, por lo que hay que multiplicarlas por 4, lo que daría un total de 124,52 €. Superando el valor de 50 €, por lo que el envío es gratuito.

En conclusión, el coste completo asciende a 196.67 €, sin el LCD y el módulo I2C. Con ellos serían 228,67 €, y tendríamos 10 PCB con la mitad de los componentes ya soldados y material para completar 4 placas. Si queremos saber por cuánto nos hubiesen salido las 10 placas basta con multiplicar por 10 los componentes adquiridos en Mouser (311,3 €), sumarlo al coste de JLCPCB (72,15 €) y de los LCD e I2C (80 €) y dividirlo por las 10 PCB completas que tendríamos. Resumiendo $46.34\ €/PCB$.

4.5.2. Presupuesto de una sola PCB

Si se opta por la opción de comprar todos los componentes por un lado y hacer la PCB por otro, el precio de los componentes sería el siguiente. Pongo el ejemplo como si se comprase todo en Mouser:

Referencia	Valor	cant	valor/u	valor/t
A1	Arduino_Nano_v3.x	1	9,82 €	9,82 €
C1, C2, C14	33u50v	3	0,10 €	0,31 €
C3, C5, C7	0.33u	3	0,10 €	0,31 €
C4, C6, C8-C10, C12, C20-C23, C31, C32	100n	12	0,05 €	0,54 €
C11, C13	27p	2	0,09 €	0,17 €
C19	56p	1	0,08 €	0,08 €
C30	10u tantalum	1	0,21 €	0,21 €
D1	LED	1	0,15 €	0,15 €
D2, D3	diodo 1A	2	0,31 €	0,61 €
F1, F2	Fuse 60V 250mA	2	0,09 €	0,17 €
H1-H4	MountingHole_Pad	4	0,00 €	0,00 €
J1	Barrel_Jack_Switch-36V	1	0,00 €	0,00 €
J2	Conn_01x02_Male	1	0,42 €	0,42 €
J4	Encoder	1	0,00 €	0,00 €
J5	nano-1_15	1	0,00 €	0,00 €
J6	ICSP	1	0,00 €	0,00 €
J7	Pot 100k	1	0,00 €	0,00 €
J8	Pot 100k	1	0,00 €	0,00 €
J3, J9	I2C	2	0,00 €	0,00 €
J10	nano_16-30	1	0,00 €	0,00 €
J11-J14	Conn_01x03_Male	4	0,00 €	0,00 €
J17, J19	Conn_Coaxial	2	0,00 €	0,00 €
J20	+Vdc, GND, -Vdc	1	0,38 €	0,38 €
J21	+15, +5, GND, -15	1	0,44 €	0,44 €
JP1-JP3	SolderJumper_2_Open	3	0,00 €	0,00 €
R1, R3	1k	2	0,09 €	0,17 €
R5-R8	100k	4	0,09 €	0,34 €
R2, R100-R116	10k	18	0,06 €	1,08 €

R4, R9, R10, R201-R216	20k	19	0,06 €	1,14 €
RV1-RV3	R_POT	3	1,22 €	3,66 €
SW1	SW_Push	1	0,20 €	0,20 €
U1	dsPIC33ev256gm102	1	3,18 €	3,18 €
U2	L78M05	1	0,47 €	0,47 €
U3	L78M15	1	0,49 €	0,49 €
U4	L79M15	1	0,50 €	0,50 €
U5	L79L05_SOT89	1	0,00 €	0,00 €
U6	ISL55004IBZ	1	7,62 €	7,62 €
Y1	4MHz	1	0,34 €	0,34 €
Encoder		1	0,64 €	0,64 €
Potenciómetros		2	1,22 €	2,44 €
PCB		1	8,00 €	8,00 €
			sin iva	43,87
			con iva	53,08

Al igual que con el otro presupuesto hay que sumarle el precio del LCD y del módulo I2C. Por lo que al final, si queremos hacer una sola placa, nos costaría 61.08 €.

Haciendo un resumen de este bloque en unas pocas palabras. Ambos presupuestos dejan al generador que hemos diseñado en un rango de precio inferior a los que podemos encontrar en tiendas, aunque claro está que estos tienen mejores especificaciones. El primer presupuesto es más abultado, pero entra dentro de la normalidad al estar hablando de construir un prototipo. El segundo tiene el inconveniente de que hay que fabricar entero. Pero esta situación puede verse como algo positivo, ya que para un alumno de electrónica puede ser beneficioso enfrentarse a la tarea de hacer una PCB y soldar los componentes.

4.6. Pruebas y comprobaciones

Una vez montada la PCB final (Fig. 41) procedemos a probarla para ver cómo se comporta. Antes de comenzar las pruebas hay que mencionar que, para conseguir la mayor precisión posible en la frecuencia, se midió la frecuencia real del cristal usado. De esta manera, la tolerancia que tuviese nos afectaría lo menos posible. En nuestro caso suponíamos que trabajábamos con un cristal de 4 MHz con el cuál conseguíamos tener una F_{cy} de 70 MHz, pero que realmente, tras medirlo con el osciloscopio, eran 70002800 Hz. Por lo tanto, se usó ese valor en el código.

Fig. 40. PCB final con componentes soldados por JLCPCB

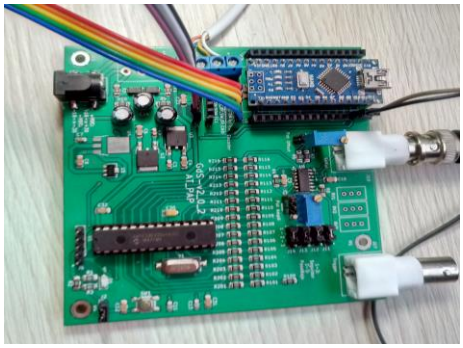

Fig. 41. PCB con casi todos los componentes soldados.

4.6.1. Mediciones de frecuencia

Primero veremos cómo genera las ondas que le pidamos a varias frecuencias. Probaremos con todas las ondas que el generador tiene programadas, con dos consideraciones. La primera es que de las ondas diente de sierra ascendente y descendente solo mostraremos la ascendente porque su comportamiento es idéntico. La segunda es que usaremos la onda seno como representante de la onda usuario porque ambas se generan mediante tablas y el resultado será similar.

Recorreremos todo el espectro para el que lo hemos diseñado, es decir, de 50 Hz a 200 kHz. Como ya sabemos que puede llegar a sobrepasarlo, tanto por arriba como por abajo, también probaremos para frecuencias que se salen del rango y veremos cómo se comporta. Aprovecharemos para ver como atenúa la amplitud, la cual se ha fijado a 6 V_{pp}.

- 1 Hz

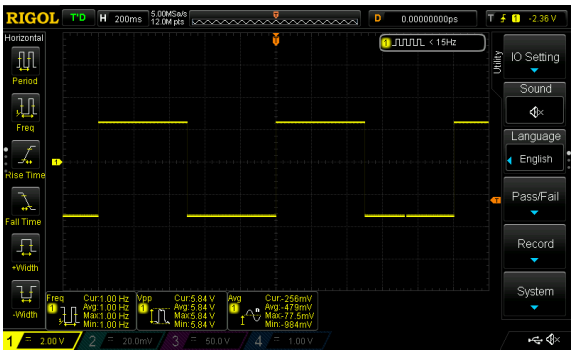

Fig. 42. Onda cuadrada de 1 Hz.

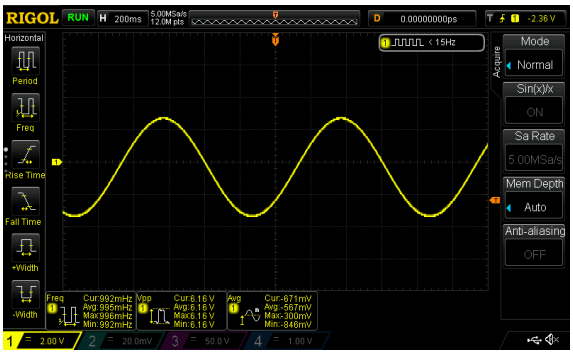

Fig. 43. Onda senoidal de 1 Hz.

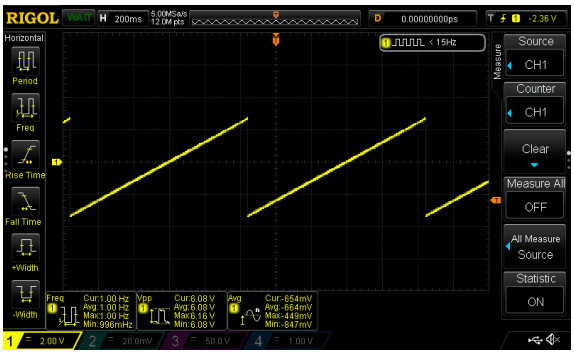

Fig. 44. Onda diente de sierra de 1 Hz.

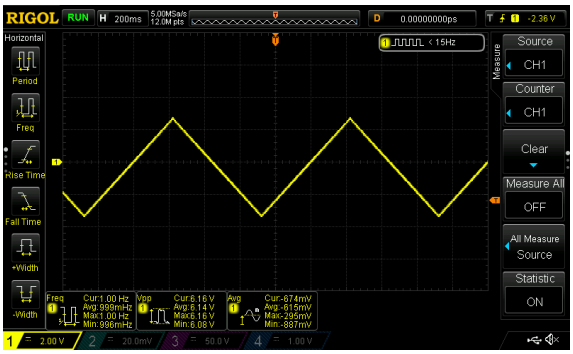

Fig. 45. Onda triangular de 1 Hz.

- 10 Hz

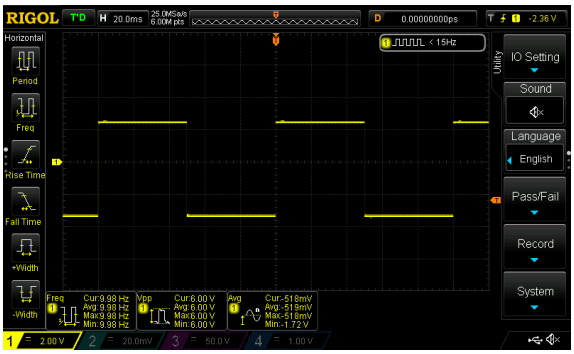

Fig. 46. Onda cuadrada de 10 Hz.

Fig. 47. Onda senoidal de 10 Hz.

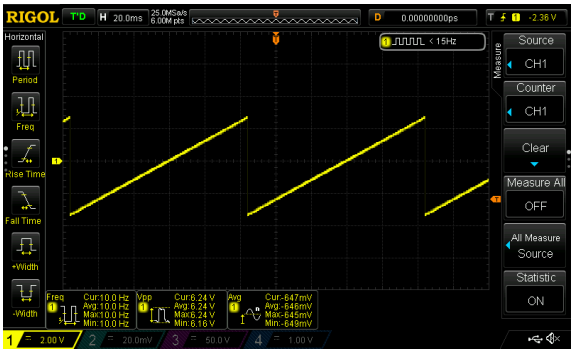

Fig. 48. Onda diente de sierra de 10 Hz.

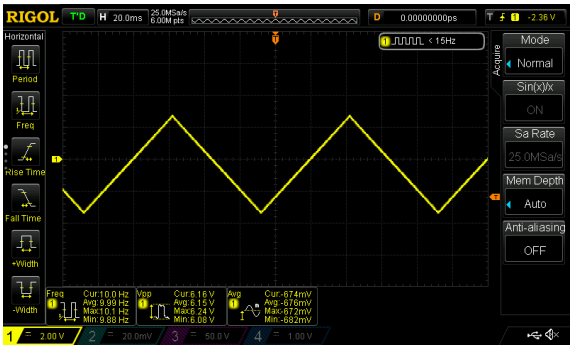

Fig. 49. Onda triangular de 10 Hz.

- 100 Hz

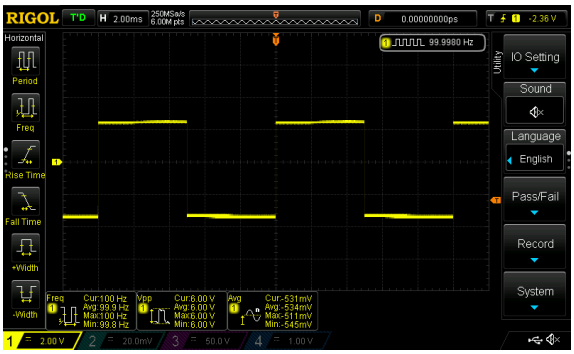

Fig. 50. Onda cuadrada de 100 Hz.

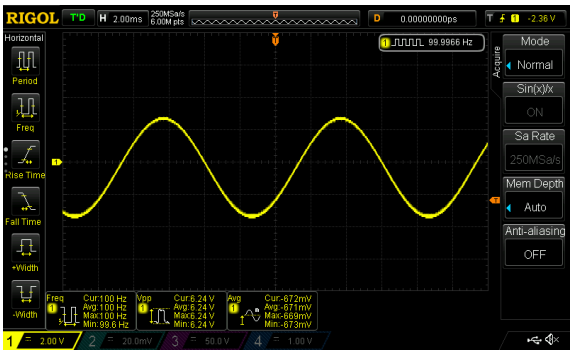

Fig. 51. Onda senoidal de 100 Hz.

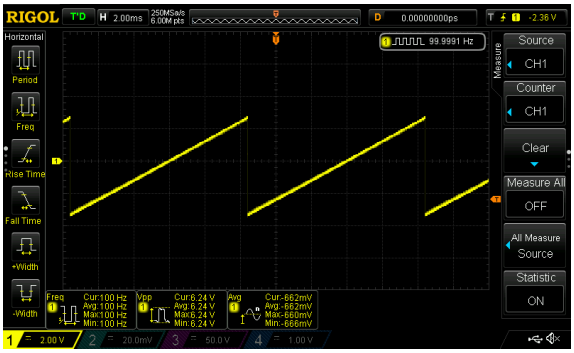

Fig. 52. Onda diente de sierra de 100 Hz.

Fig. 53. Onda triangular de 100 Hz.

- $1\ kHz$

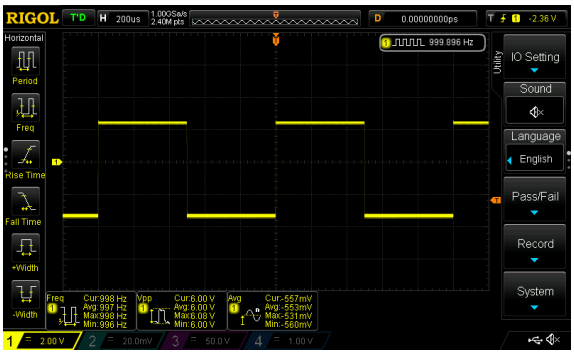

Fig. 54. Onda cuadrada de 1 kHz.

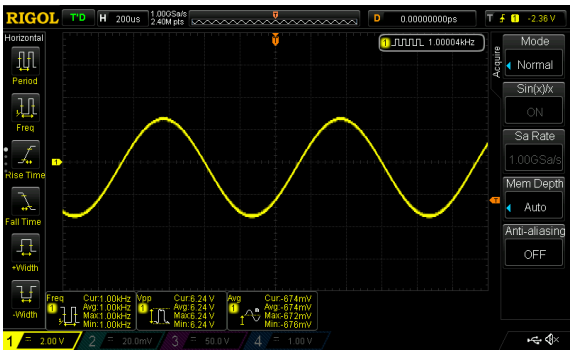

Fig. 55. Onda senoidal de 1 kHz.

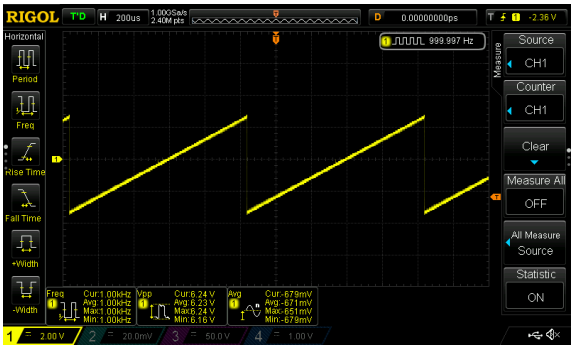

Fig. 56. Onda diente de sierra de 1 kHz.

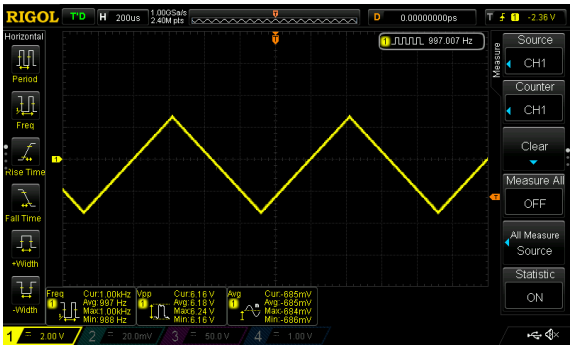

Fig. 57. Onda triangular de 1 kHz.

- $10\ kHz$

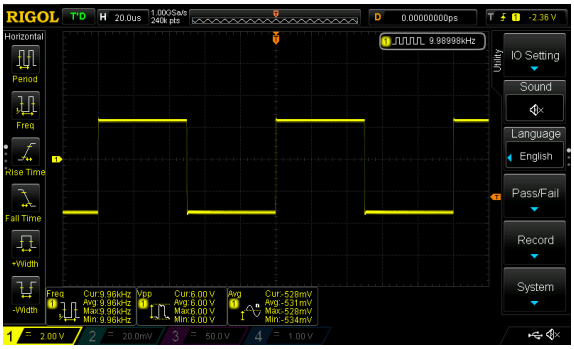

Fig. 58. Onda cuadrada de 10 kHz.

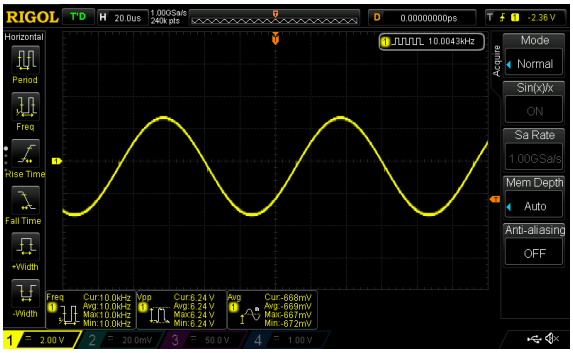

Fig. 59. Onda senoidal de 10 kHz.

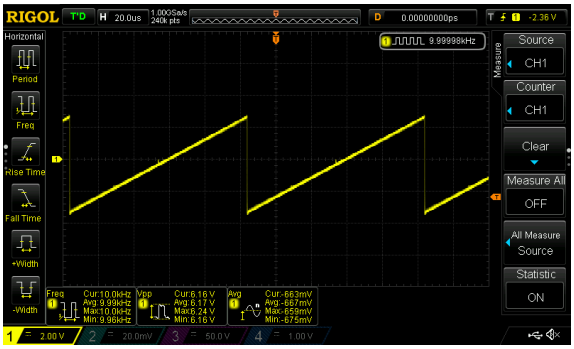

Fig. 60. Onda diente de sierra de 10 kHz.

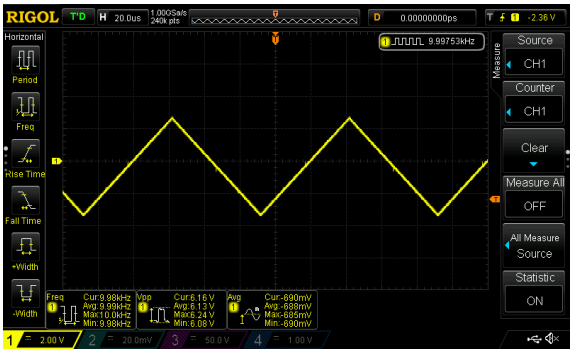

Fig. 61. Onda triangular de 10 kHz.

- 100 kHz

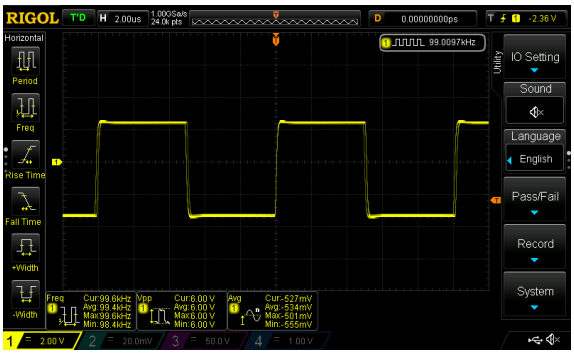

Fig. 62. Onda cuadrada de 100 kHz.

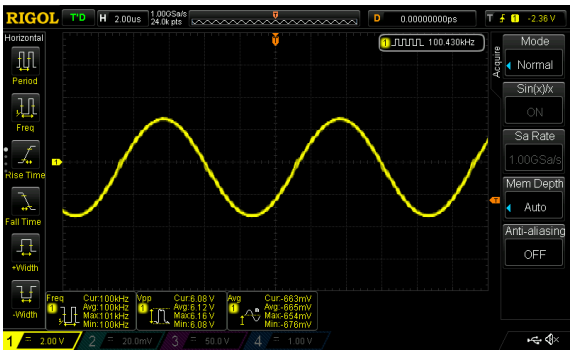

Fig. 63. Onda senoidal de 100 kHz.

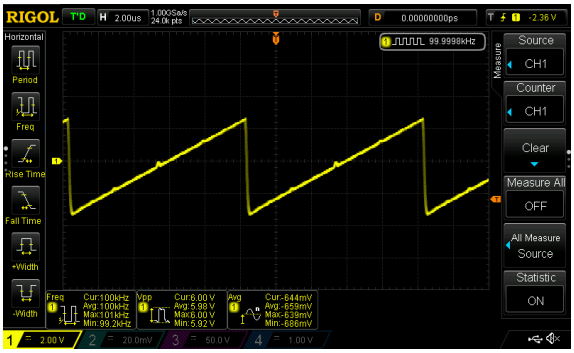

Fig. 64. Onda diente de sierra de 100 kHz.

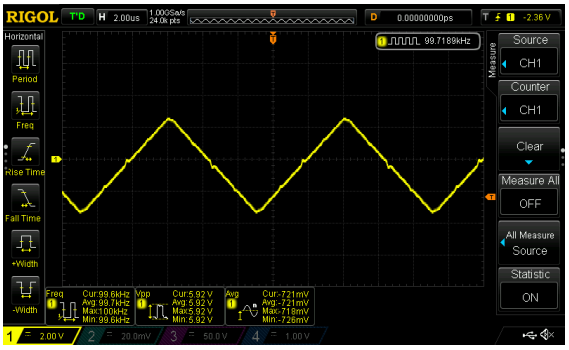

Fig. 65. Onda triangular de 100 kHz.

- 200 kHz

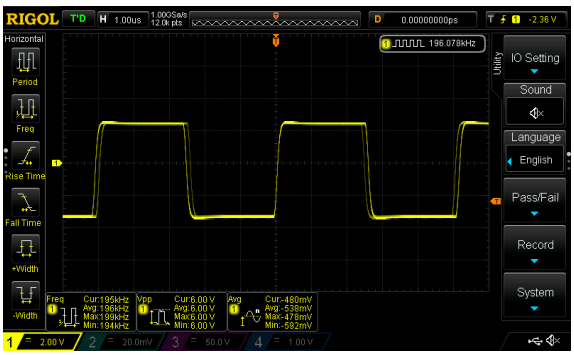

Fig. 66. Onda cuadrada de 200 kHz.

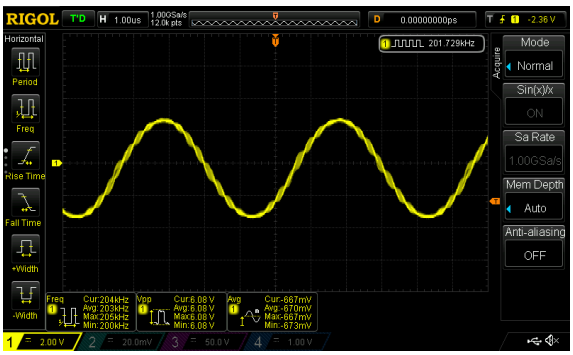

Fig. 67. Onda senoidal de 200 kHz.

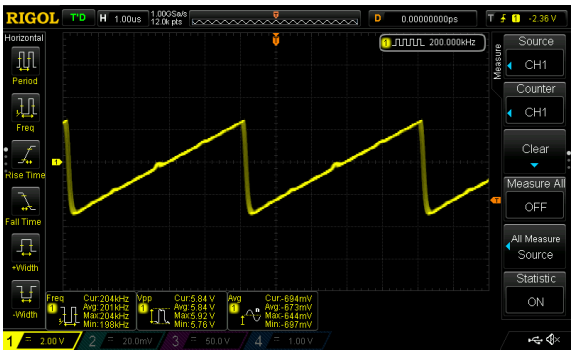

Fig. 68. Onda diente de sierra de 200 kHz.

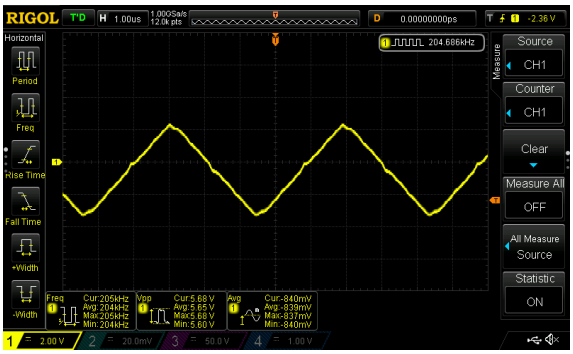

Fig. 69. Onda triangular de 200 kHz.

- 300 kHz

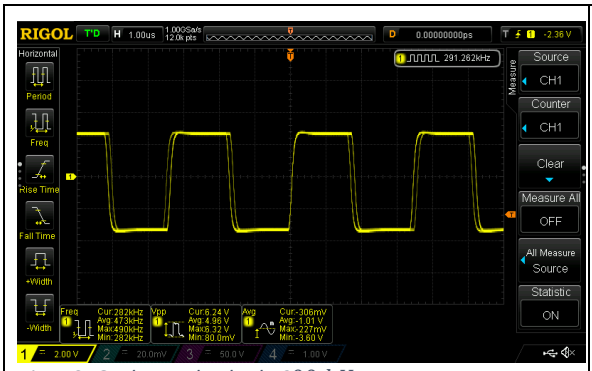

Fig. 70. Onda cuadrada de 300 kHz.

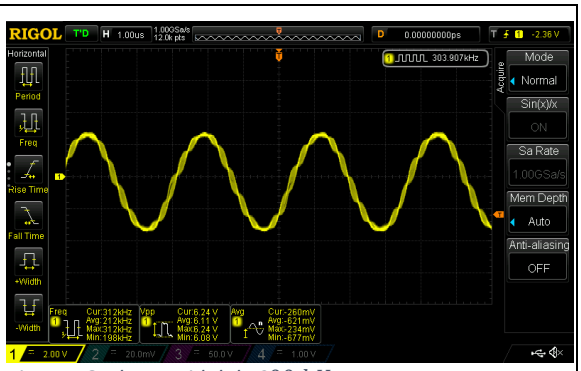

Fig. 71. Onda senoidal de 300 kHz.

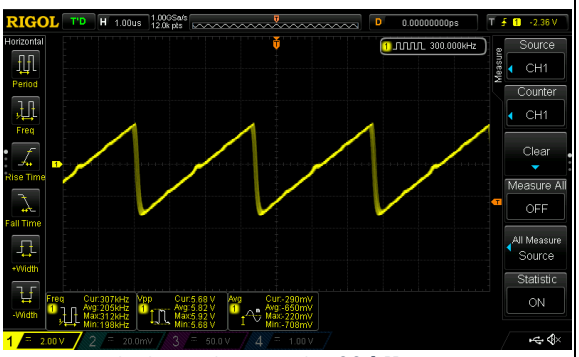

Fig. 72. Onda diente de sierra de 300 kHz.

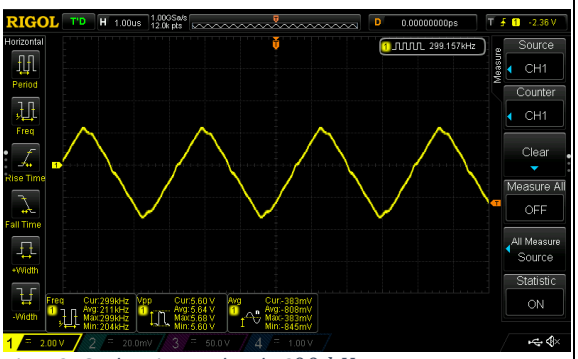

Fig. 73. Onda triangular de 300 kHz.

- 400 kHz

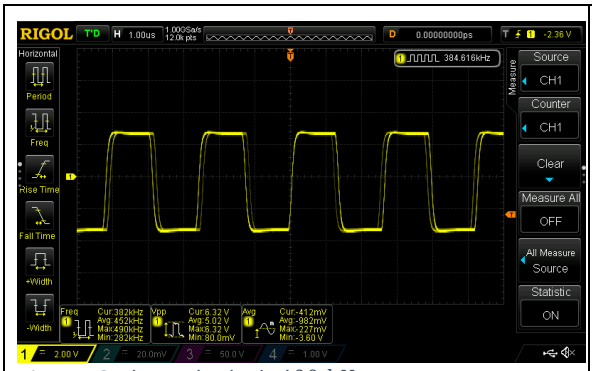

Fig. 74. Onda cuadrada de 400 kHz.

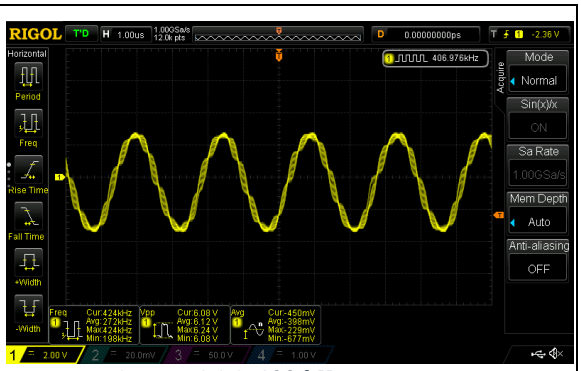

Fig. 75. Onda senoidal de 400 kHz.

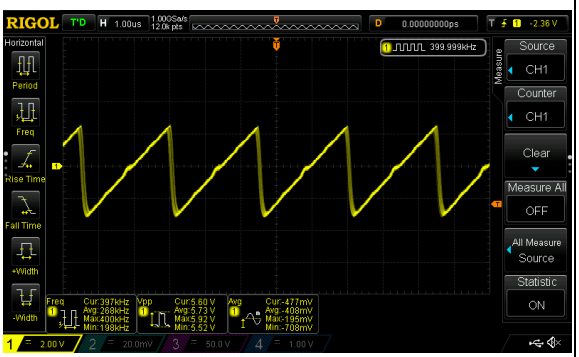

Fig. 76. Onda diente de sierra de 400 kHz.

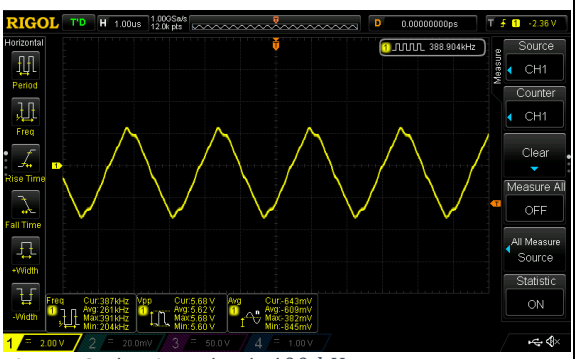

Fig. 77. Onda triangular de 400 kHz.

- 500 kHz

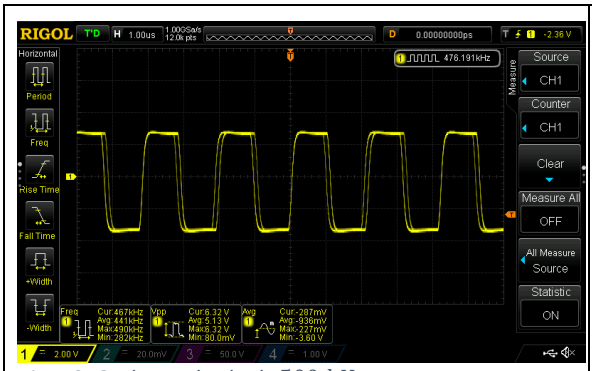

Fig. 78. Onda cuadrada de 500 kHz.

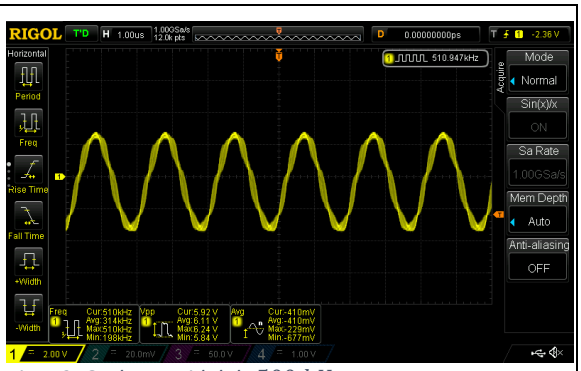

Fig. 79. Onda senoidal de 500 kHz.

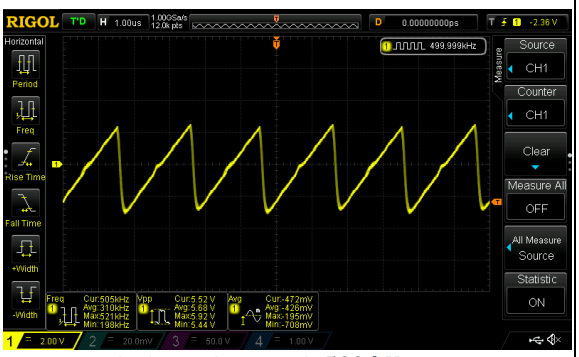

Fig. 80. Onda diente de sierra de 500 kHz.

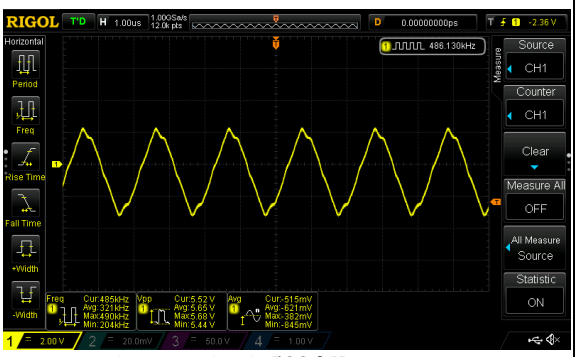

Fig. 81. Onda triangular de 500 kHz.

Observando las figuras podemos concluir que conseguimos llegar a los 200 kHz en todos los tipos de onda, pero ya se empieza a notar defectos en casi todas. Pasados los 200 kHz solo se puede destacar la onda diente de sierra como buena, y parcialmente la cuadrada y la triangular. La onda seno tiene demasiados defectos como para darla por buena.

Los defectos mencionados son producidos por el valor de la variable $incrementoPuntero$[3], el cual no es un número entero, sino que tiene decimales. El microcontrolador no es capaz de sacar muestras intermedias y lo que hace es sacar la más próxima, esto que provoca que en distintos ciclos saque muestras diferentes y por lo tanto la onda termine no siendo la ideal. Dependiendo del tipo de onda, tenemos un número de instrucciones por muestra (n_{cy}) distinto, por eso afecta de manera diferente según el tipo de onda elegida. Este hecho afecta también a la exactitud de la frecuencia que obtenemos. Cuánto más alta queramos que sea, menos precisa será.

Podemos resumir el apartado de la frecuencia en que conseguimos un ancho de banda efectivo en todas las ondas que empieza en 1 Hz y, aunque con algunos defectos, llega a los 200 kHz que teníamos como objetivo.

El desperfecto que se produce a la mitad de la altura de las ondas, sobre todo a partir de los 100 kHz, es debido a la tolerancia de las resistencias en el DAC. Aunque elegimos unas con un valor del 0,1 %, en comparación al 5% de las que usamos en las pruebas, todavía tiene influencia, aunque mucho menor. Si quisiéramos minimizarlo aún más, podríamos colocar resistencias de una tolerancia todavía menor, pero esto implicaría un mayor precio.

[3] $incrementoPuntero$: ver Bloque generador de señal (pág. 11).

Si nos fijamos en la atenuación, en algunas capturas se puede ver como el voltaje baja de los $6\ V_{pp}$ con los que empezamos, pero esto habría que achacarlo a los defectos en las ondas antes de que hayamos llegado el límite del ancho de banda del dispositivo. Hay que destacar que al haber elegido un amplificador con un "slew rate" alto no tenemos problemas en seguir los cambios bruscos que existen en las ondas como en el diente de sierra o en la cuadrada.

4.6.2. Mediciones de amplitud y offset

Una vez visto cómo se comporta el generador desde el punto de vista de la frecuencia, pasamos a estudiarlo en la amplitud y en el offset. Comprobamos ambas características en todas las ondas, pero sólo se incluyen en este trabajo imágenes de la onda senoidal por ser la más representativa.

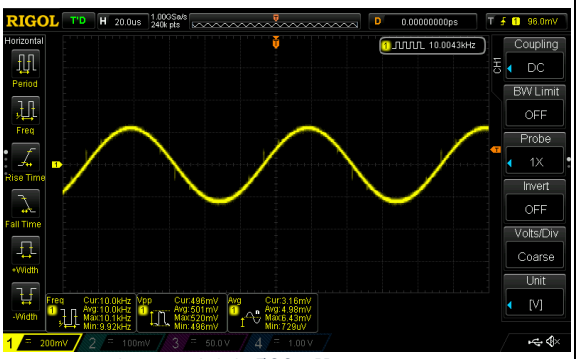

Fig. 82. Onda senoidal de $500mV$.

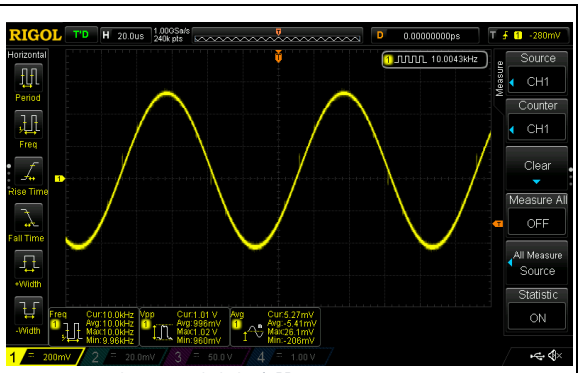

Fig. 83. Onda senoidal de $1\ V$.

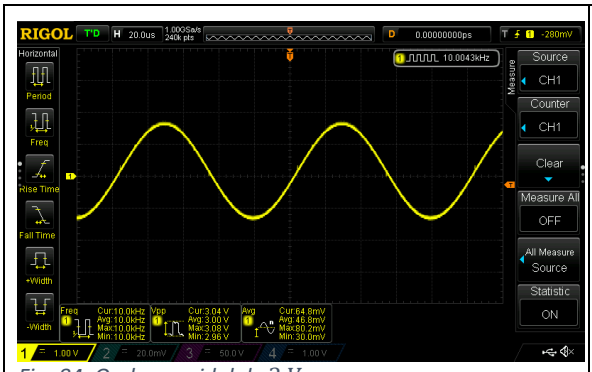

Fig. 84. Onda senoidal de $3\ V$.

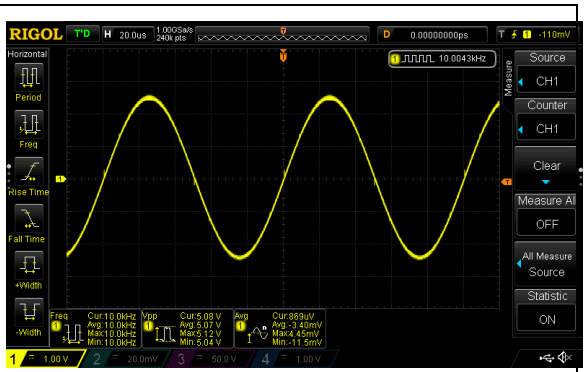

Fig. 85. Onda senoidal de $5\ V$.

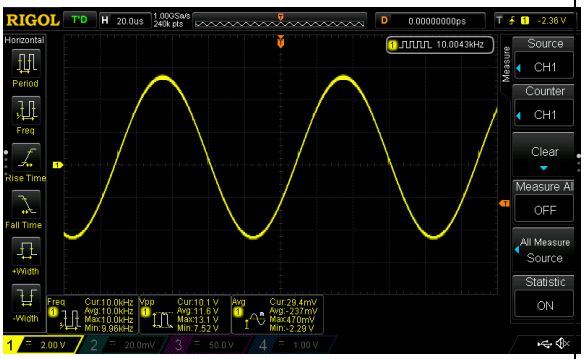

Fig. 86. Onda senoidal de $10\ V$.

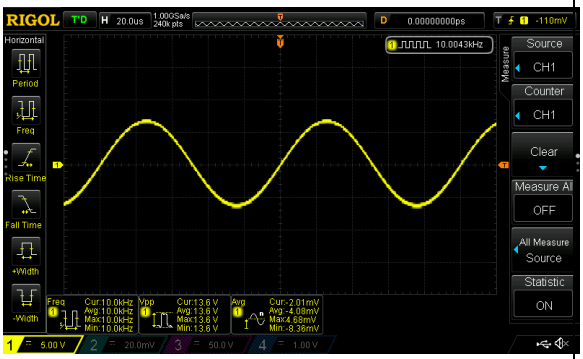

Fig. 87. Onda senoidal de $13\ V$.

En estas imágenes se puede comprobar lo que esperábamos en los cálculos y pruebas previas. Conseguimos llegar a los 13 V_{pp} que es el valor máximo que podemos conseguir con la configuración que elegimos del circuito de amplificadores operacionales. Una amplitud por debajo de los 500 mV empieza a ser inestable, tanto porque el ruido generado en el resto de la placa, como por potenciómetros usados. Por lo tanto, podemos resumir que somos capaces de generar señales de desde los 500 mV_{pp} hasta los 13 V_{pp} de una manera óptima.

Concluido el estudio del comportamiento de la amplitud, es momento de hacer lo mismo con el "offset". En las siguientes figuras podemos ver como conseguimos llevar una onda senoidal de 5 V_{pp} desde un "offset" nulo (Fig. 88) hasta los límites del amplificador. El límite superior lo encontramos al poco de superar los $+10\ V$ (Fig. 89) y el inferior en los $-10\ V$ (Fig. 90).

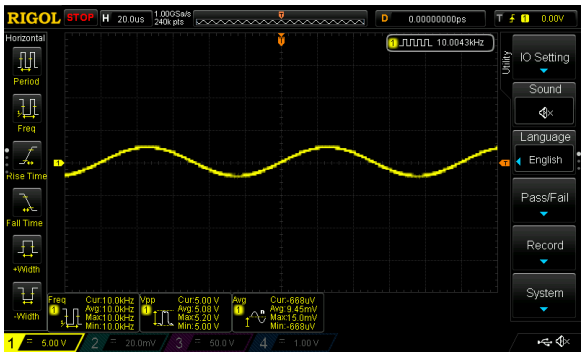

Fig. 88. Onda senoidal con offset 0 V.

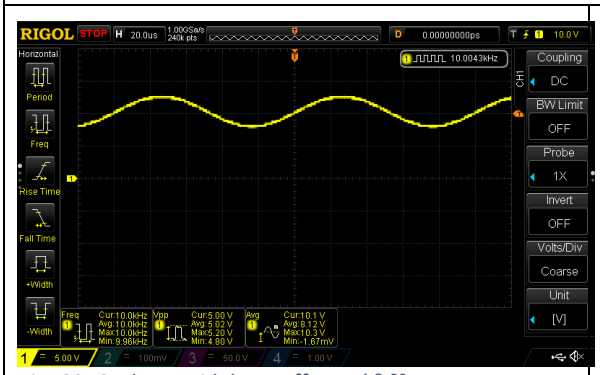

Fig. 89. Onda senoidal con offset $+10\ V$.

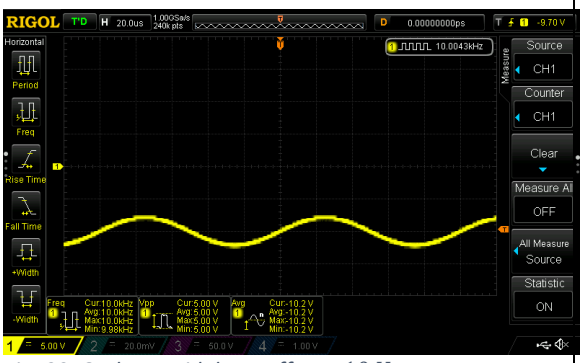

Fig. 90. Onda senoidal con offset $-10\ V$.

4.7. Pruebas de la interfaz

Durante las pruebas que se hicieron en el bloque de interfaz funcionaba todos los módulos, pero en el momento de juntarlos con la placa final nos dimos cuenta de que al intentar controlar el generador de señales con los dos métodos a la vez o reconocía uno u otro. Por falta de tiempo no se pudo indagar más y se optó por usarlos de manera independiente.

Como añadido, este bloque tiene la posibilidad de sustituir la onda usuario por otra. Algo que también se puede hacer al programarlo. La función de modificar la onda del usuario mediante la aplicación de PC funciona de correctamente de forma independiente, aunque de una manera lenta en comparación a la programación del microcontrolador. Pero a la hora de unirlo con el resto de las funciones no termina de ejecutarse como se esperaría. Es por ello por lo que hemos decidido dejar este punto para las vías futuras para poder mejorarlo y, posteriormente, si se consigue optimizar, añadirlo al conjunto.

5. Conclusiones

Una vez finalizado el análisis de la placa final podemos sacar algunas conclusiones. Vamos a empezar recordando los objetivos iniciales y los analizaremos uno por uno, razonando si se cumple o no y explicando el por qué.

- Variable en frecuencia, amplitud, y offset.

  El generador de señales que hemos construido puede producir una onda con el valor de frecuencia que el usuario desee tanto desde un control directo, con el encoder y el LCD, como desde el programa para PC mediante USB. La selección de la amplitud y del offset sólo se pueden hacer desde el control directo. Por lo tanto, podemos asegurar que este objetivo se cumple.

- Tipo de ondas: cuadrada, triangular, diente de sierra, seno e introducida por el usuario.

  Como hemos podido observar en las pruebas y comprobaciones, todas las ondas que teníamos como objetivo se pueden generar.

- Una frecuencia mínima de $50\ Hz$.

  Al no tener un filtro que se encargue de quitar la continua, las bajas frecuencias no son un problema, las ondas por debajo de la frecuencia mínima objetivo no se deforman. El valor mínimo que podemos obtener es $1\ Hz$, luego podemos concluir que este objetivo se cumple sobradamente.

- Una frecuencia máxima al menos de $200\ kHz$, con al menos 10 muestras.

  En todos los tipos de ondas que el dispositivo es capaz de generar podemos llegar a los $200\ kHz$ y podemos sobrepasarlos también en todas. Dependiendo del tipo de onda que generemos podemos llegar a unos máximos distintos, pero hay que tener en cuenta de que a medida que aumentemos la frecuencia, más se deformará y será menos exacta. Esto es debido a que los decimales despreciados en los cálculos de las muestras tendrán un peso cada vez mayor. No se ha limitado este parámetro porque considero que es más interesante que sea el usuario quien decida si le es válida una onda de una frecuencia mayor, aun no siendo perfecta.

- Salida máxima de $15\ V_{pp}$.

  Como ya describí en el transcurso de la memoria, este objetivo no se llega a cumplir porque entraba en conflicto otros propósitos que, aunque no son objetivos iniciales, resultan casi tan fundamentales. El primero conflicto es el de ser económico. Podría solucionarse añadiendo más amplificadores o eligiendo uno que admitiese un mayor rango de alimentación, pero ambas soluciones implicaban un mayor coste. El otro propósito con el que entra en conflicto es el manejo del dispositivo. Dificultaba el uso a la hora de fijar la amplitud y el offset. En las pruebas realizadas se consigue llegar a unos $13\ V_{pp}$ por lo que aun siendo un objetivo no cumplido nos quedamos cerca de cumplirlo.

- Resolución de voltaje de máximo $100\ mV$.

  En las pruebas realizadas ni si quiera en ondas de $500\ mV$ somos capaces de notar los saltos entre muestras, eso implica una resolución de voltaje mucho menor a la que tenemos como objetivo.

- Control del dispositivo directo e indirecto. Una mediante un LCD, un encoder y unos potenciómetros. La otra será una aplicación para PC.

  Dejar reflejado por escrito que el dispositivo se puede controlar de ambas formas es complicado, pero en las diversas fotos que se han adjuntado durante la memoria son una prueba de que el dispositivo cumple este objetivo. Tenemos el inconveniente de que no se pueden usar ambos a la vez.

Una vez detallados los objetivos individualmente, podemos realizar un análisis global en el que tengamos en cuenta otros factores.

El generador de señales que hemos construido puede que no sea el mejor dispositivo, pero tampoco busca serlo. Si lo comparamos con los que existen en el mercado, hay mejores, pero también más caros. La finalidad de este dispositivo, además de ser un TFE, era la de intentar diseñar y elaborar un prototipo de generador de señales que pudiese usarse en las situaciones en las que un alumno de ingeniería electrónica se pueda encontrar durante su carrera. Esas situaciones no son las que requieren tener las herramientas más caras del mercado, con solo disponer de ellas puede ser suficiente. Al ser un dispositivo económico facilita la posibilidad de que podamos tener uno casa para trabajar con él en cualquier momento. Eso puede ser más beneficioso para el alumno, a la hora de desarrollar sus conocimientos, que el tener un generador de señales carísimo en los laboratorios con el que no pueda trabajar tanto tiempo.

Una vez que hemos terminado el diseño y la fabricación, podemos echar la vista atrás y ver si las decisiones que se tomaron se elegirían de nuevo si volviéramos a empezar. La respuesta rápida es sí, pero con matices. En general, como el objetivo era conseguir un dispositivo funcional y económico, y creo que lo hemos conseguido, diría que la mayoría de las decisiones tomadas fueron las correctas. Ahora, en los siguientes párrafos, nos centraremos en comentar esos matices.

Creo que la relación microcontrolador-amplificador está algo descompensada. Si tomamos como base el microcontrolador, el amplificador es bastante caro y queda desaprovechado. Durante el diseño busqué algún amplificador que fuera más barato y se aprovechase más, pero no encontré ninguno. Algunos de los amplificadores que se adecuaban más al trabajo eran más caros, pero peores que él que elegí, por lo tanto, los descarté. No obstante, si se encontrase alguno, podría ser un buen cambio en el diseño.

Si lo vemos tomando como referencia el amplificador, un microcontrolador que alcanzase una mayor frecuencia de instrucción sería una buena mejora. Esta mejora lograría una mayor frecuencia máxima de las ondas con menor deformación y mayor exactitud. Para ello habría que irse a otras arquitecturas y, posiblemente a otros fabricantes, lo que escapa de los objetivos de este trabajo y, además, posiblemente encarecería el dispositivo.

Por otro lado, hay que destacar que hemos creado una PCB que se puede usar como plataforma que nos permita seguir desarrollando la parte del software, dándonos la posibilidad de mejorar las ondas que ya hemos implementado y/o añadir otras.

Continuando el punto anterior, pasamos a hablar del aspecto negativo del proyecto, que no es otro que el problema en la comunicación. Como ya comentamos antes existe el inconveniente de que no se pueden usar los dos métodos de comunicación a la vez. Posiblemente con más tiempo, y en otras circunstancias, se podría haber solucionado. Una posibilidad para solucionarlo podría ser cambiar el microcontrolador del bloque generador de señales a otro con más patillas, ya que en el que elegí están prácticamente todas en uso y deja poco margen de maniobra ante situaciones como esta. No obstante, siendo positivos, el generador, recordemos que es un prototipo, hace su función que es la de generar ondas.

El problema antes mencionado puede llegar a no crear inconvenientes, puesto que si solo vamos a usar uno de los modos de comunicación no supondría ningún impedimento. Incluso si sabemos de antemano que no vamos a usar uno de los métodos, podríamos quitarlo del presupuesto. Por ejemplo, si solo vamos a usarlo mediante el programa de PC, no harían falta el LCD ni el encoder. Además, podríamos sustituir el Arduino Nano por un simple conversor USB/UART, lo que rebajaría notablemente el precio del generador.

Por último, si pasamos a hablar sobre lo que he aprendido durante el desarrollo de este trabajo destacaría los siguientes puntos:

- He comprendido el trabajo que tiene realizar un proyecto prácticamente desde cero y con poca ayuda.
- He mejorado mis conocimientos sobre los buses de comunicación sobre todo a la hora de implementarlo en una aplicación para PC. También el mismo programa para PC ha sido una cosa que nunca había hecho y el lenguaje de programación que usé era nuevo. Puedo decir que he aprendido los principios de ese lenguaje.
- Hasta ahora nunca había trabajado tanto con un osciloscopio, este trabajo me ha permitido acostumbrarme al uso de esta herramienta, la cual ahora considero más esencial incluso que antes de comenzar este proyecto.

6. Vías futuras

Las vías futuras en este trabajo podrían clasificarse en dos casos. El primero serían las vías futuras modificando sólo el software. En este primer grupo entrarían algunas cosas que harían al generador de señales más interesante en el mercado, pero hay que tener en cuenta que el objetivo principal es realizar un TFE, no un producto para la venta. Realizar algunas de estas modificaciones podrían llevar mucho tiempo o necesitar de personal que no corresponden con finalidad de un TFE.

El segundo, las vías futuras modificando hardware y, de manera consiguiente el software, porque de otra manera, los cambios de hardware no servirían. En este grupo pasa lo mismo que en el anterior, algunas de las modificaciones se podrían haber incluido en el TFE, pero escapan su finalidad.

6.1. Vías futuras modificando software

Sin modificar la placa y solo realizando cambios en la programación de los microcontroladores se podrían realizar las siguientes mejoras:

- Más salidas de onda independientes, aunque parece que para realizar esta mejora necesitaríamos modificar hardware, realmente en el diseño se tuvo en consideración. Si tenemos dos PCB, solo es necesario tener las partes del bloque generador de señal y del bloque de adaptación de señal en ambas. El resto, el bloque de alimentación y el bloque interfaz, solo es necesario en una de las dos. Con solo dos cables del bloque de interfaz a la PCB sin ese bloque, podríamos tener 2 salidas independientes. Habría que hacer cambios en la programación del bloque interfaz para que pueda gestionar las dos comunicaciones con los dos bloques generadores y mandarles dos datos necesarios para que cada uno generase su onda. Para que el usuario pudiese controlar estas dos salidas, habría que modificar el código del programa de PC y el que nos deja visualiza en el LCD. Al añadir esta función, podríamos modificar el software para que pudiésemos controlar el desfase entre las dos señales generadas.
- Añadir más tipos de ondas. Realmente podríamos hacer casi cualquier onda periódica si la muestreamos y la guardamos en la memoria, pero tenemos un límite de datos que podemos guardar. Es por ello por lo que se podrían diseñar algoritmos específicos que nos permitiesen tener más ondas entre las que elegir. Las ondas dependerían de los usos que queramos darle al generador de señales, pero por nombrar algunas que no hemos incluido serían la PWM, la rampa o la trapezoidal.
- Barrido en frecuencia. Una mejora interesante sería el poder modificar en cada ciclo de onda el valor de $incrementoPuntero$. Al hacer esto podríamos conseguiríamos que a cada ciclo la frecuencia aumentase o disminuyese, a esto se le conoce barrido en frecuencia y es muy útil. Un uso de esta función sería para saber el ancho de banda de un dispositivo.
- Onda usuaria. Como dijimos con anterioridad en el transcurso de las pruebas y comprobaciones, el poder modificar de manera independiente la onda usuario que tenemos en memoria no terminó de funcionar correctamente cuando se unía al resto de funciones. Con más tiempo disponible se podría subsanar este problema para no tener que reprogramar el microcontrolador cada vez que queramos una onda distinta.

6.2. Vías futuras modificando hardware

El número de cambios a nivel de hardware para mejorar el dispositivo que podrían hacerse son muchos, pero manteniendo el objetivo de que sea económico, no son tantas.

La modificación más interesante que se podría hacer en el hardware ya se comentó en las conclusiones, buscar otro microcontrolador con una mayor frecuencia de instrucción. Esto puede parecer sencillo, pero cambiar de microcontrolador, sobre todo si se cambia de arquitectura o de fabricante, provocaría posiblemente tener que reescribir todo el código del bloque generador de señal. Lógicamente habría que hacer modificaciones en el diseño de la PCB y en el bloque de alimentación, ya que normalmente los microcontroladores de mucha frecuencia funcionan a $3{,}3\ V$. También podría ser necesario hacer cambios en el código del bloque de comunicación.

Otras posibles mejoras serían añadir otros modos de control del usuario con el dispositivo. Podríamos añadir un módulo Wifi de manera que podemos controlar el generador de señales desde dispositivos inalámbricos como móviles o tabletas. Para poder añadir esta mejora sólo tendríamos que modificar el bloque de comunicación, tanto en la parte del PCB como en la de código. También habría que hacer las aplicaciones para los dispositivos.

Dentro de este tipo de mejoras podríamos añadir más buses de comunicación como por ejemplo “ethernet”. Para que podamos hacer cualquiera de este tipo de mejoras habría que añadir tanto los conectores como los módulos conversores para que, en el caso de que el microcontrolador no pueda entenderse con el bus, sea capaz de hacerlo. También habría que añadir el código en el bloque interfaz para que funcione correctamente.

Por último, una idea que se tuvo durante el desarrollo fue la de construirle al generador una caja impresa en 3D, pero por falta de tiempo, solo se pudo plantear y hacer algunos diseños. Así que, lo dejamos en el bloque de vías futuras.

7. Bibliografía

Como libros de consulta generales se han usado los siguientes documentos;

- 16-Bit MCU and DSC Programmer's Reference Manual

  https://ww1.microchip.com/downloads/en/DeviceDoc/70157D.pdf

- Hoja de datos del dsPIC33EV256GM106:

  https://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33EVXXXGM00X-10X-Family-Data-Sheet-DS70005144H.pdf

- MPLAB Assembler, Linker and Utilities:

  https://ww1.microchip.com/downloads/en/DeviceDoc/50002106C.pdf

- MPLAB XC16 C Compiler Users Guide:

  http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB%20XC16%20C%20Compiler%20Users%20Guide%20DS50002071H.pdf

- dsPIC33-PIC24-FRM Flash-Programming:

  http://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33-PIC24-FRM,-Flash-Programming-DS70000609E.pdf

- Foro de microcontroladores con especial mención al hilo sobre el trabajo de los dsPIC en ensamblador:

  http://www.todopic.com.ar/foros/index.php?topic=45240.0

[1]. http://dispositivoselectrnicadepotencia.blogspot.com/2012/03/

[2].https://www.electronics-notes.com/articles/test-methods/signal-generators/what-is-a-signal-generator.php

[3]. https://www.redeweb.com/_txt/628/66.pdf

[4]. https://support.jlcpcb.com/article/83-smt-assembly-faqs

[5]. https://jlcpcb.com/capabilities/Capabilities

[6]. https://www.mouser.es/

Anexo I

PCB Esquemático

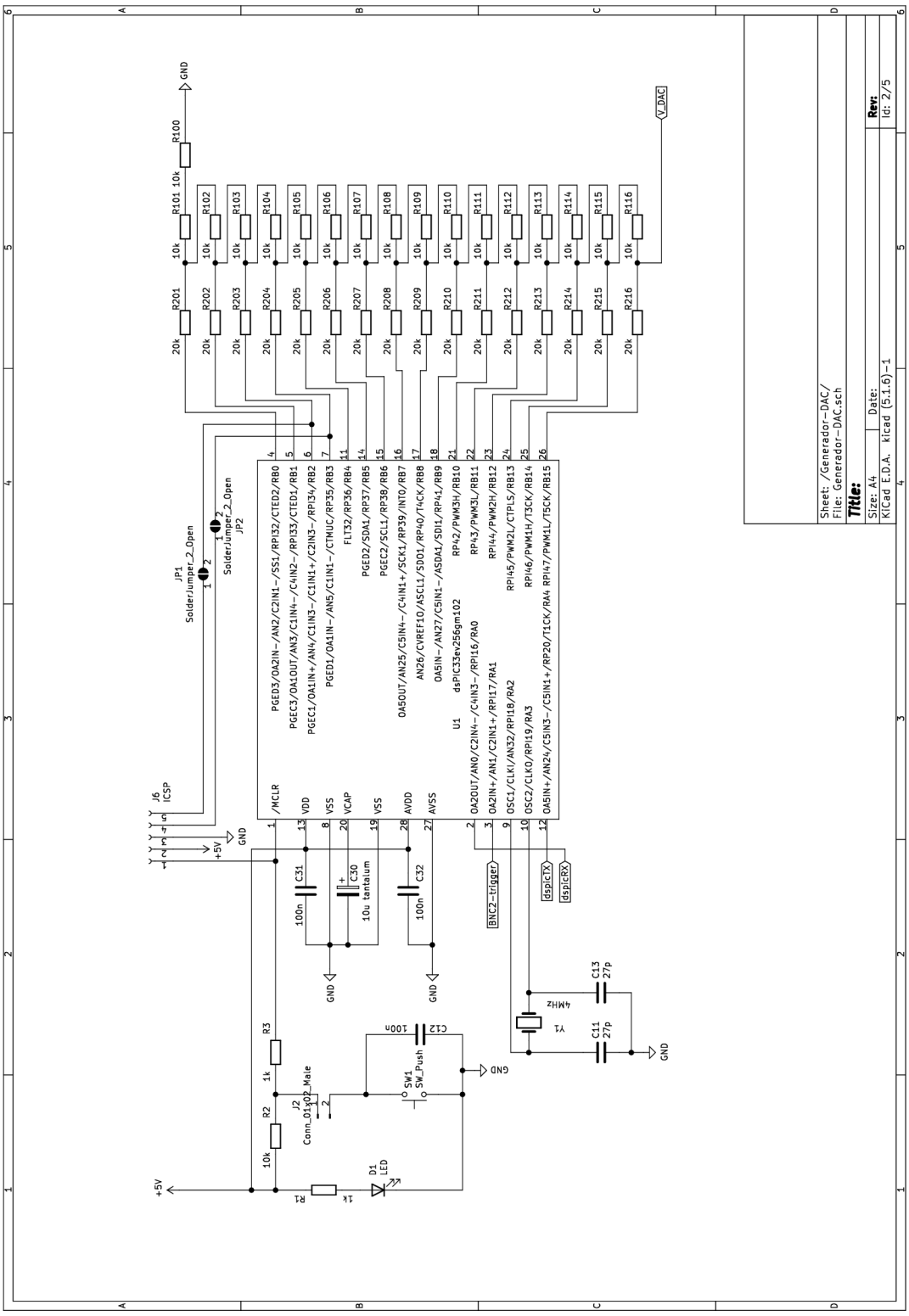

Fig. 91. Esquemático bloque generador de señal y DAC.

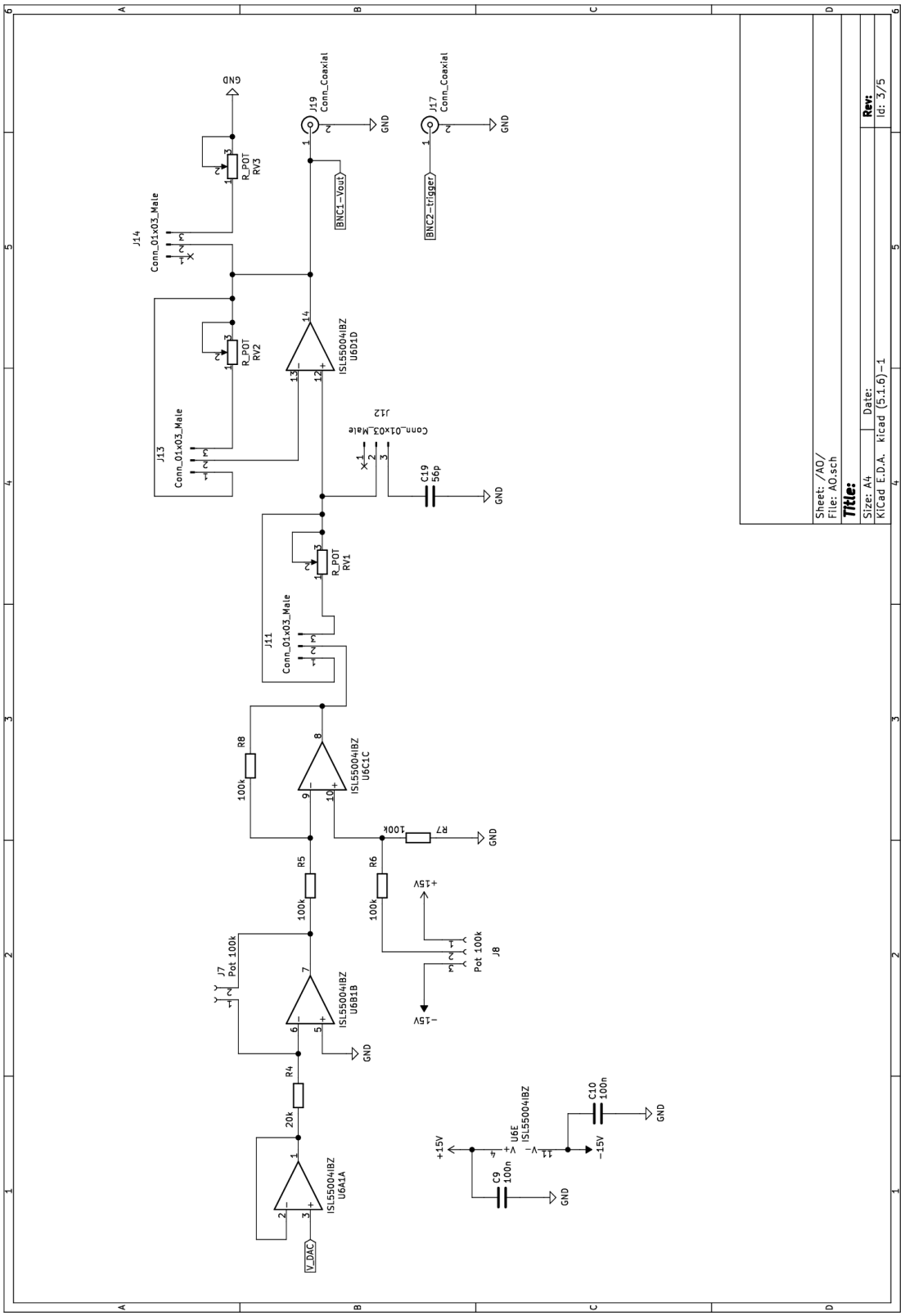

Fig. 92. Esquemático circuito amplificadores operacionales.

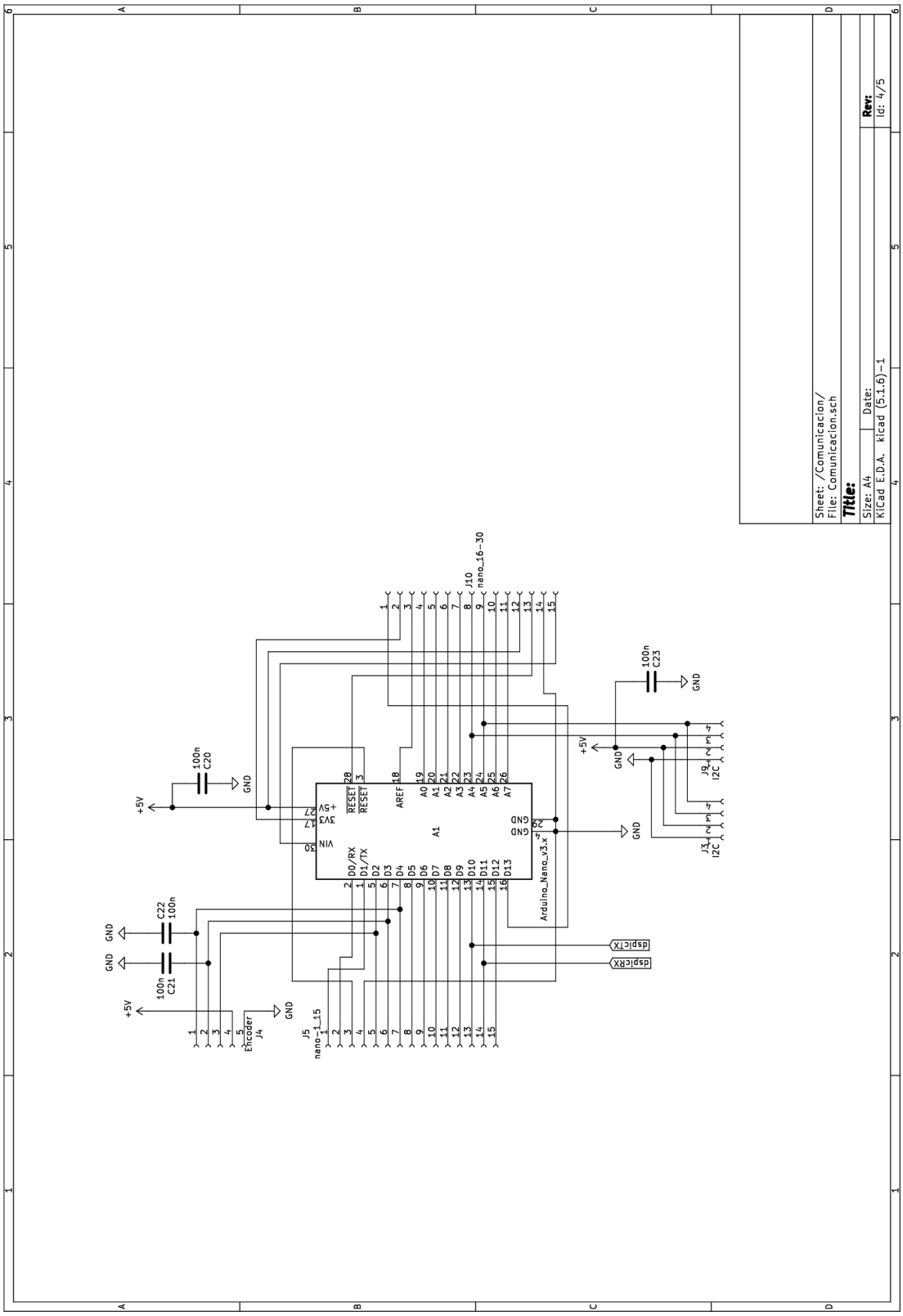

Fig. 93. Esquemático bloque interfaz y comunicación.

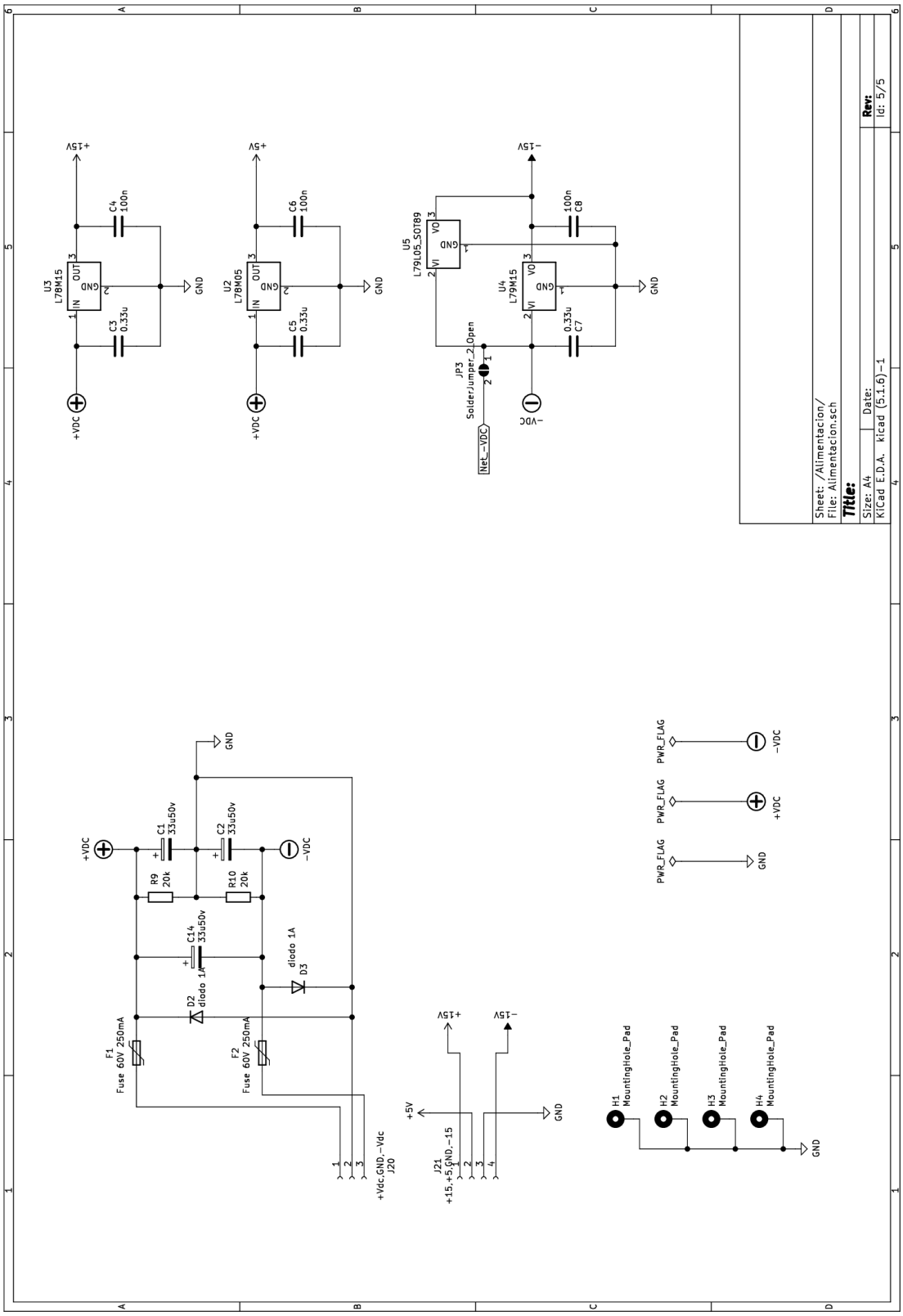

Fig. 94. Esquemático bloque alimentación.

PCB Layout

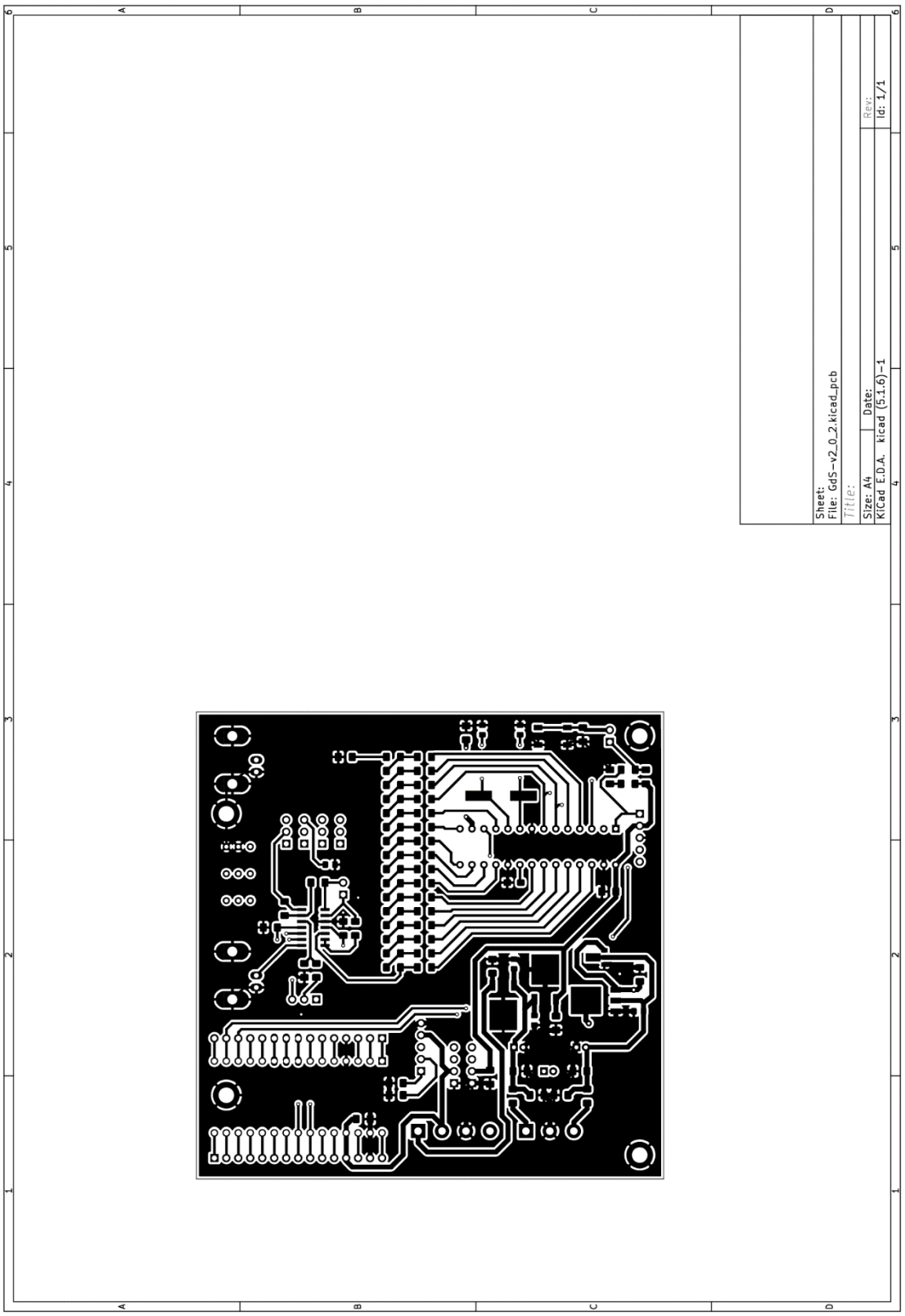

Fig. 95. PCB TOP (vista superior).

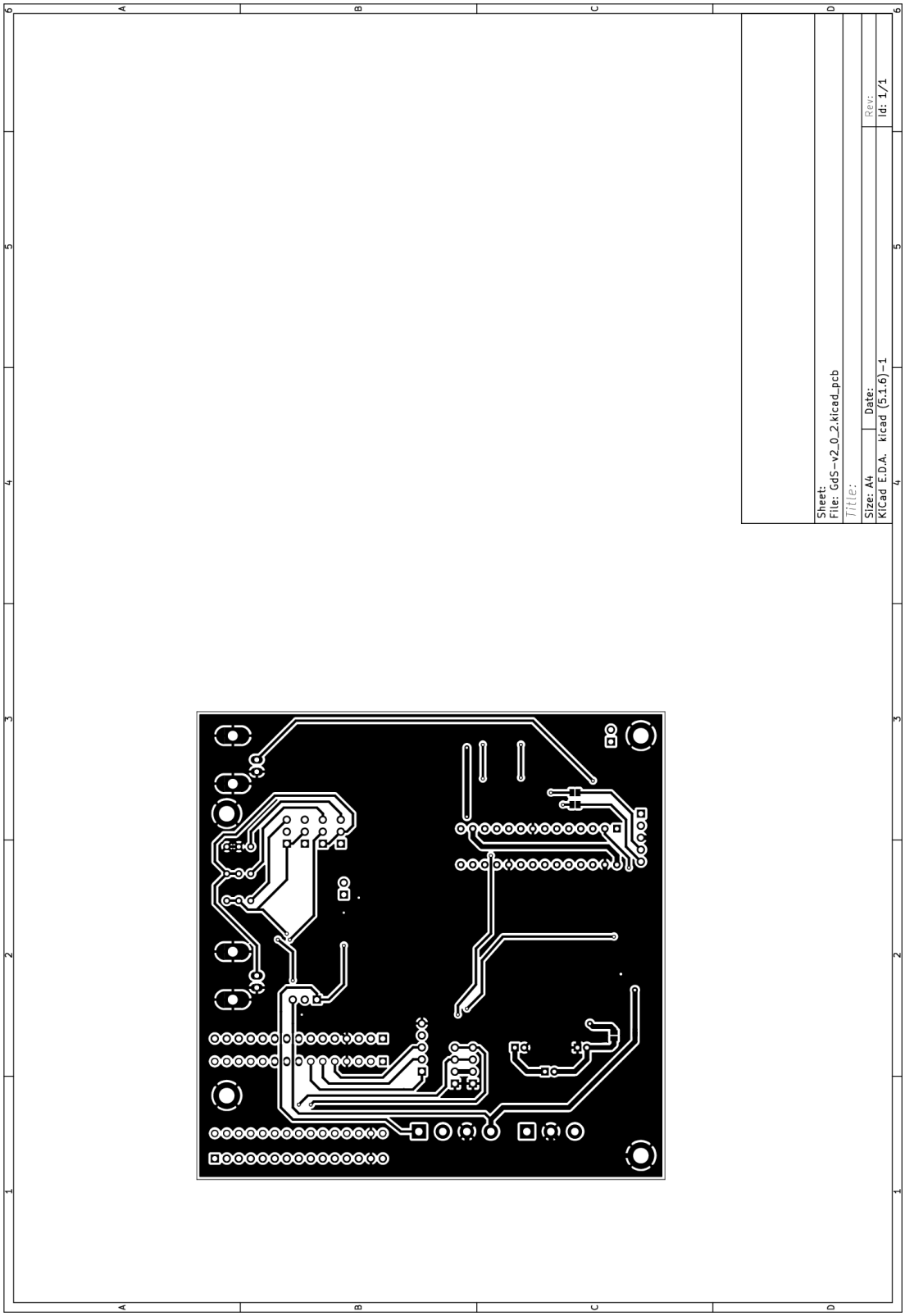

Fig. 96. PCB BOTTOM (vista inferior).

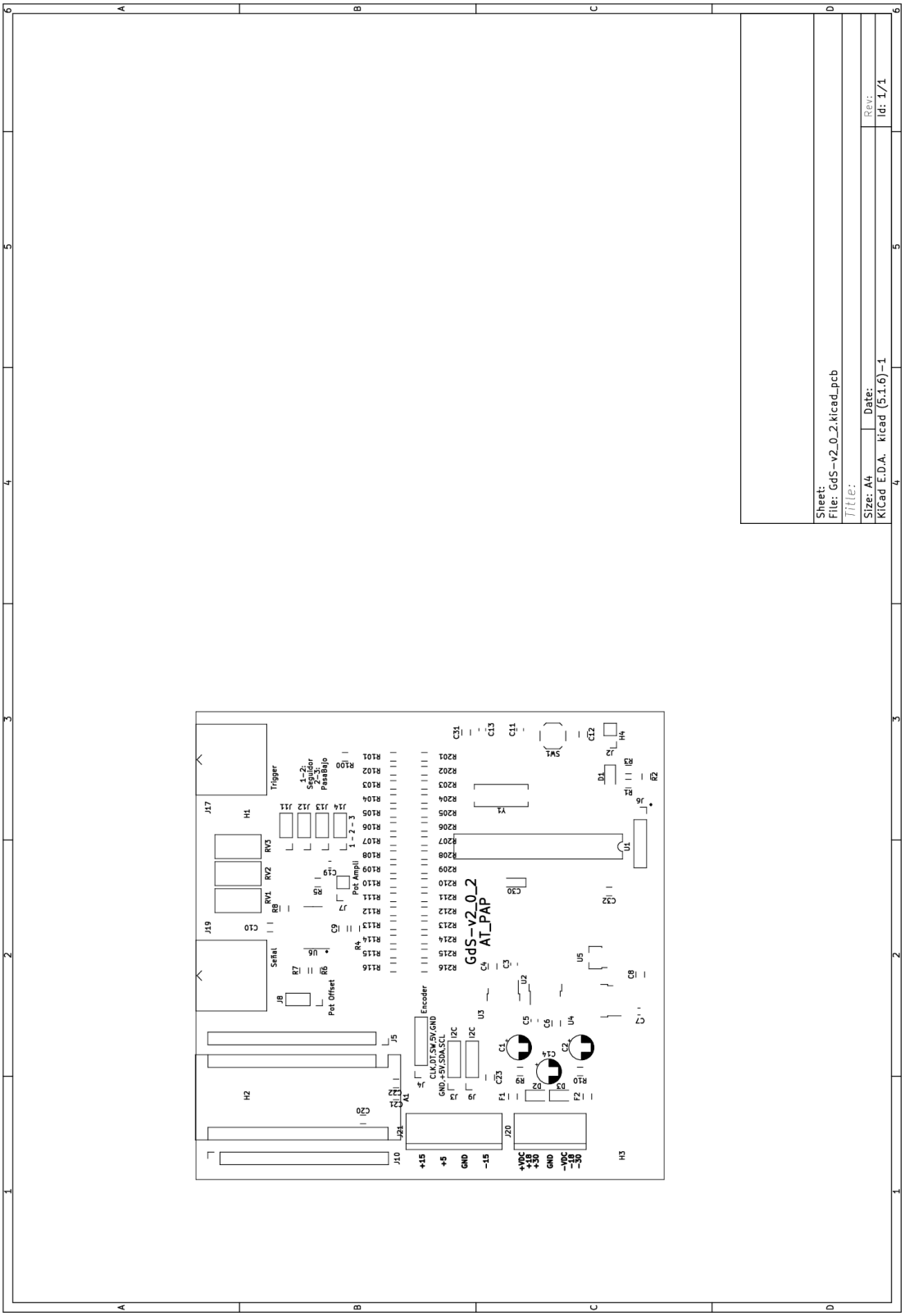

Fig. 97. PCB serigrafia (vista superior).

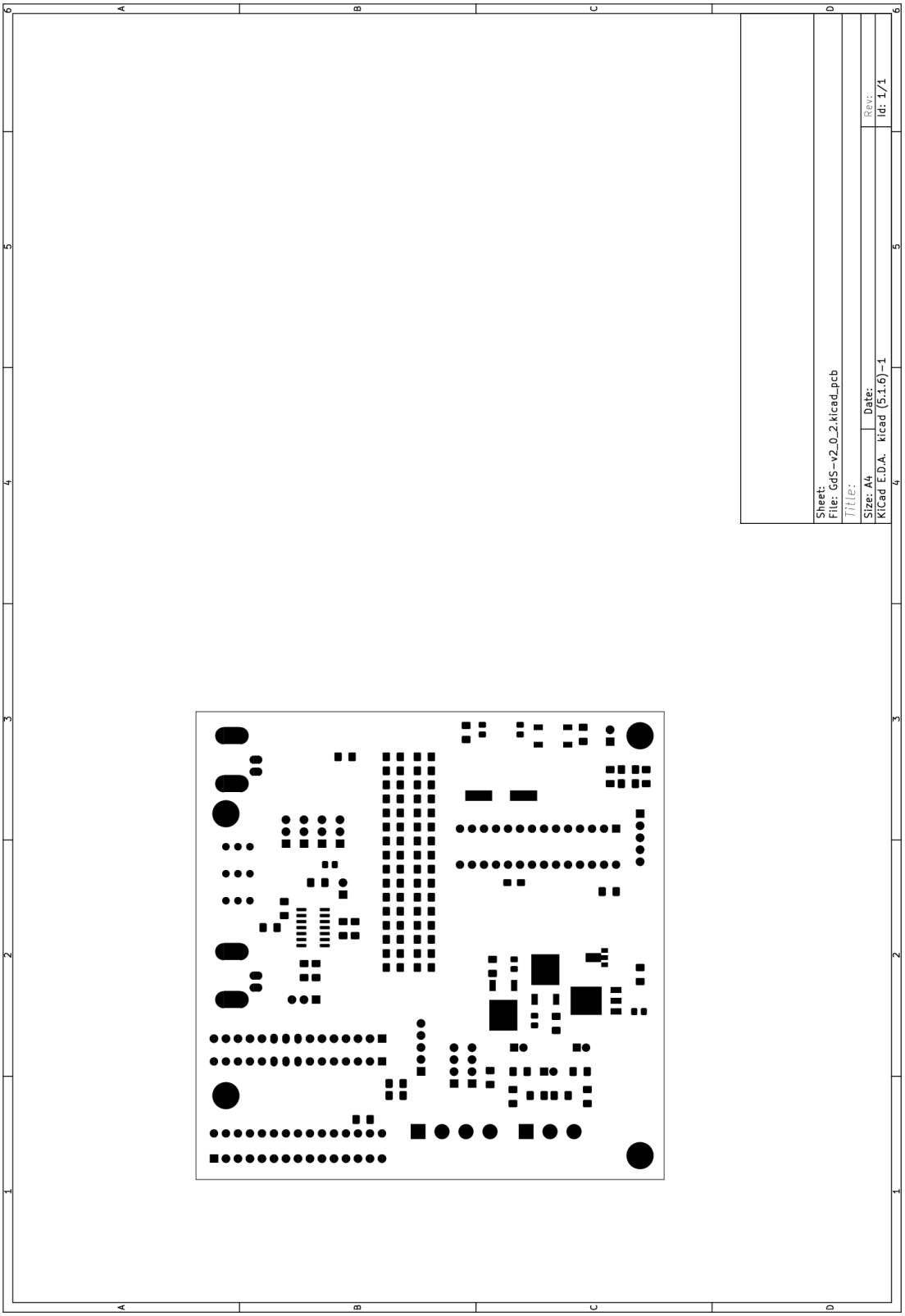

Fig. 98. PCB máscara de soldadura (vista superior).

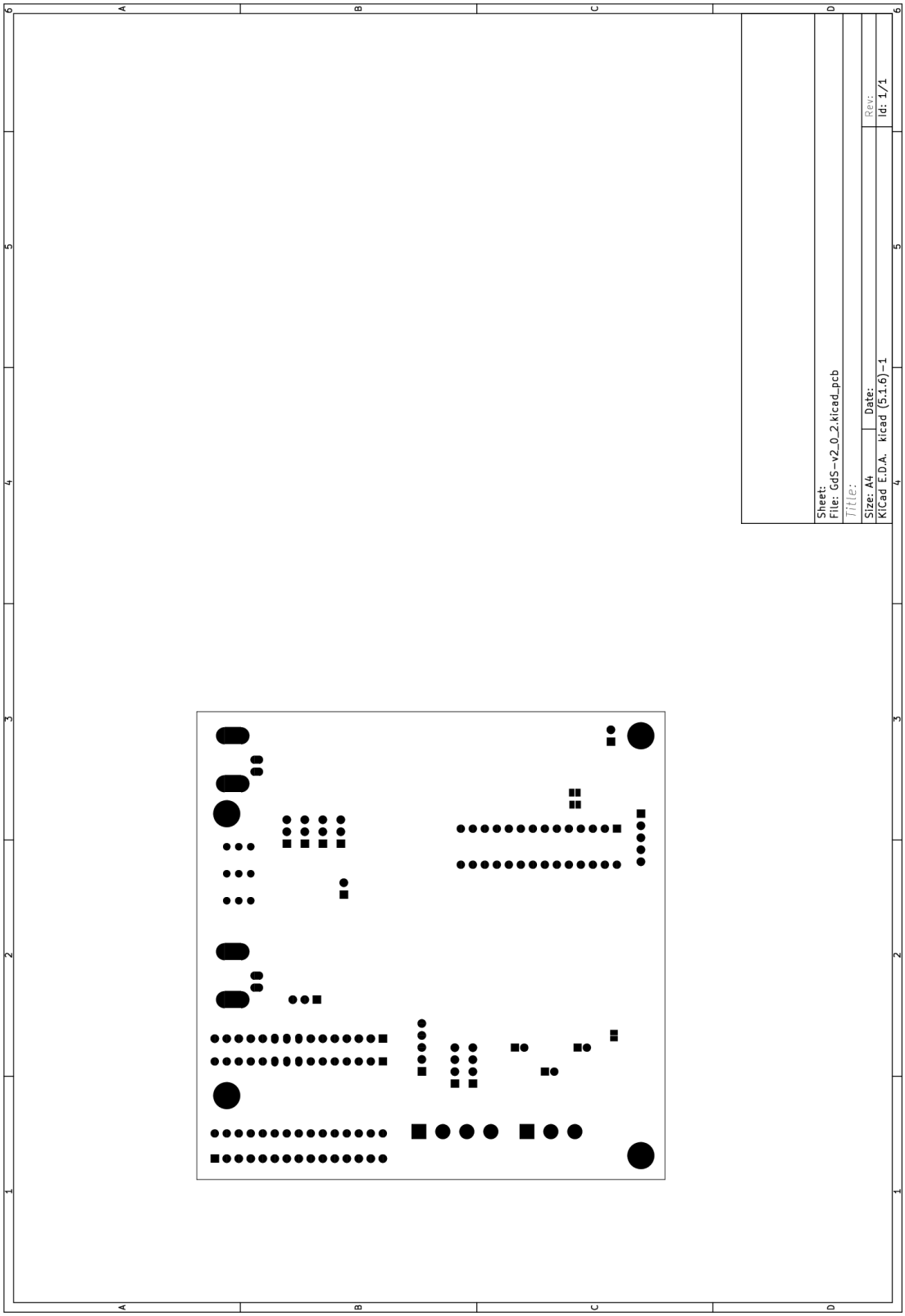

Fig. 99. PCB máscara de soldadura (vista inferior).

Anexo II

Programación bloque generador de señal

Main.c

```
#define SEPARADOR_UART     ':'
#define BYTES_UART_INICIO   1
#define BYTES_UART_FOUT    10
#define BYTES_UART_TIPOONDA 2

#define BYTES_UART    BYTES_UART_FOUT +1+ BYTES_UART_TIPOONDA
//BYTES_UART = BYTES_UART_FOUT : BYTES_UART_TIPOONDA
//         |Byte_UART_FOUT   |
//                        |Byte_UART_TIPOONDA
// Donde inicia los bytes de cada variable
#define Byte_UART_FOUT BYTES_UART-(BYTES_UART_FOUT +1+ BYTES_UART_TIPOONDA)
#define Byte_UART_TIPOONDA  BYTES_UART - (BYTES_UART_TIPOONDA)

volatile unsigned char dato[BYTES_UART] = {};
volatile unsigned char datoAnterior[BYTES_UART] = {};

#define BITS 16
#define ESTADOS pow(2, BITS)
#define BITS_32 32
#define ESTADOS_32 pow(2, BITS_32)
#define CY_OFFSET 0

#define ONDA 11
#define FOUT 69999
#define MUESTRAS 0


#define F_IN 4000000       //Frecuencia del cristal
//#define FCY 70000000UL
#define FCY 70002800       //Frecuencia de instruccion


#include <xc.h>
#include "setup.h"
#include "uart1.h"
#include <math.h>
#include <libpic30.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
42
43 int esperaDatos();
44 int esperaInicio();
45 int tratamientoDatos();
46 void generacionOnda();
47 int bucle();
48 long long calculo_muestras(long Fcy, long Fout, int n_cy);
49 long tablaCompensacion(long compensacion);
50 void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void);
51
52 //ensamblador
53 extern void pruebas(long salto, long n_delay, long compensacion);
54 extern void maxFrec();
55 extern void MHz_1();
56 extern void kHz_100();
57 extern void kHz_10();
58 extern void kHz_1();
59
60 extern void ondaSierraAsc_v2(long salto);
61 extern void ondaSierraDes_v2(long salto);
62 extern void ondaTriangular_v2(long salto, long salto4);
63 extern void ondaCuadrada_2(long salto);
64 extern void ondaSeno_v2(long salto);
65 extern void ondaUsuario_2(long salto);
66
67 volatile long Fout = FOUT;
68 volatile int tipoOnda = 11;
69 volatile float muestras = 0;
70 volatile long salto = 0;
71 volatile int n_cy = 20;
72
73 int main() {
74   //U1RX a RPI16(RA0)
75   //U1TX a RP20(RA4))
76   setup();
77   printf("\nFout:TipoOnda\n\n\n");
78
79   while(1){
80     Fout = 1000;
81     tipoOnda = 20;
82     while(esperaDatos() == 0){      }
83
84     n_cy = 20;  //n:cy normalizada
85     __asm__ __volatile__ ("clr    w10    \n");
86     generacionOnda(tipoOnda, salto, 0, 0);
87     __asm__ __volatile__ ("clr    LATB");
88     __delay_ms(10);
89   }
90
```

```
91    return 0;
92 }
93
94 int esperaDatos(){
95    //esperamos a recibir datos
96    if(dato[BYTES_UART - 1] != 0){
97       tratamientoDatos();
98       return 1;
99    }
100    else{
101       return 0;
102    }
103 }
104
105 int tratamientoDatos(){
106    if(dato[Byte_UART_FOUT] != 0){
107       int i = 0;
108       int j = BYTES_UART_FOUT - 1;
109       Fout = 0;
110       for(i = Byte_UART_FOUT; i < Byte_UART_TIPOONDA - 1; i++, j--){
111          Fout = Fout + (dato[i] - '0') * pow(10, j);
112       }
113       if(Fout <= 0 || Fout >= 500001){
114          Fout = FOUT;
115       }
116    }
117
118    if(dato[Byte_UART_TIPOONDA] != 0){
119       int i = 0;
120       int j = BYTES_UART_TIPOONDA - 1;
121       tipoOnda = 0;
122       for(i = Byte_UART_TIPOONDA; i < BYTES_UART /*- 1*/; i++, j--){
123          tipoOnda = tipoOnda + (dato[i] - '0') * pow(10, j);
124       }
125       if(tipoOnda < 1 || tipoOnda > 60){
126          tipoOnda = 11;
127
128       }
129    }
130
131    printf("Recibidos: %ld:%d\n", Fout, tipoOnda);
132    __delay_ms(10);
133
134    int i = 0;
135    for(i = 0; i < BYTES_UART; i++){
136       datoAnterior[i] = dato[i];
137       dato[i] = 0;
138    }
139
```

```
140     printf("\n\n\nDato Recibido. Esperando a inicio...\n");
141     return 1;
142 }
143
144 void generacionOnda() {
145     switch (tipoOnda) {
146         case 0:
147             muestras = (n_cy * Fout);
148             muestras = FCY / muestras;
149             salto = (long) (ESTADOS_32 / muestras);
150
151             pruebas(salto, 0, 0);
152             break;
153         case 1:
154             maxFrec();
155             break;
156         case 2:
157             MHz_1();
158             break;
159         case 3:
160             kHz_100();
161             break;
162         case 4:
163             kHz_10();
164             break;
165         case 5:
166             kHz_1();
167             break;
168         case 10:
169             n_cy = 8;
170             muestras = (n_cy * Fout);
171             muestras = FCY / muestras;
172             salto = (long) (ESTADOS_32 / muestras);
173             salto = salto*2;
174
175             ondaCuadrada_2(salto);
176             break;
177         case 11:
178             n_cy = 8;
179             muestras = (n_cy * Fout);
180             muestras = FCY / muestras;
181             salto = (long) (ESTADOS_32 / muestras);
182
183             ondaSierraAsc_v2(salto);
184             break;
185         case 12:
186             n_cy = 8;
187             muestras = (n_cy * Fout);
188             muestras = FCY / muestras;
```

```
189         salto = (long) (ESTADOS_32 / muestras);
190
191         ondaSierraDes_v2(salto);
192         break;
193       case 13:
194         n_cy = 9;
195         muestras = (n_cy * Fout);
196         muestras = FCY / muestras;
197         muestras = muestras - 1;
198         salto = (long) (ESTADOS_32 / muestras);
199         long salto2 = salto * 2;
200         long salto4 = salto * 4;
201
202         ondaTriangular_v2(salto2, salto4);
203         break;
204       case 20:
205         n_cy = 20;
206         muestras = (n_cy * Fout);
207         muestras = FCY / muestras;
208         salto = (long) (ESTADOS_32 / muestras);
209         salto = salto / 8;
210
211         ondaSeno_v2(salto);
212         break;
213       case 21:
214
215         n_cy = 20;
216         muestras = (n_cy * Fout);
217         muestras = FCY / muestras;
218
219         salto = (long) (ESTADOS_32 / muestras);
220         salto = salto / 8;
221         //ondaCoseno_v2(salto, 0, 0);
222
223         break;
224       case 25:
225
226         break;
227       case 30:
228
229         break;
230       case 40:
231
232         break;
233       case 41:
234         n_cy = 20;
235         muestras = (n_cy * Fout);//4.294.967.296
236         muestras = FCY / muestras;
237         salto = (long) (ESTADOS_32 / muestras);
```

```
238         salto = salto / 8;
239
240         ondaUsuario_2(salto);
241
242         break;
243       case 51:
244         //          NUsuario1();
245         break;
246       default:
247         break;
248   }
249   printf("PARADO. ESPERANDO NUEVA INICIO");
250 }
251
252 void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void) {
253
254   int i = 0;
255   for (i = 0; i < BYTES_UART; i++) {
256     if (dato[i] == 0)
257       break;
258   }
259   dato[i] = U1RXREG;
260   __asm__ __volatile__("setm    w10    \n");
261
262   IFS0bits.U1RXIF = 0;
263 }
```

Setup.c

```
1 #pragma config BWRP = OFF              // Boot Segment Write-Protect Bit (Boot Segment may be written)
 2 #pragma config BSS = DISABLED         // Boot Segment Code-Protect Level bits (No Protection (other than BWRP))
 3 #pragma config BSS2 = OFF             // Boot Segment Control Bit (No Boot Segment)
 4 #pragma config GWRP = OFF             // General Segment Write-Protect Bit (General Segment may be written)
 5 #pragma config GSS = DISABLED         // General Segment Code-Protect Level bits (No Protection (other than GWRP))
 6 #pragma config CWRP = OFF             // Configuration Segment Write-Protect Bit (Configuration Segment may be written)
 7 #pragma config CSS = DISABLED         // Configuration Segment Code-Protect Level bits (No Protection (other than CWRP))
 8 #pragma config AIVTDIS = DISABLE      // Alternate Interrupt Vector Table Disable Bit  (Disable Alternate Vector Table)
 9
10 // FBSLIM
11 #pragma config BSLIM = 0x1FFF         // Boot Segment Code Flash Page Address Limit Bits (Enter Hexadecimal value)
12
13 // FOSCSEL
14 #pragma config FNOSC = PRIPLL          // Initial oscillator Source Selection Bits (Primary Oscillator with PLL module (XT + PLL, HS + PLL, EC + PLL))
15 #pragma config IESO = ON               // Two Speed Oscillator Start-Up Bit (Start up device with FRC,then automatically switch to user selected oscillator source)
16
17 // FOSC
18 #pragma config POSCMD = HS             // Primary Oscillator Mode Select Bits (HS Crystal Oscillator mode)
19 #pragma config OSCIOFNC = ON           // OSC2 Pin I/O Function Enable Bit (OSC2 is general purpose digital I/O pin)
20 #pragma config IOL1WAY = OFF          // Peripheral Pin Select Configuration Bit (Allow Multiple reconfigurations)
21 #pragma config FCKSM = CSDCMD         // Clock Switching Mode Bits (Both Clock Switching and Fail-safe Clock Monitor are disabled)
22 #pragma config PLLKEN = ON            // PLL Lock Enable Bit (Clock switch to PLL source will wait until the PLL lock signal is valid)
23
24 // FWDT
25 #pragma config WDTPOST = PS32768       // Watchdog Timer Postscaler Bits (1:32,768)
26 #pragma config WDTPRE = PR128          // Watchdog Timer Prescaler Bit (1:128)
27 #pragma config FWDTEN = OFF            // Watchdog Timer Enable Bits (WDT and SWDTEN Disabled)
```

```
28 #pragma config WINDIS = OFF          // Watchdog Timer Window Enable Bit
(Watchdog timer in Non-Window Mode)
29 #pragma config WDTWIN = WIN25        // Watchdog Window Select Bits (WDT
Window is 25% of WDT period)
30
31 // FPOR
32 #pragma config BOREN0 = OFF          // Brown Out Reset Detection Bit (BOR is
Disabled)
33
34 // FICD
35 #pragma config ICS = PGD1            // ICD Communication Channel Select Bits
(Communicate on PGEC1 and PGED1)
36
37 // FDMTINTVL
38 #pragma config DMTIVTL = 0xFFFF      // Lower 16 Bits of 32 Bit DMT Window
Interval (Enter Hexadecimal value)
39
40 // FDMTINTVH
41 #pragma config DMTIVTH = 0xFFFF      // Upper 16 Bits of 32 Bit DMT Window
Interval (Enter Hexadecimal value)
42
43 // FDMTCNTL
44 #pragma config DMTCNTL = 0xFFFF      // Lower 16 Bits of 32 Bit DMT
Instruction Count Time-Out Value (Enter Hexadecimal value)
45
46 // FDMTCNTH
47 #pragma config DMTCNTH = 0xFFFF      // Upper 16 Bits of 32 Bit DMT
Instruction Count Time-Out Value (Enter Hexadecimal value)
48
49 // FDMT
50 #pragma config DMTEN = DISABLE       // Dead Man Timer Enable Bit (Dead Man
Timer is Disabled and can be enabled by software)
51
52 // FDEVOPT
53 #pragma config PWMLOCK = ON          // PWM Lock Enable Bit (Certain PWM
registers may only be written after key sequence)
54 #pragma config ALTI2C1 = OFF         // Alternate I2C1 Pins Selection Bit (I2C1
mapped to SDA1/SCL1 pins)
55
56 // FALTREG
57 #pragma config CTXT1 = NONE          // Interrupt Priority Level (IPL) Selection
Bits For Alternate Working Register Set 1 (Not Assigned)
58 #pragma config CTXT2 = NONE          // Interrupt Priority Level (IPL) Selection
Bits For Alternate Working Register Set 2 (Not Assigned)
59
60
61 #include "xc.h"
62 #include "uart1.h"
63
```

```
64
65
66 void setup(void){
67    TRISB = 0x0000;
68    LATB = 0x0000;
69    float M = 140, N1 = 2, N2 = 2;
70    PLLFBDbits.PLLDIV = M - 2;
71    CLKDIVbits.PLLPRE = N1 - 2;
72    CLKDIVbits.PLLPOST = N2 / 2 - 1;
73
74    //UART
75    UART1_Initialize();
76
77    //Interrupciones
78    IEC0bits.U1RXIE = 1;    //Enable
79    //IFS0bits.U1RXIF = 1;    //Flag
80 }
81
82
```

Uart1.c

```
#include <xc.h>
#include "uart1.h"

void UART1_Initialize(void)
{
   //UART
   TRISAbits.TRISA4=0;
   TRISB=0x0000;
   LATAbits.LATA4=1;

    //PSS
    PWMKEY = 0xABCD;
    PWMKEY = 0x4321;
    RPINR18bits.U1RXR = 16; // U1RX a RPI16(RA0)
    PWMKEY = 0xABCD;
    PWMKEY = 0x4321;
    RPOR0bits.RP20R=1;      //U1TX a RP20(RA4))

    ANSELAbits.ANSA0 = 0;   // Digital
    ANSELAbits.ANSA4 = 0;   // Digital

    U1BRG = 37;    //115200  baudios
    U1MODEbits.UARTEN = 1;     // enabling UART ON bit
    U1STAbits.UTXEN = 1;
}

uint8_t UART1_Read(void)
{
    while(!(U1STAbits.URXDA == 1))
    {

    }

    if ((U1STAbits.OERR == 1))
    {
       U1STAbits.OERR = 0;
    }

    return U1RXREG;
}

void UART1_Write(uint8_t txData)
{
    while(U1STAbits.UTXBF == 1)
    {

    }
```

```
48
49    U1TXREG = txData;    // Write the data byte to the USART.
50 }
51
52 bool UART1_IsRxReady(void)
53 {
54    return U1STAbits.URXDA;
55 }
56
57 bool UART1_IsTxReady(void)
58 {
59    return ((!U1STAbits.UTXBF) && U1STAbits.UTXEN );
60 }
61
62 bool UART1_IsTxDone(void)
63 {
64    return U1STAbits.TRMT;
65 }
```

MaxFrec.s

```
1
2    .global     _maxFrec
3
4    .text
5
6
7  _maxFrec:
8    clr LATB
9    setm LATB
10    nop
11    goto _maxFrec
12
13    .end
14
```

Multiplos_kHz.s

```
1
2   .global   _MHz_1
3   .global   _kHz_100
4   .global   _kHz_10
5   .global   _kHz_1
6   .text
7
8
9  _MHz_1:       ;35 positivos - 35 negativos
10    SETM   LATB
11    REPEAT  #35-3   ;33
12    NOP
13    CLR    LATB
14    REPEAT  #35-8;  8   ;27
15    NOP
16
17    BTSS   w10, #10
18    GOTO   _MHz_1
19    GOTO   fin
20
21 _kHz_100:      ;350 positivos - 350 negativos
22    SETM   LATB
23    REPEAT  #350-3
24    NOP
25    CLR    LATB
26    REPEAT  #350-8
27    NOP
28
29    BTSS   w10, #10
30    GOTO   _kHz_100
31    GOTO   fin
32
33 _kHz_10:           ;3500 positivos - 3500 negativos
34    SETM   LATB
35    REPEAT  #3500-3
36    NOP
37    CLR    LATB
38    REPEAT  #3500-8
39    NOP
40
41    BTSS   w10, #10
42    GOTO   _kHz_10
43    GOTO   fin
44
45 _kHz_1:       ;35000 positivos - 35000 negativos
46    SETM   LATB
47    REPEAT  #30000
```

```
48   NOP
49   REPEAT  #5000-3-2
50   NOP
51   CLR    LATB
52   REPEAT  #30000
53   NOP
54   REPEAT  #5000-8-2
55   NOP
56
57   BTSS   w10, #10
58   GOTO   _kHz_1
59   GOTO   fin
60
61
62
63
64 fin:
65   RETURN
66   .end
67
```

OndaCuadrada_v2. s

```
1
2    .include "p33EV256GM102.inc"
3
4    .global _ondaCuadrada_v2
5
6    .text
7
8  _ondaCuadrada_v2:
9     CLR    w5
10    CLR    w10
11    CLR    LATB
12    GOTO   _bucle
13
14
15 _bucle:
16 positiva:
17    SETM   LATB
18    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
19    ADDC   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
20
21    BTSS   SR,#0      ;f,#bit4 ;Bit Test f, Skip if Set 1 1(2or3) None
22    GOTO   positiva
23    REPEAT  #0
24    NOP
25    GOTO   negativa
26
27 negativa:
28    CLR    LATB
29    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
30    ADDC   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
31
32    BTSS   SR,#0      ;f,#bit4 ;Bit Test f, Skip if Set 1 1(2or3) None
33    GOTO   negativa
34    BTSS   w10, #10   ;f,#bit4 ;Bit Test f, Skip if Set 1 1(2or3) None
35    GOTO   positiva
36    GOTO   fin
37
38 fin:
39    RETURN
40    .end
41
```

OndaSierraAsc_v2. s

```
1
2   .include "p33EV256GM102.inc"
3
4   .global _ondaSierraAsc_v2
5
6   .text
7   ;sato   = w1:w0
8   ;salida = w5:w4
9   ;      LATB:---
10   ;      LATB=w5
11   ;w0:   salto[15:0]
12   ;w1:   salto[31:16]
13   ;w2:
14   ;w3:
15   ;w4:   salida[15:0]
16   ;w5:   salida[31:16] -> LATB
17
18 _ondaSierraAsc_v2:
19    CLR    w5
20    CLR    w4
21    CLR    w10
22    CLR    LATB
23    GOTO   _bucle
24
25 _bucle:
26    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
27    ADDC   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
28    MOV    w5, LATB
29
30    BTSS   w10, #10
31    GOTO   _bucle
32    GOTO   fin
33
34 fin:
35    RETURN
36    .end
37
38
```

OndaSierraDes_v2.s

```
1
2   .include "p33EV256GM102.inc"
3
4   .global _ondaSierraDes_v2
5
6   .text
7   ;sato   = w1:w0
8   ;salida = w5:w4
9   ;      LATB:---
10   ;      LATB=w5
11   ;w0:   salto[15:0]
12   ;w1:   salto[31:16]
13   ;w2:
14   ;w3:
15   ;w4:   salida[15:0]
16   ;w5:   salida[31:16] -> LATB
17
18 _ondaSierraDes_v2:
19    CLR    w5
20    CLR    w4
21    CLR    w10
22    CLR    LATB
23    GOTO   _bucle
24
25 _bucle:
26    SUB    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb ? Ws; 1 1 C,DC,N,OV,Z
27    SUBB   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb ? Ws ? (/C); 1 1 C,DC,N,OV,Z
28    MOV    w5, LATB
29
30    BTSS   w10, #10
31    GOTO   _bucle
32    GOTO   fin
33
34 fin:
35    RETURN
36    .end
37
```

OndaTriangular_v2.s

```
1
2   .include "p33EV256GM102.inc"
3
4   .global _ondaTriangular_v2
5
6   .text
7   ;sato * 2 = w1:w0
8   ;salida   = w5:w4
9   ;      LATB:---
10   ;      LATB = w5
11   ;w0:   2 * salto[15:0]
12   ;w1:   2 * salto[31:16]
13   ;w2:   4 * salto2[15:0]
14   ;w3:   4 * salto2[31:16]
15   ;w4:   salida[15:0]
16   ;w5:   salida[31:16] -> LATB
17
18 _ondaTriangular_v2:
19    CLR    w5
20    CLR    w4
21    CLR    w10
22    CLR    LATB
23    GOTO   _bucle
24
25 _bucle:
26 _bucleSubir:
27    NOP
28    MOV    w5, LATB
29    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
30    ADDC   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
31
32    BTSS   SR, #0     ;f,#bit4 ;Bit Test f, Skip if Set 1 1(2or3) None
33    GOTO   _bucleSubir
34
35    SUB    w4, w2, w4  ;Wb,Ws,Wd ;Wd = Wb - Ws; 1 1 C,DC,N,OV,Z
36    SUBB   w5, w3, w5  ;Wb,Ws,Wd ;Wd = Wb - Ws - (/C); 1 1 C,DC,N,OV,Z
37    MOV    w5, LATB
38    REPEAT  #1
39    NOP
40    GOTO   _bucleBajar
41
42
43 _bucleBajar:
44    NOP
45    MOV    w5, LATB
46    SUB    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb - Ws; 1 1 C,DC,N,OV,Z
47    SUBB   w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb - Ws - (/C); 1 1 C,DC,N,OV,Z
```

```
48
49    BTSC   SR, #0     ;f,#bit4 ;Bit Test f, Skip if Clear 1 1(2or3) None
50    GOTO   _bucleBajar
51
52    REPEAT  #0
53    NOP
54    ADD    w4, w2, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
55    ADDC   w5, w3, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
56    MOV    w5, LATB
57
58    REPEAT  #0
59    NOP
60    BTSS   w10, #10   ;f,#bit4 ;Bit Test f, Skip if Set 1 1(2or3) None
61    GOTO   _bucleSubir
62    GOTO   fin
63
64 fin:
65    RETURN
66    .end
67
```

OndaTrigonometrica_v2.s

```
1
2   .equ      nBits,  12
3   .equ      bits,   4096
4   .equ      bits10, 1023
5
6   .global _ondaUsuario_2
7
8   .text
9
10 _ondaUsuario_2:
11    MOV    #tblpage(TablaUsuario_2), w5      ; obtenemos la pagina de nuestros
valores
12    MOV    w5, TBLPAG                  ; y movemos al registro en cuestion
13                                  ; Terminamos de armar nuestro puntero ahora queda
cargar el offset
14    MOV    #tbloffset(TablaUsuario_2), w5     ; Ahora ya tenemos nuestro puntero
completo
15                                  ; Y ubicado en TBLPAG:W5
16                  ;fin tabla (w7) = inicio de la tabla (w5) + elementos tabla (w8)
17    MOV    w5, w7        ;incio tabla
18    MOV    #bits*2, w8    ;elementos tabla 4096*2-1=8191, las posiciones son x2.
19    ADD    w7, w8, w7     ;Wb,Ws,Wd   .Wd = Wb + Ws
20    GOTO   bucleUsuario
21
22 bucleUsuario:
23    REPEAT  #5
24    NOP
25    TBLRDL  [w5], w6      ;Saco el valor correspondiente de la tabla
26    MOV    w6, LATB       ; lo mando a B
27    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
28    ADDC    w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
29                  ;Sumo el salto de cada ciclo
30
31                  ;si la posicion actual de la tabla es mayor que la posicion del final
32    CPSGT   w5, w7        ;Wb,Wn Compare Wb with Wn, Skip if >. 1 1 (2or3) None
33    GOTO   bucleUsuario
34    GOTO   _reinicioTabla
35
36 _reinicioTabla:    ;reiniciamos la tabla para el siguente ciclo, restando el tamaño
de la tabla
37    SUB    w5, w8, w5  ;Wb,Ws,Wd ;Wd = Wb - Ws. 1 1 C,DC,N,OV,Z
38    TBLRDL  [w5], w6
39    MOV    w6, LATB
40    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
41    ADDC    w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
42
43    ;REPEAT  #0
```

```
44    ;NOP
45    BTSS    w10, #10
46    GOTO    bucleUsuario
47    GOTO    fin
48
49
50 fin:
51    RETURN
52    .end
53
```

OndaUsuario_2.s

```
1
2   .equ      nBits,  12
3   .equ      bits,   4096
4   .equ      bits10, 1023
5
6   .global _ondaUsuario_2
7
8   .text
9
10 _ondaUsuario_2:
11    MOV    #tblpage(TablaUsuario_2), w5      ; obtenemos la pagina de nuestros
valores
12    MOV    w5, TBLPAG                  ; y movemos al registro en cuestion
13                                  ; Terminamos de armar nuestro puntero ahora queda
cargar el offset
14    MOV    #tbloffset(TablaUsuario_2), w5     ; Ahora ya tenemos nuestro puntero
completo
15                                  ; Y ubicado en TBLPAG:W5
16                  ;fin tabla (w7) = inicio de la tabla (w5) + elementos tabla (w8)
17    MOV    w5, w7        ;incio tabla
18    MOV    #bits*2, w8    ;elementos tabla 4096*2-1=8191, las posiciones son x2.
19    ADD    w7, w8, w7     ;Wb,Ws,Wd   .Wd = Wb + Ws
20    GOTO   bucleUsuario
21
22 bucleUsuario:
23    REPEAT  #5
24    NOP
25    TBLRDL  [w5], w6      ;Saco el valor correspondiente de la tabla
26    MOV    w6, LATB       ; lo mando a B
27    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
28    ADDC    w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
29                  ;Sumo el salto de cada ciclo
30
31                  ;si la posicion actual de la tabla es mayor que la posicion del final
32    CPSGT   w5, w7        ;Wb,Wn Compare Wb with Wn, Skip if >. 1 1 (2or3) None
33    GOTO   bucleUsuario
34    GOTO   _reinicioTabla
35
36 _reinicioTabla:    ;reiniciamos la tabla para el siguente ciclo, restando el tamaño
de la tabla
37    SUB    w5, w8, w5  ;Wb,Ws,Wd ;Wd = Wb - Ws. 1 1 C,DC,N,OV,Z
38    TBLRDL  [w5], w6
39    MOV    w6, LATB
40    ADD    w4, w0, w4  ;Wb,Ws,Wd ;Wd = Wb + Ws. 1 1 C,DC,N,OV,Z
41    ADDC    w5, w1, w5  ;Wb,Ws,Wd ;Wd = Wb + Ws + (C). 1 1 C,DC,N,OV,Z
42
43    ;REPEAT  #0
```

```
44    ;NOP
45    BTSS    w10, #10
46    GOTO    bucleUsuario
47    GOTO    fin
48
49
50 fin:
51    RETURN
52    .end
53
```

Anexo III

Programación bloque interfaz y comunicación, mediante Arduino

Interfaz directa

```
#include <SoftwareSerial.h>
#include <String.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 20, 4);

int pinA = 4;  // Connected to CLK on KY-040
int pinB = 3;  // Connected to DT on KY-040
int encoderPosCont = 0;
int pinALast;
int aVal;


int btnOK_pin = 2;
boolean btnOK_status;

int tipoOnda = 0;
long Fout = 0;
int cifrasFout[7] = { 0, 0, 0, 0, 0, 0, 0 };

#define numOndas 8
String sTipoOnda[numOndas] = { "Pruebas", "Max Frecuencia", "Cuadrada",
"Sierra Asc", "Sierra Des", "Triangular", "Seno", "Usuario"};

//RPINR18bits.U1RXR = 16;      // U1RX a RPI16(RA0)
//RPOR0bits.RP20R = 1;         // U1TX a RP20(RA4)
//SoftwareSerial uart(Rx, Tx)
SoftwareSerial uart(10, 11);
bool stringComplete = false;
int stringCont = 0;

String strOut;
String strIn;
String strFout;
String strTipoOnda;

// the setup function runs once when you press reset or power the board
void setup() {
  Serial.begin(115200);
  uart.begin(115200);

  attachInterrupt(digitalPinToInterrupt(btnOK_pin), btnOK_interrupt,
RISING);

  // Inicializar el LCD
  lcd.init();
  //Encender la luz de fondo.
  lcd.backlight();

  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
```

```
  pinMode(btnOK_pin, INPUT_PULLUP);
  pinALast = digitalRead(pinA);

  btnOK_status = 0;

  lcd.clear();
  delay(1000);
  lcd.setCursor(8, 1);
  lcd.print("GdS");
  delay(3000);
  //delay(1000);
  lcd.clear();
  lcd.setCursor(0, 2);
}

void loop() {
  btnOK_status = 0;

  if(stringComplete == true){
    uart.print(strIn);
    Serial.print(strIn);
    strIn = "";
    stringCont = 0;
    stringComplete = false;
  }

  //seleccion de onda
  encoderPosCont = 0;
  ondaSeleccion();
  tipoOnda = encoderPosCont;  //asociar a valor para enviar por la UART
  //btnOK_status = 0;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Onda seleccionada:");
  lcd.setCursor(0, 1);
  lcd.print(sTipoOnda[tipoOnda]);
  delay(2000);


  //selccion de frecuencia
  encoderPosCont = 0;
  frecuenciaSeleccion();
  //btnOK_status = 0;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Onda seleccionada:");
  lcd.setCursor(0, 1);
  lcd.print(sTipoOnda[tipoOnda]);
  lcd.setCursor(0, 2);
  lcd.print("Fout seleccionada:");
  lcd.setCursor(0, 3);
  lcd.print(Fout);
  //lcd.print(strFout);
  delay(5000);

  valorTipoOnda();
  conversionString();
  //Comprobacion
  //

  //Envio
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enviando..");
  lcd.setCursor(0, 1);
  lcd.print(strOut);
  //delay(2000);
  uart.print(strOut);
  //
  //lcd.clear();
  lcd.setCursor(0, 3);
  lcd.print("Envio completado");
  delay(10000);
}

void serialEvent() {
    while (Serial.available()) {
        // get the new byte:
        char inChar = (char)Serial.read();
        // add it to the inputString:
        strIn += inChar;
        stringCont ++;
        // if the incoming character is a newline, set a flag so the main
loop can
        // do something about it:
        //if (inChar == '\n') {
        if (inChar == '|') {
            stringComplete = true;
        }
    }
}

void btnOK_interrupt() {
  btnOK_status = !btnOK_status;
}

int cambiosEncoder(int maxVal) {
  int bVal = 0;
  aVal = digitalRead(pinA);
  bVal = digitalRead(pinB);
  if (aVal != pinALast) { // Means the knob is rotating
    // if the knob is rotating, we need to determine direction
    // We do that by reading pin B.
    //if (digitalRead(pinB) != aVal) {  // Means pin A Changed first -
We're Rotating Clockwise
    //if (bVal != aVal) {  // Means pin A Changed first - We're Rotating
Clockwise
    if (bVal == aVal) {  // Means pin A Changed first - We're Rotating
Clockwise
      encoderPosCont--;
      if (encoderPosCont < 0) {
        encoderPosCont = maxVal;
      }
    }
    else {// Otherwise B changed first and we're moving CCW
      encoderPosCont++;
      if (encoderPosCont > maxVal) {
        encoderPosCont = 0;
      }
    }
    pinALast = aVal;
    return 1;
```

```
  }
  return 0;
}

void ondaSeleccion() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Onda: ");
  lcd.setCursor(3, 1);
  lcd.print(sTipoOnda[numOndas - 1]);
  lcd.setCursor(0, 2);
  lcd.print("->");
  lcd.setCursor(3, 2);
  lcd.print(sTipoOnda[0]);
  lcd.setCursor(3, 3);
  lcd.print(sTipoOnda[1]);

  while (btnOK_status == 0) {
    if (cambiosEncoder(numOndas - 1) == 1) {
      lcd.clear();

      lcd.setCursor(0, 0);
      lcd.print("Onda: ");

      lcd.setCursor(0, 1);
      lcd.print("                    ");
      lcd.setCursor(3, 1);
      if (encoderPosCont == 0) {
        lcd.print(sTipoOnda[encoderPosCont + (numOndas - 1)]);
        //lcd.print(sTipoOnda[0]);
      }
      else {
        lcd.print(sTipoOnda[encoderPosCont - 1]);
      }

      lcd.setCursor(0, 2);
      lcd.print("                    ");
      lcd.setCursor(0, 2);
      lcd.print("->");
      lcd.setCursor(3, 2);
      lcd.print(sTipoOnda[encoderPosCont]);

      lcd.setCursor(0, 3);
      lcd.print("                    ");
      lcd.setCursor(3, 3);
      if (encoderPosCont == numOndas - 1) {
        lcd.print(sTipoOnda[encoderPosCont - (numOndas - 1)]);
      }
      else {
        lcd.print(sTipoOnda[encoderPosCont + 1]);
      }

    }
    delay(100);
  }
  btnOK_status = 0;
}

void frecuenciaSeleccion() {
  //  6543210
  //  0000000 -> long Fout;
```

```
//  |  |  |->uFout,         umFout,    uMFout
//    |  |->dFout,          dmFout
//   |  |->cFout,           cmFout

boolean bucleFout = 0;
boolean bucleCifra = 0;
int cifraPos = 0;
//  int cifrasFout[7] = {0, 0, 0, 0, 0, 0, 0};

encoderPosCont = 0;

lcd.clear();
lcd.setCursor(0, 0);
lcd.print("                    ");
lcd.setCursor(0, 0);
lcd.print("F: ");
for (int i = 0; i < 7; i++) {
  lcd.print(cifrasFout[i]);
}
lcd.setCursor(15, 0);
lcd.print("OK");

while (bucleFout == 0) {
  lcd.setCursor(0, 0);
  lcd.print("F: ");
  cambiosEncoder(7);
  lcd.setCursor(0, 1);
  lcd.print("                    ");
  if (encoderPosCont != 7) {
    lcd.setCursor(3 + encoderPosCont, 1);
    lcd.print("^");
    cifraPos = encoderPosCont;
  }
  else {
    lcd.setCursor(15, 1);
    lcd.print("^");
  }
  if (encoderPosCont == 7 && btnOK_status == 1) {
    bucleFout = 1;
  }
  else if (btnOK_status == 1) {
    //if (btnOK_status == 1 && encoderPosCont != 7) {
    lcd.setCursor(3 + encoderPosCont, 1);
    lcd.print("=");
    btnOK_status = 0;
    delay(100);
    bucleCifra = 0;
    //encoderPosCont = 0;
    encoderPosCont = cifrasFout[encoderPosCont];
    while (bucleCifra == 0) {
      cambiosEncoder(9);
      cifrasFout[cifraPos] = encoderPosCont;
      lcd.setCursor(3 + cifraPos, 0);
      lcd.print(cifrasFout[cifraPos]);
      if (btnOK_status == 1) {
        bucleCifra = 1;
        btnOK_status = 0;
        encoderPosCont = cifraPos;
      }
      delay(100);
    }
```

```
      delay(200);
    }
  }
  Fout = 0;
  Fout = cifrasFout[0] * pow(10, 6) + cifrasFout[1] * pow(10, 5) +
cifrasFout[2] * pow(10, 4) + cifrasFout[3] * pow(10, 3) + cifrasFout[4] *
pow(10, 2) + cifrasFout[5] * pow(10, 1) + cifrasFout[6] * pow(10, 0);
  strFout = "";

  for (int i = 0; i < 7; i++) {
    strFout = strFout + String(cifrasFout[i]);
  }
  btnOK_status = 0;
}

void valorTipoOnda() {
  switch (tipoOnda) {
    case 0:
      strTipoOnda = "00";
      break;
    case 1:
      strTipoOnda = "01";
      break;
    case 2:
      strTipoOnda = "10";
      break;
    case 3:
      strTipoOnda = "11";
      break;
    case 4:
      strTipoOnda = "12";
      break;
    case 5:
      strTipoOnda = "13";
      break;
    case 6:
      strTipoOnda = "20";
      break;
    case 7:
      strTipoOnda = "41";
      break;
    case 8:
      strTipoOnda = "41";
      break;
  }
}

void conversionString() {
  strOut = "";
  //strFout = "XxxxXXX"
  strOut = "000" + strFout + ":" + strTipoOnda;// + ":" +
"00000:00000:00000";
}
```

Interfaz indirecta, con Visual Studio en C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
//puerto serie
using System.IO.Ports;
//trabajar archivos
using System.IO;

using System.Threading;


namespace GdS_Interfaz_v1
{
   public partial class Form1 : Form
   {
      private delegate void DelegadoAcceso(string accion);

      private string strBufferIn;
      private string strBufferOut;
      private string strCorrecto;

      private int estado = 0;

      private int numeroIntentos = 0;

      public Form1()
      {
```

```
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    //strBufferIn = "";
    strBufferOut = "";

    //btnConectar.Enabled = false;
    btnConectar.Enabled = true;
    btnEnviar.Enabled = false;
    //btnEnviar.Enabled = true;
    btnIniciar.Enabled = false;
}

private void btnBuscarPuertos_Click(object sender, EventArgs e)
{
    string[] PuertosDisponibles = SerialPort.GetPortNames();

    cboPuertos.Items.Clear();

    foreach (string puerto_simple in PuertosDisponibles)
    {
        cboPuertos.Items.Add(puerto_simple);
    }

    if (cboPuertos.Items.Count > 0)
    {
        cboPuertos.SelectedIndex = 0;
        //MessageBox.Show("SELECCIONAR PUERTO");
        btnConectar.Enabled = true;
    }
    else
    {
        MessageBox.Show("No hay puertos");
```

```
        cboPuertos.Items.Clear();
        cboPuertos.Text = "                    ";

        //strBufferIn = "";
        strBufferOut = "";

        btnConectar.Enabled = false;
        btnEnviar.Enabled = false;
    }
}

private void btnConectar_Click(object sender, EventArgs e)
{
    try
    {
        if (btnConectar.Text == "Conectar")
        {
            spPuertoUART.BaudRate = Int32.Parse(cboBaudRate.Text);
            spPuertoUART.DataBits = 8;
            spPuertoUART.Parity = Parity.None;
            spPuertoUART.StopBits = StopBits.One;
            spPuertoUART.Handshake = Handshake.None;
            spPuertoUART.PortName = cboPuertos.Text;

            try
            {
                spPuertoUART.Open();
                btnConectar.Text = "Desconectar";
                btnEnviar.Enabled = true;
                btnIniciar.Enabled = true;
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message.ToString());
            }
```

```
        }
        else if (btnConectar.Text == "Desconectar")
        {
            spPuertoUART.Close();
            btnConectar.Text = "Conectar";
            btnEnviar.Enabled = false;
        }
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message.ToString());
    }
}

private void btnEnviar_Click(object sender, EventArgs e)
{
    string aux_txtFrecuencia = "";
    string aux_cboTipoOnda = "";

    try
    {
        spPuertoUART.DiscardOutBuffer();

        aux_txtFrecuencia = Frecuencia_conversion();

        aux_cboTipoOnda = tipoOnda_conversion();


        //Enviar Datos
        txtDatosRecibidos.Text = "";
        //metodo 1
        txtDebug.Text = aux_txtFrecuencia + ":" + aux_cboTipoOnda + "|";
        spPuertoUART.Write(txtDebug.Text);

        spPuertoUART.DiscardOutBuffer();
```

```
            numeroIntentos = 0;
        }
        catch (Exception exc)
        {
            MessageBox.Show(exc.Message.ToString());
        }

        btnIniciar.Text = "Iniciar";
    }

    private void btnIniciar_Click(object sender, EventArgs e)
    {
        if (btnIniciar.Text == "Iniciar")
        {
            try
            {
                spPuertoUART.DiscardOutBuffer();

                spPuertoUART.Write(txtDebug.Text);

                spPuertoUART.DiscardOutBuffer();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message.ToString());
            }
            txtInicio.Text = "Parar";
            btnIniciar.Text = "Parar";

            btnEnviar.Enabled = false;
        }
        else
        {
            spPuertoUART.Write("0");
            txtInicio.Text = "Iniciar";
```

```
            btnIniciar.Text = "Iniciar";

            btnEnviar.Enabled = true;
        }

    }

    private void btnCargar_Click(object sender, EventArgs e)
    {
        if(btnCargar.Text == "Cargar archivo")
        {
            try
            {
                //Pass the file path and file name to the StreamReader constructor
                StreamReader sr = new StreamReader("<Direccion donde se encuentre el archivo>");

                String line = sr.ReadLine();
                String texto = "";
                int i = 0;
                while ( line != null)
                {
                    i++;
                    line = sr.ReadLine();
                    texto += line + Environment.NewLine;
                }
                txtDebug.Text = "lineas: " + i + Environment.NewLine;

                //sr.Close();

                txtDebug.Text += texto;

                //close the file
                sr.Close();
                //Console.ReadLine();
```

```
        }
        catch (Exception exc)
        {
            Console.WriteLine("Exception: " + exc.Message);
        }
    }
    //mostrar la señal cargada

    if(btnCargar.Text == "Enviar tabla")
    {
        enviarTabla();
    }
}

private void DatoRecibido(object sender, SerialDataReceivedEventArgs e)
{
    strBufferIn = "";
    AccesoInterrupcion(spPuertoUART.ReadExisting());
    spPuertoUART.DiscardInBuffer();
}

private void AccesoInterrupcion(string accion)
{
    DelegadoAcceso Var_DelegadoAcceso;
    Var_DelegadoAcceso = new DelegadoAcceso(AccesoForm);
    object[] arg = { accion };
    base.Invoke(Var_DelegadoAcceso, arg);
}

private void AccesoForm(string accion)
{
    strBufferIn = accion;

    txtDatosRecibidos.Text += strBufferIn + "\t\t\t\t";
}
```

```
void Reenviar()
{
   spPuertoUART.DiscardOutBuffer();
   spPuertoUART.Write(txtDebug.Text);
}

void UARTCorrecto(int nBits)
{
   strCorrecto = "";
   for(int i = 0; i < nBits; i++)
   {
      strCorrecto += "1";
   }
   spPuertoUART.DiscardOutBuffer();
   spPuertoUART.Write(strCorrecto);
}

private string Frecuencia_conversion()
{
   string aux_txtFrecuencia = "";
   int i_aux_txtFrecuencia = 0;

   //FrecuenciaUnidades
   aux_txtFrecuencia = txtFrecuencia.Text;
   switch (cboFrecuenciaUnidad.Text)
   {
      case "Hz":
         i_aux_txtFrecuencia = 1;
         break;
      case "kHz":
         i_aux_txtFrecuencia = 1000;
         break;
      case "MHz":
         i_aux_txtFrecuencia = 1000 * 1000;
```

```
        break;
}
i_aux_txtFrecuencia = i_aux_txtFrecuencia * Int32.Parse(aux_txtFrecuencia);

//Frecuencia Formateo
if (i_aux_txtFrecuencia >= 1 && i_aux_txtFrecuencia < 10)
{
   aux_txtFrecuencia = "000000000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 10 && i_aux_txtFrecuencia < 100)
{
   aux_txtFrecuencia = "00000000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 100 && i_aux_txtFrecuencia < 1000)
{
   aux_txtFrecuencia = "0000000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 1000 && i_aux_txtFrecuencia < 10000)
{
   aux_txtFrecuencia = "000000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 10000 && i_aux_txtFrecuencia < 100000)
{
   aux_txtFrecuencia = "00000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 100000 && i_aux_txtFrecuencia < 1000000)
{
   aux_txtFrecuencia = "0000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 1000000 && i_aux_txtFrecuencia < 10000000)
{
   aux_txtFrecuencia = "000" + i_aux_txtFrecuencia.ToString();
}
else if (i_aux_txtFrecuencia >= 10000000 && i_aux_txtFrecuencia < 100000000)
{
```

```
        aux_txtFrecuencia = "00" + i_aux_txtFrecuencia.ToString();
    }
    else if (i_aux_txtFrecuencia >= 100000000 && i_aux_txtFrecuencia < 1000000000)
    {
        aux_txtFrecuencia = "0" + i_aux_txtFrecuencia.ToString();
    }

    Periodo_conversion_frec(i_aux_txtFrecuencia);

    return aux_txtFrecuencia;
}


private void Periodo_conversion_frec(int i_aux_txtFrecuencia)
{
    float f_aux_txtPeriodo = 0;

    //Periodo a enviar
    f_aux_txtPeriodo = 1 / ((float)i_aux_txtFrecuencia);
    if ((f_aux_txtPeriodo * 1) >= 1)
    {
        f_aux_txtPeriodo = f_aux_txtPeriodo * 1;
        cboPeriodoUnidad.Text = "s";
    }
    else if ((f_aux_txtPeriodo * 1000) > 1)
    {
        f_aux_txtPeriodo = f_aux_txtPeriodo * 1000;
        cboPeriodoUnidad.Text = "ms";
    }
    else if ((f_aux_txtPeriodo * 1000 * 1000) > 1)
    {
        f_aux_txtPeriodo = f_aux_txtPeriodo * 1000 * 1000;
        cboPeriodoUnidad.Text = "us";
    }
    else if ((f_aux_txtPeriodo * 1000 * 1000 * 1000) > 1)
```

```
            {
                f_aux_txtPeriodo = f_aux_txtPeriodo * 1000 * 1000 * 1000;
                cboPeriodoUnidad.Text = "ns";
            }

            txtPeriodo.Text = f_aux_txtPeriodo.ToString();
        }

        private string tipoOnda_conversion()
        {
            string aux_cboTipoOnda = "";

            //TipoOnda a enviar
            aux_cboTipoOnda = cboTipoOnda.Text;
            switch (aux_cboTipoOnda)
            {
                case "Pruebas":
                    aux_cboTipoOnda = "00";
                    break;
                case "MaxFrecuencia":
                    aux_cboTipoOnda = "01";
                    break;
                case "1MHZ":
                    txtFrecuencia.Text = "1";
                    cboFrecuenciaUnidad.Text = "MHz";
                    pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaCuadrada_v1;
                    aux_cboTipoOnda = "02";
                    break;
                case "100kHz":
                    txtFrecuencia.Text = "100";
                    cboFrecuenciaUnidad.Text = "kHz";
                    pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaCuadrada_v1;
                    aux_cboTipoOnda = "03";
                    break;
```

```
            case "10kHz":
                txtFrecuencia.Text = "10";
                cboFrecuenciaUnidad.Text = "kHz";
                pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaCuadrada_v1;
                aux_cboTipoOnda = "04";
                break;
            case "1kHz":
                txtFrecuencia.Text = "1";
                cboFrecuenciaUnidad.Text = "kHz";
                pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaCuadrada_v1;
                aux_cboTipoOnda = "05";
                break;
            case "Cuadrada":
                aux_cboTipoOnda = "10";
                pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaCuadrada_v1;
                break;
            case "Sierra Ascendente":
                aux_cboTipoOnda = "11";
                pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaSierraAsc_v1;
                break;
            case "Sierra Descendente":
                aux_cboTipoOnda = "12";
                pictureBox1.Image =
GdS_Interfaz_v1.Properties.Resources.ondaSierraDesc_v1;
                break;
            case "Triangular":
                aux_cboTipoOnda = "13";
                break;
            case "Seno":
                aux_cboTipoOnda = "20"; //"20";
                pictureBox1.Image = GdS_Interfaz_v1.Properties.Resources.ondaSeno_v2;
                break;
            case "Usuario":
```

```
            aux_cboTipoOnda = "41"; //"41";
            break;
        case "Nuevo Usuario 1":
            aux_cboTipoOnda = "51"; //"51";
            break;
    }

    return aux_cboTipoOnda;
}

private string Compen_conversion()
{
    string aux_txtCompen = "";
    int i_aux_txtCompen = 0;

    //Muestras a enviar
    aux_txtCompen = txtCompen.Text;
    i_aux_txtCompen = Int32.Parse(aux_txtCompen);

    if (i_aux_txtCompen == 0)
    {
        //txtDebug.Text = "00000";
        aux_txtCompen = "000" + "00";
    }
    else if (i_aux_txtCompen >= 1 && i_aux_txtCompen < 10)
    {
        //txtDebug.Text = "0000" + i_aux_txtCompen.ToString();
        aux_txtCompen = "0000" + i_aux_txtCompen.ToString();
    }
    else if (i_aux_txtCompen >= 10 && i_aux_txtCompen < 100)
    {
        //txtDebug.Text = "000" + i_aux_txtCompen.ToString();
        aux_txtCompen = "000" + i_aux_txtCompen.ToString();
    }
    else if (i_aux_txtCompen >= 100 && i_aux_txtCompen < 1000)
```

```
    {
        //txtDebug.Text = "00" + i_aux_txtCompen.ToString();
        aux_txtCompen = "00" + i_aux_txtCompen.ToString();
    }
    else if (i_aux_txtCompen >= 1000 && i_aux_txtCompen < 10000)
    {
        //txtDebug.Text = "0" + i_aux_txtCompen.ToString();
        aux_txtCompen = "0" + i_aux_txtCompen.ToString();
    }
    else if (i_aux_txtCompen >= 10000 && i_aux_txtCompen < 100000)
    {
        //txtDebug.Text = i_aux_txtCompen.ToString();
        aux_txtCompen = i_aux_txtCompen.ToString();
    }

    return aux_txtCompen;
}

private void cboTipoOnda_SelectedIndexChanged(object sender, EventArgs e)
{
    tipoOnda_conversion();

    if (cboTipoOnda.Text == "Nuevo Usuario 1")
    {   //Nuevo Usuario 2
        btnCargar.Text = "Enviar tabla";
    }
    else
    {
        btnCargar.Text = "Cargar archivo";
    }
}

private void enviarTabla()
{
    try
```

```
        {
            if(cboTipoOnda.Text == "Nuevo Usuario 1")
            {
                StreamReader sr = new StreamReader("C://Users/Spartero/Dropbox/Universidad/TFG/Interfaz/GdS-Interfaz_v_actual/Resources/Usuario1.txt");
                String line = "";
                txtDebug.Text = "Enviando...";
                //int i = 0;
                int i = 1;
                while (line != null)
                {
                    Thread.Sleep(2);
                    line = sr.ReadLine();
                    spPuertoUART.Write(line);
                    spPuertoUART.DiscardOutBuffer();

                    i++;
                    if (i - 1024 == 0)
                    {
                        i = 1;
                        txtDebug.Text = txtDebug.Text + i + "\t" + line + Environment.NewLine;
                        spPuertoUART.DiscardOutBuffer();
                        Thread.Sleep(500);
                    }
                }
                spPuertoUART.DiscardOutBuffer();
                Thread.Sleep(2);
                txtDebug.Text = txtDebug.Text + "ENVIADO. i: " + i;
                sr.Close();

            }

            if (cboTipoOnda.Text == "Usuario 2")
            {
```

```
            StreamReader sr = new StreamReader("<Direcion donde se encuentre el
archivo");
            String line = sr.ReadLine();
            txtDebug.Text = line;
            while (line != null)
            {
               Console.WriteLine(line);
               line = sr.ReadLine();
            }
            sr.Close();
            Console.ReadLine();
         }
      }
      catch (Exception exc)
      {
         Console.WriteLine("Exception: " + exc.Message);
      }
   }

   private void cboxFrecuencia_CheckedChanged(object sender, EventArgs e)
   {
      if( cboxFrecuencia.Checked == true)
      {
         cboxPeriodo.Checked = false;
      }
   }

   private void cboxPeriodo_CheckedChanged(object sender, EventArgs e)
   {
      if (cboxPeriodo.Checked == true)
      {
         cboxFrecuencia.Checked = false;
      }
   }
```

```
        private void cboBaudRate_SelectedIndexChanged(object sender, EventArgs e)
        {

        }
    }
}
```