“Generador de funciones arbitarias (AWG)”

Índice

1 Raspberry Pi Pico y AWG

1.1 Introducción

El proyecto consiste en la realización de un generador de funciones arbitrarias (AWG) con el microcontrolador Raspberry Pi Pico. Un AWG es un dispositivo que se encargar de generar una señal eléctrica, en la que el usuario puede modificar, en base a sus limitaciones, la frecuencia, amplitud, offset, entre otros parámetros. A diferencia de un generador de funciones convencional, este puede generar no solo las típicas funciones senoidal, triangular, cuadrada; sino que mediante una tabla o la ley de la función puede generar cualquier señal que sea periódica. Nos preguntamos ...

¿Cómo se produce la señal?¿Qué limitaciones posee el dispositivo final? Entre otras ...

1.2 Convertidor Digital a Analógico R-2R

Para generar la señal del generador de funciones, debemos tener a la salida un tipo de señal analógica, sin embargo *¿cómo logramos tener una señal analógica a partir de pines digitales del micro?* Una forma de lograr esto es utilizar un convertidor de digital a analógico del tipo escalera R-2R. Este convertidor funciona enviando un valor en bits, es decir activando diferentes GPIOs, que representan un valor de tensión. En la siguiente imagen se puede observar lo mencionado.

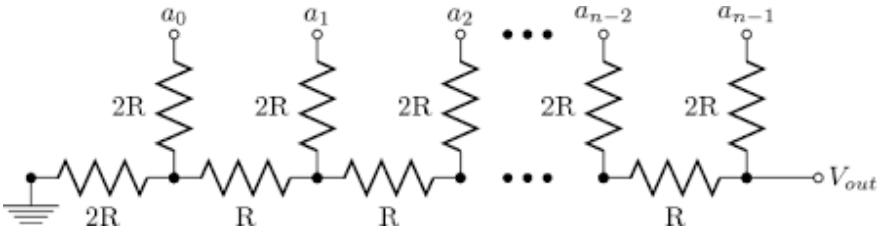

Figura 1: DAC R-2R

El bit mas significativo es el a_{n-1} y representa el bit con mayor nivel de tensión. Mientras que el menos significativo es el a_0 y representa la tensión mas baja posible. Cada a_n representa un GPIO del microcontrolador, por lo tanto poniendo en estado alto o bajo cada GPIO logramos tener una tensión de salida determinada. Notemos además que como estamos trabajando con un generador de funciones, la frecuencia debería variar en función de lo seteado por el usuario y las limitaciones del dispositivo. Por lo tanto fue un desafío importante encontrar un microcontrolador que permita cambiar los estados de los GPIOs con la mayor frecuencia posible para poder lograr la señal requerida.

1.3 Entrada/Salida Programable

Con el objetivo de poder obtener la mayor frecuencia posible, encontramos un microcontrolador (Raspberry Pi Pico) que posee dos periféricos especiales denominados PIO (Programable Input Output). Dicho periférico tiene 8 máquinas de estados, que actúan independientemente de los núcleos del microcontrolador y funciona a la velocidad máxima del reloj (configurado en 125 Mhz en este caso). Este periférico junto con dos DMAs anidados permiten generar la señal final. En el siguiente diagrama se puede visualizar mejor su funcionamiento general.

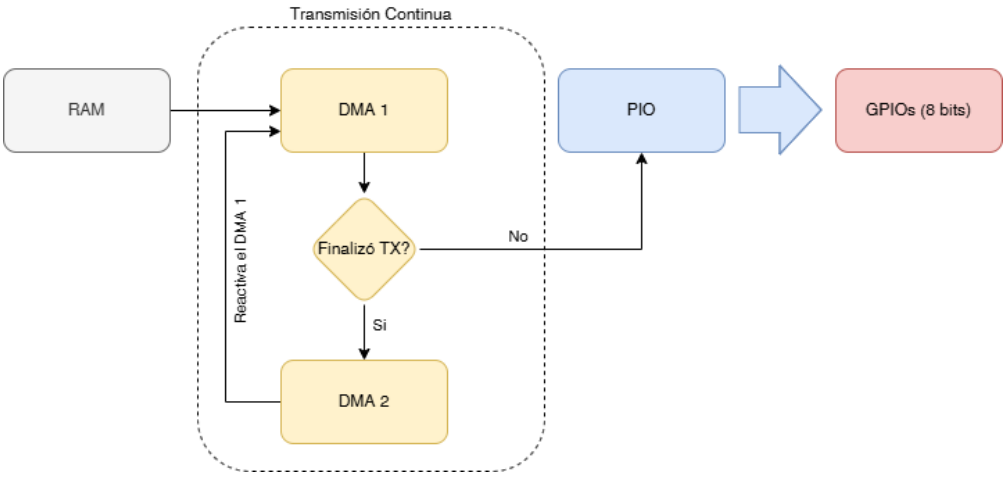

Figura 2: Diagrama en bloques del funcionamiento

Observamos que desde la RAM se pasa la información de una variable (haciendo referencia al valor en bits que toma la señal en ese momento) hacia el DMA 1. Este transfiere la información hacia el PIO y luego hacia los GPIOs. Cuando la transferencia termina activa el DMA 2, que esta programado para reactivar el DMA 1 y seguir ese mismo proceso siempre. De esta forma existe una transmisión continua.

1.4 Alcance del dispositivo

Debido a las limitaciones prácticas re planteamos los alcances con los valores que se comentan a continuación, señalizando aquellos que por tiempo o pudieron llegarse a lograr:

- $f_{max} = 200$ KHz y $f_{max} = 1{,}5$ MHz con 3,65 V_{pp}
- $V_{out} = 10\ V_{pp}$
- $V_{offset} = 5\ V_p$
- $I_{outmax} \simeq 100$ mA (No alcanzado)
- Salida de AC o DC.
- $Z_{out} \simeq 0\ \Omega$ o $Z_{out} \simeq 50\ \Omega$ (No alcanzado).

- $V_{in} = \pm 15V, 5V$

Para visualizar la información contamos con un display a color Nextion de 2,4", un encoder rotativo que nos permitirá cambiar los valores de los parámetros y pulsadores para seleccionar y avanzar en el menu. En las próximas secciones comentaremos mas acerca del código y respecto a la implementación.

2 Sistema en Tiempo Real

Un sistema en tiempo real procesa datos de acuerdo a tiempos específicos definidos por el usuario en las configuraciones. En nuestro proyecto implementamos FreeRTOS que es gratuito y junto con este utilizamos un sistema de modelado, llamado Quantum Leaps. En este último se definen objetos activos, que luego serán codificados como tareas y funcionarán a través de una máquina de estados.

¿Porqué utilizamos un RTOS?

Si bien en este caso no es estrictamente necesario utilizar un rtos, este tipo de sistemas en tiempo real nos permite tener un código mas limpio y escalable, a costa de utilizar un poco de memoria extra. De igual manera hicimos uso de varias tareas con diferentes prioridades.

2.1 Diagrama de Tareas

Mostramos a continuación un esquema de las tareas implementadas en el programa. No se vera en profundidad sobre las configuraciones de las tareas, pero si se harán observaciones que se consideren necesarias para su funcionamiento.

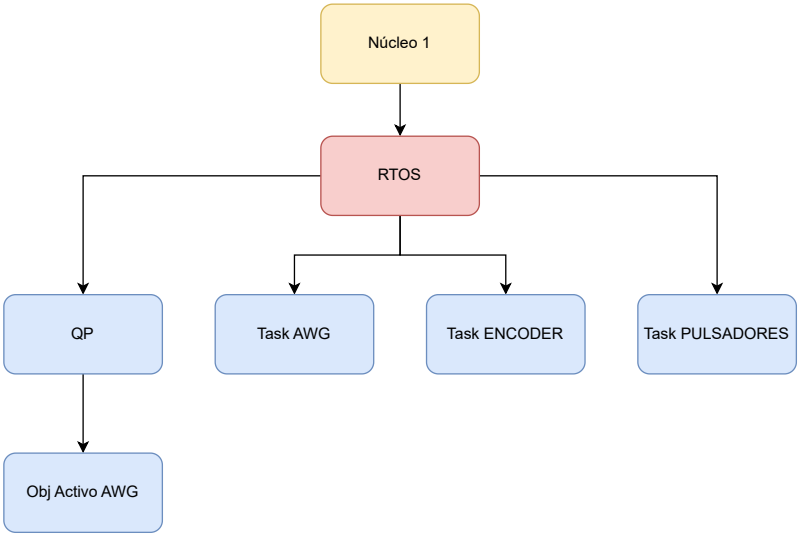

Figura 3: Diagrama de tares de RTOS

Si bien este microcontrolador posee dos núcleos, en este caso utilizamos uno solo y de esta forma mejoramos el consumo energético. Vemos que podemos pensar el programa con capas. En la capa más alta se encuentra el RTOS y luego bajamos hacia las tareas **AWG**,**ENCODER**,**PULSADORES** y por último la tarea del objeto activo.

- **Tarea ENCODER:** se encarga de detectar los pulsos del encoder rotativo nombrado anteriormente. Esta tarea posee una prioridad alta, de forma tal que cuando se genere una notificación (utilizada como semáforo), ingrese a dicha tarea y lea la dirección del giro.
- **Tarea PULSADORES:** esta es una tarea de procesamiento continuo, pero de baja prioridad, permitiendo así detectar por polling el estado de los pulsadores y ejecutar una acción en respuesta.
- **Tarea AWG:** la función de esta tarea es actualizar información proveniente de otras etapas (en especial la etapa de periférico, donde existe una interacción usuario-software).
- **Tarea QP Obj.Activo AWG:** se encarga de ejecutar la máquina de estados que se verá mas adelante. La prioridad es media y además posee una característica especial. Dicha tarea es creada, a diferencia del resto de tareas, de forma estática lo cual no permite ejecutar ciertas acciones como borrar, detener, etc, que se ejecutan en tiempo de ejecución. Esto posee la ventaja de que no ocupa memoria dinámica como si lo hacen el resto de tareas nombradas antes.

2.2 Cola de datos

Con la intención de intercambiar información entre archivos (awg.c y periferico.c) se realizó una cola de datos que consta de cinco elementos de tipo objeto *periferico_t*. Este objeto recopila toda la información relacionada con eventos producidos por el pulsado de botones, giro y dirección de encoder y demás. Podemos ver una representación en la siguiente figura.

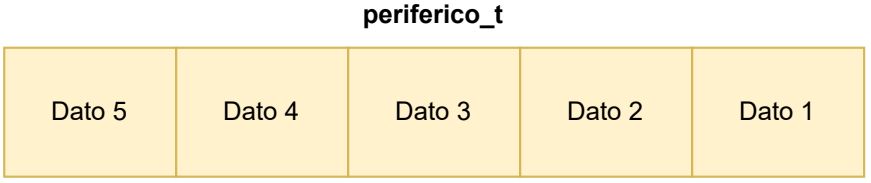

Figura 4: Cola de datos

3 Modelado UML y QP

3.1 Máquina de estado finita

En esta sección mostraremos como fue el modelado de nuestro sistema. Para esto comenzaremos mostrando la máquina de estado que utilizamos en Quantum Leaps. En la figura 5 se presenta la SM (State Machine), en donde podemos observar la mef principal denominada **Configuración**. En esta mef nos encargaremos de configurar los distintos parámetros que posee una señal en particular, a decir tipo de función, frecuencia, amplitud de salida en volts de pico y offset en volts de pico.

¿Cuál es la función de cada estado?

- **Estado Tipo_Func:** durante este estado se configura el tipo de señal a generar. Puede elegirse entre tres señales cargadas (senoidal, cuadrada, triangular). Posee el evento *ENCODER* que es activado cada vez que surge un movimiento en cualquier sentido del periférico. Por último posee dos transiciones *AVANC* y *ATRAS* que son generadas mediante los pulsadores correspondientes y su única función es intercambiar entre estados.

- **Estado Freq:** utilizamos este estado para configurar la frecuencia que queremos en nuestro dispositivo, dicho valor se muestra en pantalla. Además del evento *ENCODER*, se presenta un nuevo evento que también aparece en otros estados, este nuevo evento es denominado *MULTIPLICADOR*. Dicho evento es utilizado para cambiar el aumento de frecuencia de forma mas sencilla, cambiando de a múltiplos de 10 desde 1 Hz hasta 1 MHz.

- **Estado Amplitud-Offset:** en estos estados se configuran la amplitud final y el offset en volts de pico, respectivamente. Al igual que en el estado de frecuencia se tienen los mismos eventos.

- **Estado Confirm_config:** por último tenemos el estado de confirmación, durante este estado podremos elegir con el pulsador si seguir configurando nuestra señal o activarla con las configuraciones cargadas. Dicha elección se hace mediante el evento *CONFIRM*.

¿Cómo se representa la máquina de estados en FreeRTOS?

En la sección anterior comentamos el uso de FreeRTOS en el programa, mas aún esta máquina de estados es parte de un denominado objeto activo, el cual es generado (mediante código automático) como una tarea estática de FreeRTOS.

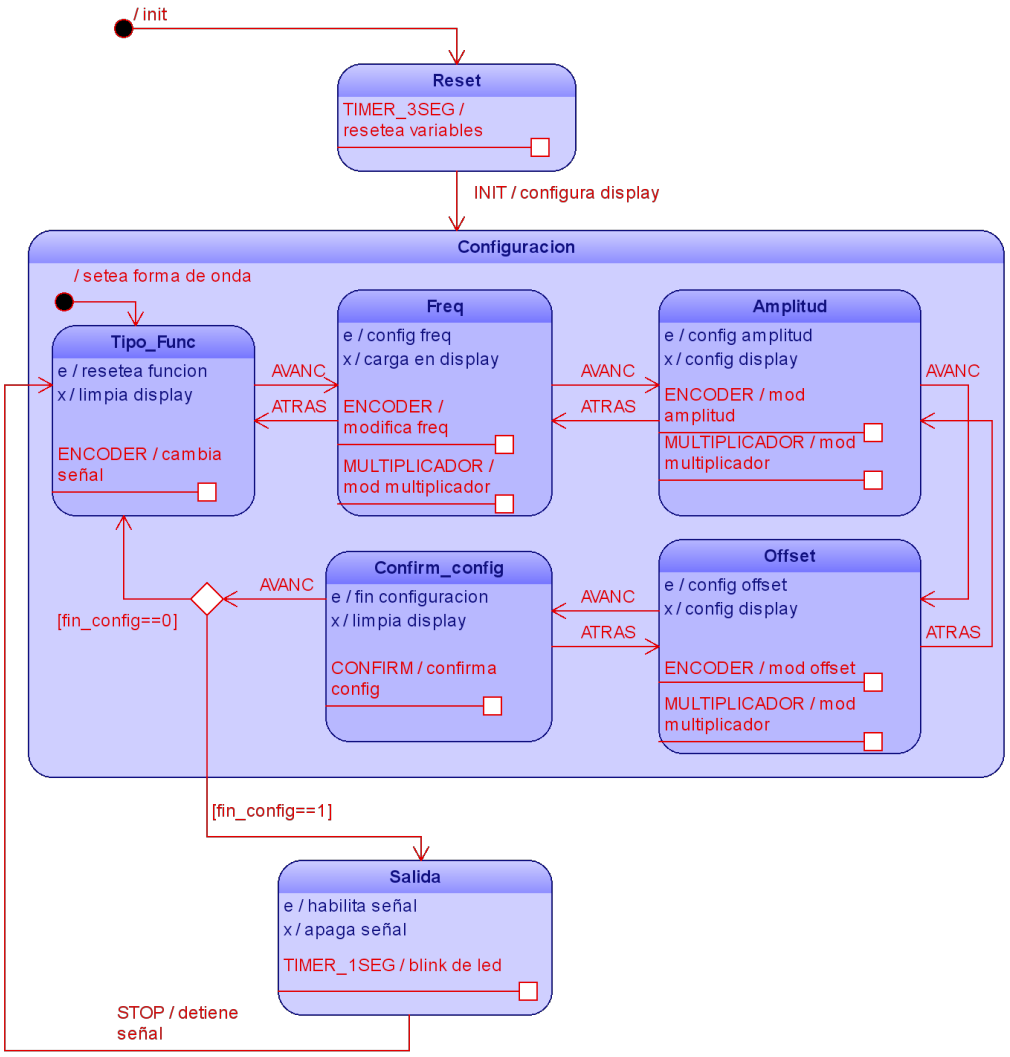

Figura 5: Máquina de estados

3.2 Diagramas de UML

La primera versión de programa no tuvo un modelado en especial, dicho programa fue realizado con intenciones de probar el funcionamiento en mayor medida del hardware. Luego como una segunda versión realizamos un modelado, orientado a objetos (programado en c, mediante estructuras). La estructura que elegimos para el modelado puede apreciarse en las siguientes figuras.

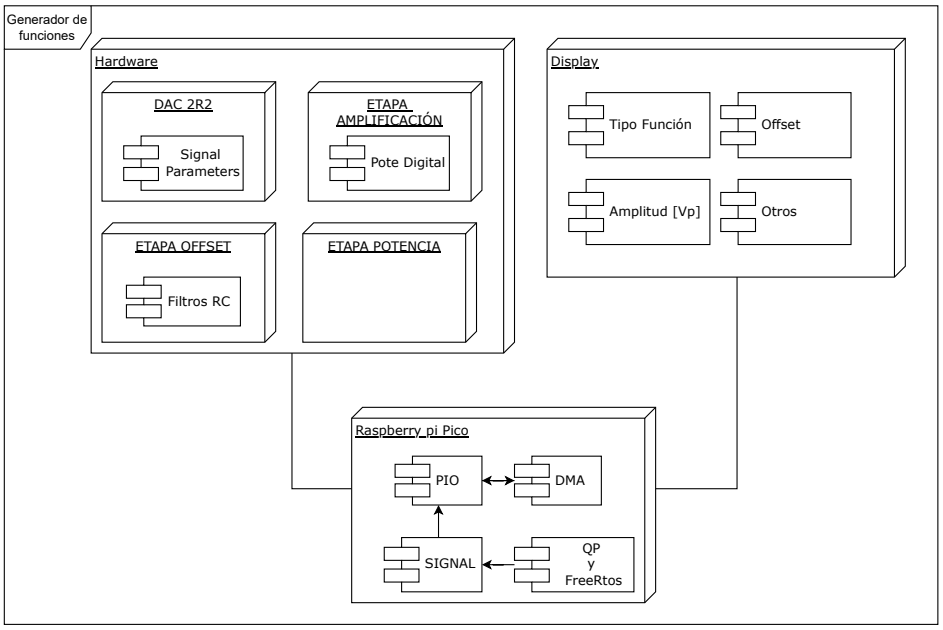

Figura 6: Diagrama de despliegue

Como podemos observar en el diagrama de despliegue, figura 6. El generador de funciones arbitrarias consta de tres bloques, *hardware, display, raspberry pi pico*. Detallamos a continuación que hace cada etapa:

- *Hardware:* consta de las etapas presentes en el circuito. Encontramos por un lado la generación de la señal mediante le convertidor digital a analógico (dac R2R), la etapa de amplificación (con el potenciómetro digital), etapa de offset (con los filtros RC) y la etapa de potencia, que no forma parte del modelado de software.
- *Display:* representa la interfaz entre el usuario y la máquina, en donde se pueden visualizar los parámetros modificados y el estado de la salida.
- *Raspberry pi Pico:* dicho bloque representa el microcontrolador que utilizamos para generar la señal e interactuar con el hardware. Los componentes que se nombran dentro son aquellos que tienen mayor relevancia. Encontramos los periféricos *PIO* y *DMA* que se encargan de generar la señal y transferir el vector a los GPIOs. Además se muestra el componente *QP y FreeRTOS* que representan una parte importante del programa, como ya fuimos detallando antes.

En la siguiente figura podemos observar como interactúa cada uno de los archivos (componentes) del software. La idea es que funcione como capas, comenzando desde la máquina de estados hasta el ciclo de trabajo del pwm necesario para generar un offset adecuado.

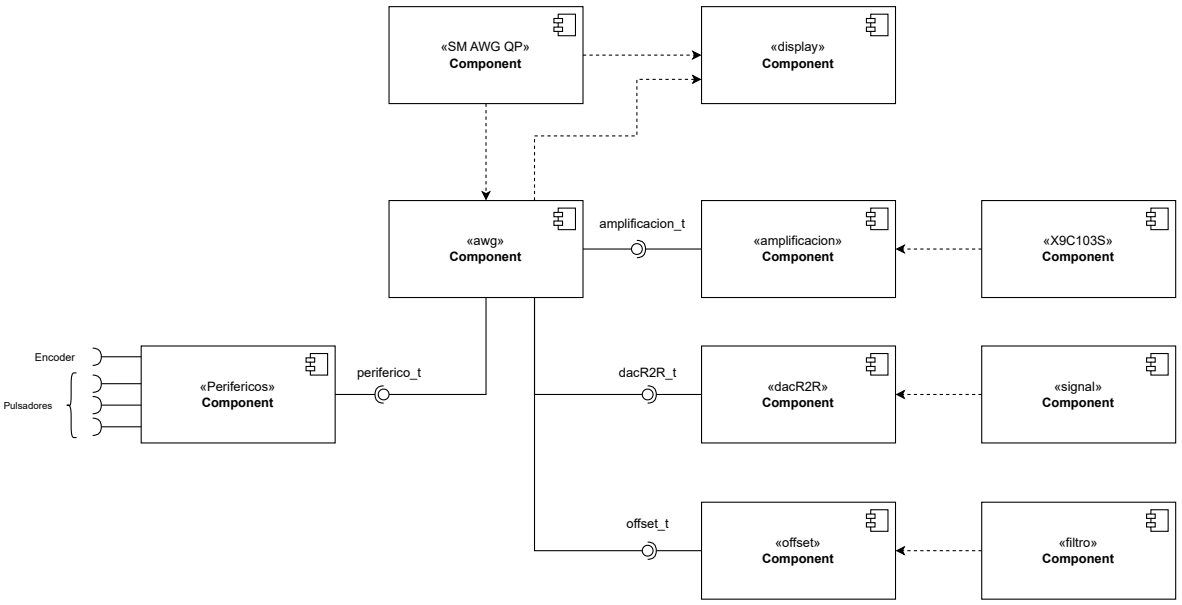

Figura 7: Diagrama de componente

En el diagrama de componentes observamos que para conectar cada componente con el componente awg, es necesario un objeto que se pasa por parámetro al archivo. Debido a esto en el componente awg, generamos un objeto que incluye un objeto de cada componente vecino. Para entender mejor esto presentamos la figura 8 en donde se observan cada una de la capas y como interactúan entre ellas con *Awg*.

La capa *Awg* funciona como un intermediario, pasando información a aquellas capas mas bajas y recibiendo información de capas mas altas. El archivo *Display* queda aparte del resto (a pesar de que pueda perder la modularidad), ya que puede ser accedido por el framework de QP y por *Awg*, esto fue realizado así debido a que nose encontró otra manera de no hacer repetitivo el código, lo correcto sería que quede por debajo de la capa *Periférico*. En violeta podemos encontrar las capas mas bajas, dado que son aquellas que están mas en relación con el hardware. Por ejemplo *X9C103S* se encarga de modificar el potenciómetro digital, esto modifica directamente la ganancia en tensión; *Filtros* modifica el ciclo de trabajo de cada pwm, logrando así un cambio en el valor medio y por ende modificando el offset de la señal final; *Signal* genera el buffer de la señal, que luego será cargado en los GPIOs.

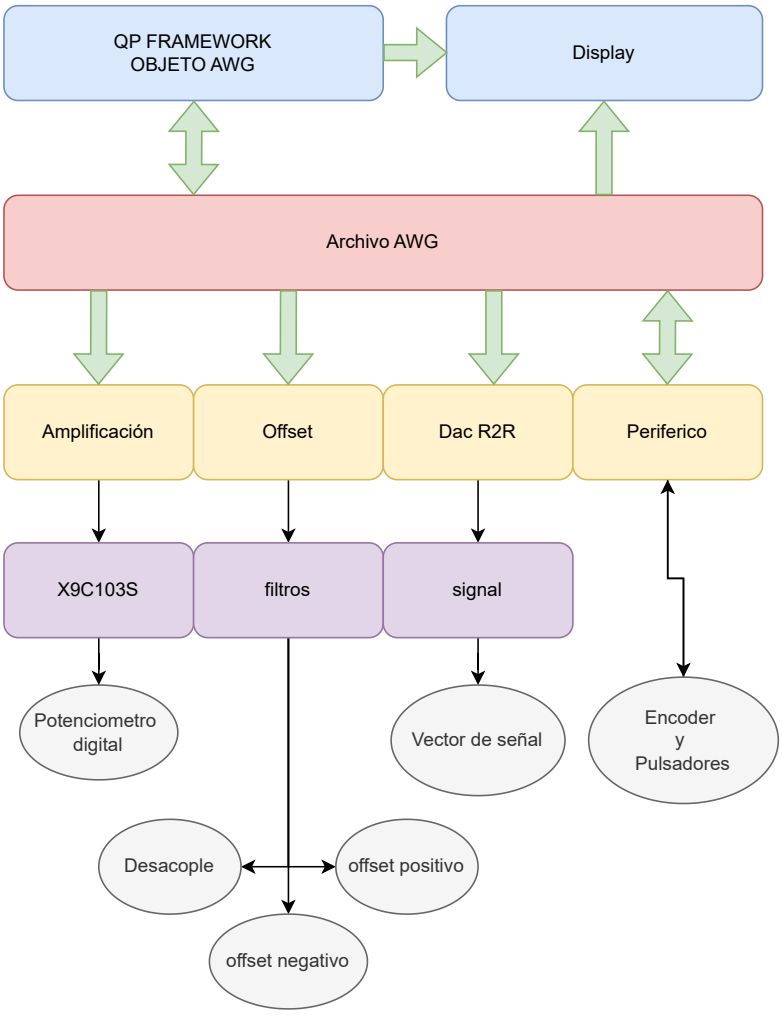

Figura 8: Diagrama de capas

Por último para terminar se presenta un diagrama de casos de usos, en donde un actor modificando el encoder y seleccionando la configuración con los pulsadores logra obtener una señal objetivo.

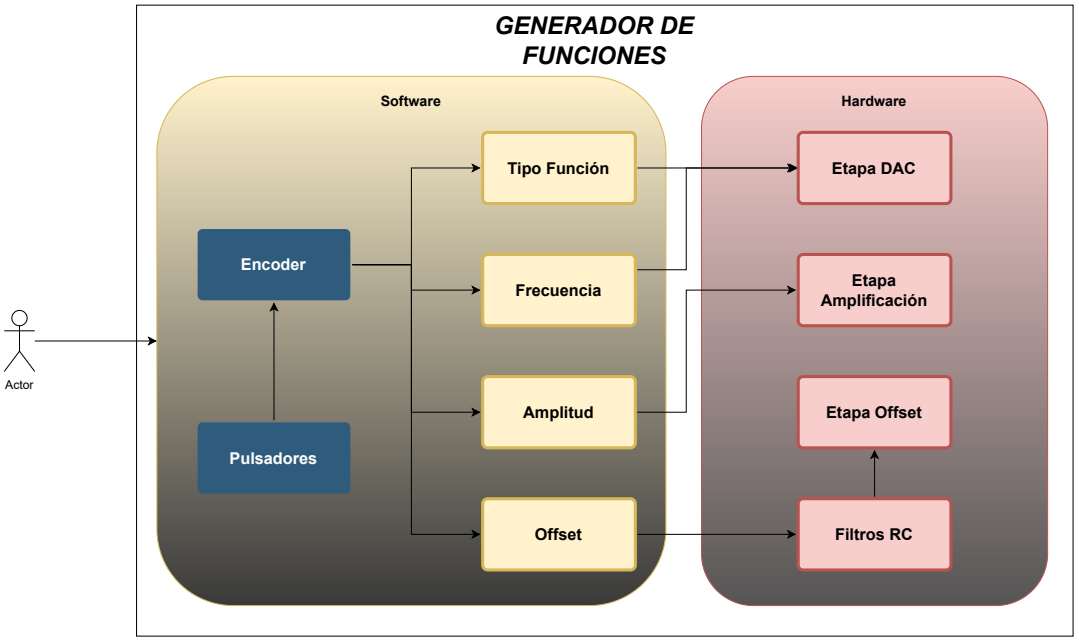

Figura 9: Diagrama de casos de usos

4 Programa Github

Se detalla el link en donde se puede ver el programa:
`https://github.com/Agustin586/TPs-ECA4/tree/main`

5 Circuito impreso

En la siguiente imagen podemos observar una imagen del circuito impreso que se realizó para ensayar el dispositivo.

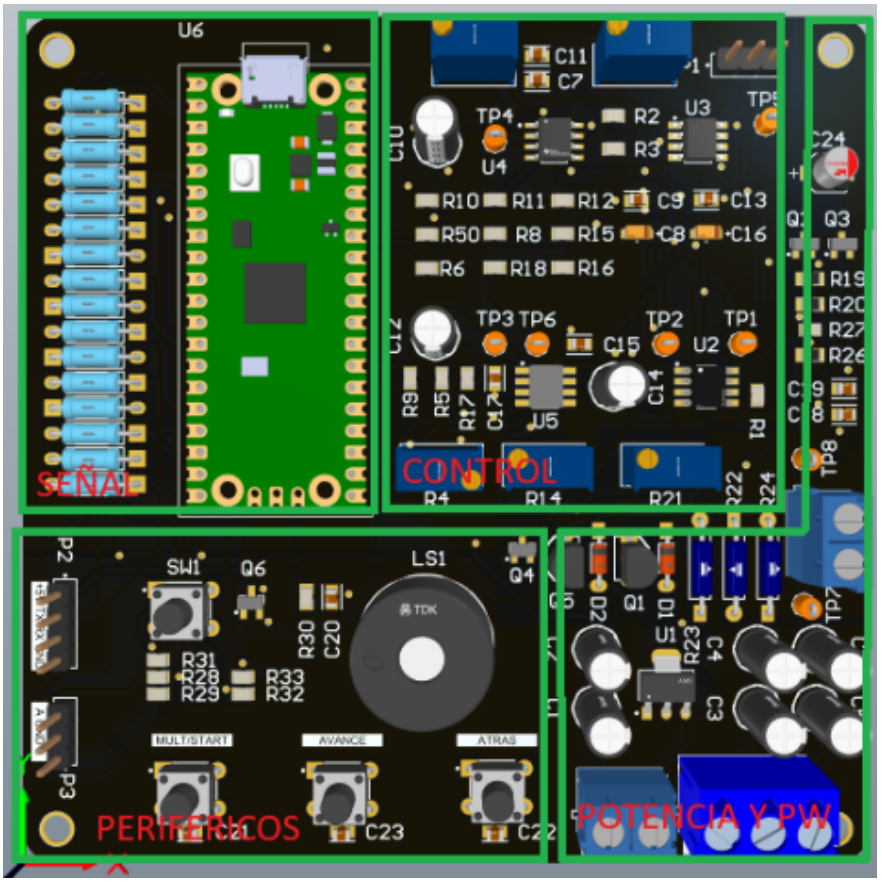

Figura 10: PCB del generador de funciones

5.1 Primeros ensayos

Mostramos a continuación los primeros ensayos realizados con la primera versión de la pcb (no es la que se muestra en la foto).

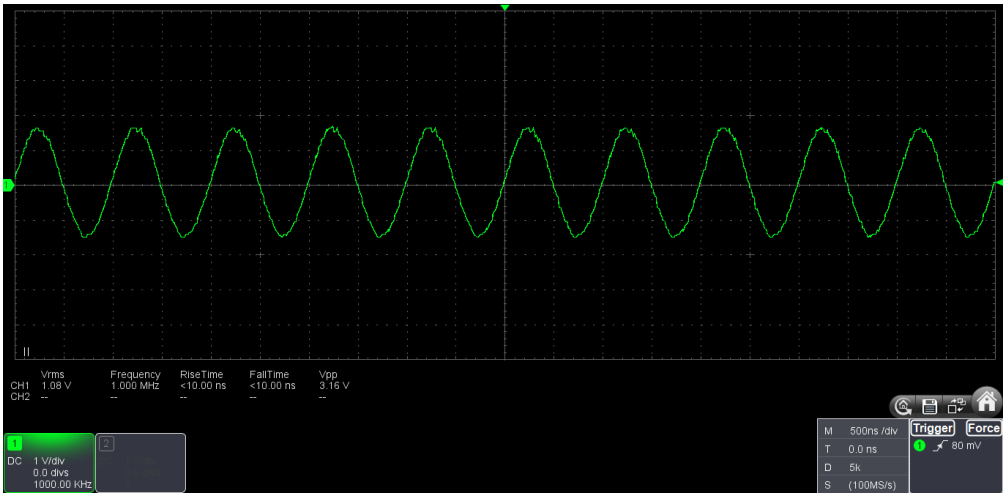

Figura 11: Senoidal a 1 MHz

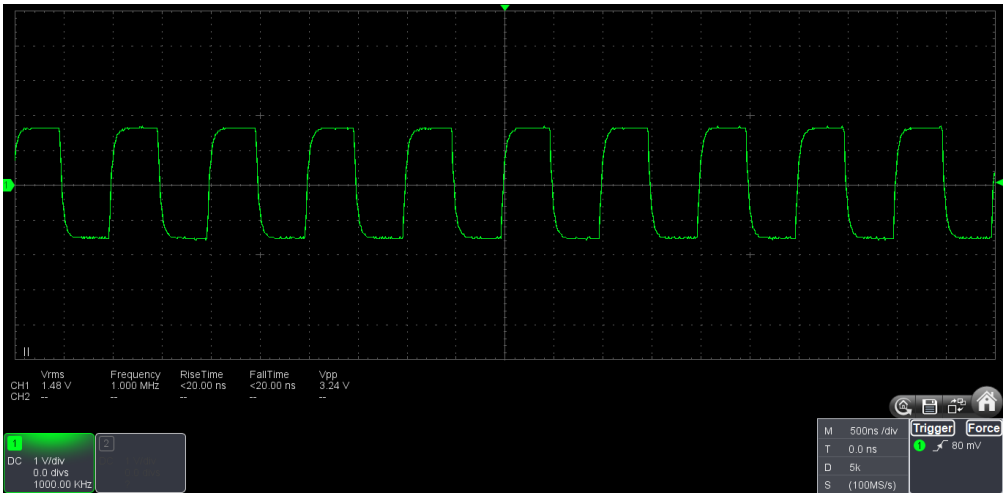

Figura 12: Cuadrada a 1 MHz

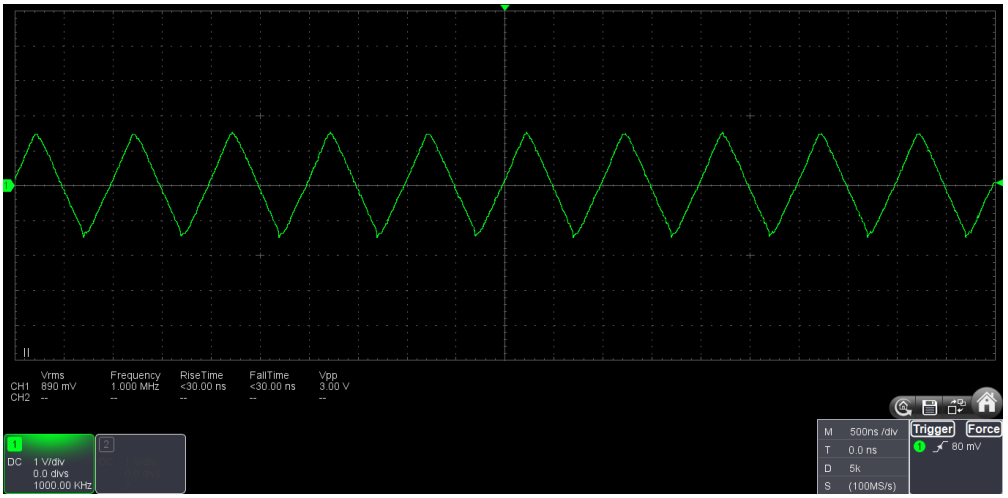

Figura 13: Triangular a 1 MHz