# Prueba del amplificador de audio en Ophyra.


Autor: Javier Martínez Fernández

Otoño 2021

# Contenido



## 1. Características generales del amplificador de audio.

La tarjeta Ophyra de Intesc integra un amplificador de audio con un Jack de 3.5 mm. Para trabajar con esta función es necesario utilizar el módulo DAC de la tarjeta, este módulo es un convertidor digital-analógico con salida de tensión de 12 bits. El DAC tiene dos canales que podemos utilizar para el amplificador de audio (pero solo debe conectarse una de las dos salidas analógicas DAC1 o DAC2, nunca al mismo tiempo).

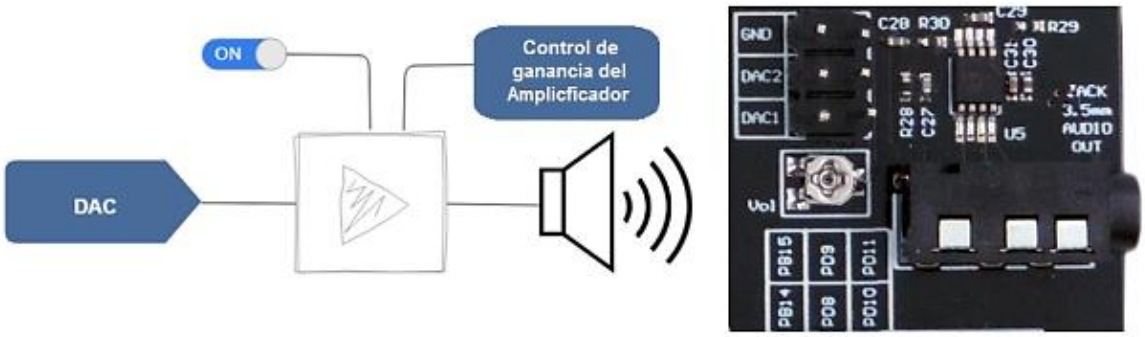



Figura 1. Jack de 3.5 mm del amplificador de audio y modulo DAC en la tarjeta Ophyra.

El amplificador de audio cuenta con un potenciómetro que regula la ganancia de salida, el usuario puede ajustar la potencia de salida en caso de ser necesario.

Tabla 1. Disposición de los pines del DAC del microcontrolador hacia el amplificador de Audio.

| Nombre | GPIO | Función | Descripción |
|---|---|---|---|
| AMP_ENABLE | PA6 | GPIO_ouput | GPIO de habilitación del amplificador. |
| DAC_1 | PA4 | DAC_OUT1 | Salida Analógica no. 1 |
| DAC_2 | PA5 | DAC_OUT2 | Salida Analógica no. 2 |

## 2. Waveform Audio Format (WAV o WAVE)

WAV es un formato de audio digital con o sin compresión de datos desarrollado por Microsoft e IBM que se utiliza para almacenar flujos digitales de audio en formato mono o estéreo a diversas resoluciones y velocidades de muestreo. Las extensiones de los archivos de este tipo son **.wav** y **.wave**.

Los archivos **wav** se componen de varias secciones y pueden contener el audio en diversas formas. El ejemplo descrito en la tabla 2 considera una grabación tipo PCM (muestreo y cuantificación uniformes).

Tabla 2. Ejemplo de la estructura de un archivo WAV.

| Sección | Tamaño (bytes) | Descripción |
|---|---|---|
| RIFF | 4 | Contiene los códigos ASCII de la letras: 'R', 'I', 'F' y 'F'.RIFF significa formato de archivo para el intercambio de recursos (*Resource Interchange File Format*). |
| | 4 | Entero positivo de 32 bits que almacena el tamaño de bytes del resto del archivo. Es decir, tamaño total del archivo – 8 (4 bytes de RIFF, y 4 bytes de este número). |
| WAVE | 4 | Estos 4 bytes indican que el archivo almacena audio, contienen los códigos ASCII de las letras: 'W', 'A', 'V' y 'E'. |
| | 4 | Estos 4 bytes contienen los caracteres 'f', 'm', 't' y '', indican que hay una subsección de "WAVE" denominada "fmt". Esta subsección almacena las características de la grabación. |
| | 4 | Entero positivo de 32 bits que indica el tamaño en bytes del resto del bloque. En caso de ser un número impar, será necesario considerar la existencia de un byte de relleno. |
| | 2 | Entero positivo de 16 bits que indica el tipo de grabación. Un 1 significa PCM. |
| | 2 | Número de canales. Cuando se usa un canal se habla de una grabación mono o monoaural; mientras que al uso de dos canales se le suele llamar estéreo. |
| | 4 | Frecuencia de muestreo expresada en Hz. |
| | 4 | Número promedio de bytes por segundo. Los programas para reproducir audio suelen estimar el tamaño de su ***buffer*** usando este dato. |
| | 2 | Alineamiento, corresponde con el número de bytes usados en el archivo por cada muestra (si es el caso, se consideran ambos canales). |
| | 2 | Bits por muestra. |
| data | 4 | Estos 4 bytes contienen los códigos ASCII de las letras: 'd', 'a', 't' y 'a', indican que a continuación están los datos. |

| | | |
|---|---|---|
| | 4 | Entero positivo de 32 bytes que indica el espacio en bytes que ocupan los datos. |
| | n | Datos. |

Los archivos WAV sin comprimir son grandes, por lo que es poco común compartir archivos WAV a través de Internet. Sin embargo, es un tipo de archivo de uso común, adecuado para conservar archivos de audio de primera generación de alta calidad, para usar en un sistema donde el espacio en disco no es una restricción, o en aplicaciones como la edición de audio, donde los tiempos implicados en comprimir y descomprimir los datos son una preocupación.

El uso del formato WAV tiene más que ver con su familiaridad y estructura simple. Debido a esto, continúa disfrutando de un uso generalizado con una variedad de aplicaciones de software, es comúnmente utilizado cuando se trata de intercambiar archivos de sonido entre diferentes programas.

## 3. Práctica - Uso del amplificador de audio: Utilizando el módulo DAC y la unidad DMA para reproducir archivos .wav

En prácticas anteriores ya habíamos utilizado el módulo DAC y la unidad DMA del microcontrolador de Ophyra para generar señales. Para esta práctica, utilizaremos algunos recursos de las practicas anteriores, pero enfocados a reproducir archivos .wav a través del amplificador de audio de la tarjeta.

**Proyecto a realizar:** reproducir un archivo de audio .wav en el amplificador de audio de Ophyra.

**Material.**

- Tarjeta Ophyra
- Memoria Micro SD (4 GB recomendado).
- 2 jumpers hembra-hembra.
- Bocina con cable auxiliar o audífonos con conector de 3.5 mm.

El contenido estará dividido en 2 secciones:

1. Explicación de la estructura general de la librería ***wav_ophyra.***
2. Desarrollo de la práctica.

**Nota:** la explicación de la librería tiene como objetivo mostrar al usuario la estructura y lógica que tiene el código, de esta forma, en caso de requerir modificar la librería para adaptarse a una aplicación más específica, podrá hacerlo sin tener mayor problema.

## 3.1. Librería wav_ophyra.

**Encabezado del código:**

```
"""
----- REPRODUCTOR DE AUDIO WAV -----
- WAV_FILE ---> Nombre del archivo .wav
- seg --------> Tiempo en segundos para reproducir el audio
"""

from pyb import DAC
from pyb import delay
```

Importamos las clases ***DAC*** y ***delay*** de la librería ***pyb***. Ambas clases ya las hemos utilizado anteriormente, su uso será similar en esta práctica.

**Configuración de la clase *wav* y función *play*:**

En la siguiente sección creamos la función ***play*** dentro de la clase ***wav,*** esta función tendrá cuatro parámetros de entra (nombre del archivo, segundos de duración, activación del DAC1 o DAC2 y tasa de muestreo en Hz).

```
class wav:

    def play(archivo, seg, canal, muestreo):
        #-------------CONFIGURACIÓN DE AUDIO---------------
        dac = DAC(canal, bits=8)        #Inicialización del DAC
en modo de 8 bits / 8 bits de resolución.
```

Primero creamos un objeto llamado ***dac*** de la clase ***DAC*** y como argumentos tendremos el ***canal***, para activar el DAC que vayamos a utilizar. Y en el segundo argumento configuraremos el DAC para una resolución de 8 bits.

Es importante señalar que el módulo DAC de Ophyra tiene una resolución de hasta 12 bits, sin embargo, algunos convertidores de audio solo muestran la opción de 8 bits y después saltan a 16 bits, debido a esto, se decidió dejarlo en 8 bits de resolución. En caso de ser necesario se puede modificar este valor, siempre y cuando no se supere el límite del modulo DAC de la tarjeta.

**Inicio de reproducción:**

```
1.  #------- Inicio de reproducción -------
2.          ini = 44  #Se ignora el encabezado del .WAV
3.          for i in range(seg+1):
4.              wav = open(archivo, "rb") #Apertura del archivo de
    audio
5.              pos = wav.seek(ini)
```

Declaramos una variable ***ini*** con un valor inicial igual a 44, esta variable nos permitirá ir moviendo el curso del archivo a una nueva posición conforme avance el programa, lo iniciamos en 44 para ignorar el encabezado del archivo y pasar directamente al primer segundo de reproducción de audio. En la siguiente línea agregamos el bucle ***for*** con un rango desde 0 hasta el número de segundos que el usuario ingreso en el programa principal.

Dentro del bucle abrimos el archivo en modo se sólo lectura **"rb"** y lo asignamos al objeto ***wav***. En la siguiente línea hacemos el salto a la posición donde iniciara la lectura del archivo (definido por ***ini***) y lo asignamos al objeto ***pos***.

**Declaración de buffers de datos:**

```
1.  #---- Declaración de buffers de datos ----
2.              wav_samples = bytearray(muestreo)  #Matriz de bytes
    equivalentes a 1 seg de reproducción
3.              wav_samples_mv = memoryview(wav_samples)
```

Creamos un objeto con el nombre ***wav_samples*** de tipo bytearray de un tamaño igual al ***muestreo*** que ingreso el usuario, que, de acuerdo con la configuración elegida, será equivalente a 1 segundo de reproducción de audio. En la siguiente línea, ***memoryview*** expone la interfaz del búfer a nivel de C como un objeto Python, que luego puede pasarse como cualquier otro objeto, este es asignado a ***wav_samples_mv***.

**Inicio de la reproducción del archivo:**

```
#---- Escritura en buffer DAC ----
try:
    num_read =wav.readinto(wav_samples_mv) #Se escriben los datos del .WAV al buffer de datos
    if num_read == 0:
        pos = wav.seek(ini) #Se ignora el encabezado del .WAV
    else:
        dac.write_timed(wav_samples_mv[:num_read], muestreo, mode=DAC.CIRCULAR) #Se escriben los datos del buffer al DAC
except (KeyboardInterrupt, Exception) as e:
    print("caught exception {} {}".format(type(e).name, e))
```

Dentro de ***try***, usando el método ***readinto*** leemos los datos de ***wav_samples_mv*** y los asignamos al objeto ***num_read***, que será el búfer de datos que usaremos en el DAC. En las siguientes líneas tenemos un ***if*** y un ***else.*** Dentro del ***if*** verificaremos que el búfer no este iniciando en 0, de ser así, haremos un salto para ignorar el encabezado del archivo. Dentro del ***else*** activamos el DAC y el módulo DMA con el método ***write_timed***. Como lo habíamos visto en prácticas anteriores, este método requiere tres parámetros de configuración para su funcionamiento.

- **Primer parámetro:** espacio de memoria RAM o búfer donde se encuentran los datos que se enviaran al DAC.
- **Segundo parámetro:** temporizador que indica el momento en el que el DMA toma un dato nuevo y lo envía hacia el DAC.
- **Tercer parámetro:** modo de funcionamiento del DMA y el DAC, en este caso será un modo circular que se refiere a que el DMA tomará los datos desde el primero hasta el último, para después volver a tomar el primer dato, es decir, se ciclará en un círculo infinito de recolección de datos. Recordemos que en el modo circular los datos son tomados del búfer y no le es permitido al DMA salirse de ese espacio de memoria asignado.

Dentro del ***except*** tendremos un mensaje en caso de ocurrir una interrupción por el usuario, nótese que una interrupción generada por el usuario es señalizada generando la excepción ***keyboardInterrupt***. Esta instrucción permite al usuario interrumpir el programa en cualquier momento, usando **ctrl + c** en el teclado. Sin embargo, el funcionamiento del DAC queda activado y gestionado en segundo plano por el DMA, es por esto que se seguirá escuchando el fragmento de audio en la posición donde se quedó.

```
1.      print("Reproduciendo... " + str(i) +" seg")
2.      ini = ini + muestreo #Se mueve el cursor para leer el
    siguiente segundo del archivo .WAV
3.      wav.close() #Se cierra el archivo de audio
4.      delay(1000)  #Espera 1 seg, tiempo para reproducir el sonido
    en el amplificador
5.  dac.deinit()  #Se desactiva el DAC
```

Para visualizar el tiempo que lleva la reproducción del audio agregamos un mensaje con el número de ciclos (en este caso, tiempo en segundos) que han transcurrido desde el inicio. En la siguiente línea sumamos ***muestreo*** a ***ini*** para ir moviendo el cursor a la siguiente posición del archivo.

Para finalizar, cerramos el archivo de audio con el método ***.close()*** y colocamos un ***delay(1000)*** equivalente a un segundo, esto es para darle tiempo al DAC y pueda pasar el audio al amplificador de la tarjeta y reproducirlo en la bocina. Como se había mencionado, el funcionamiento del DAC queda activado, para evitar esto, utilizamos el método ***.deinit()*** que desactiva el DAC una vez que termino el tiempo de reproducción.

## 3.2. Desarrollo de la práctica.

**Paso 1.** Conectar el módulo DAC con el amplificador de audio de Ophyra.

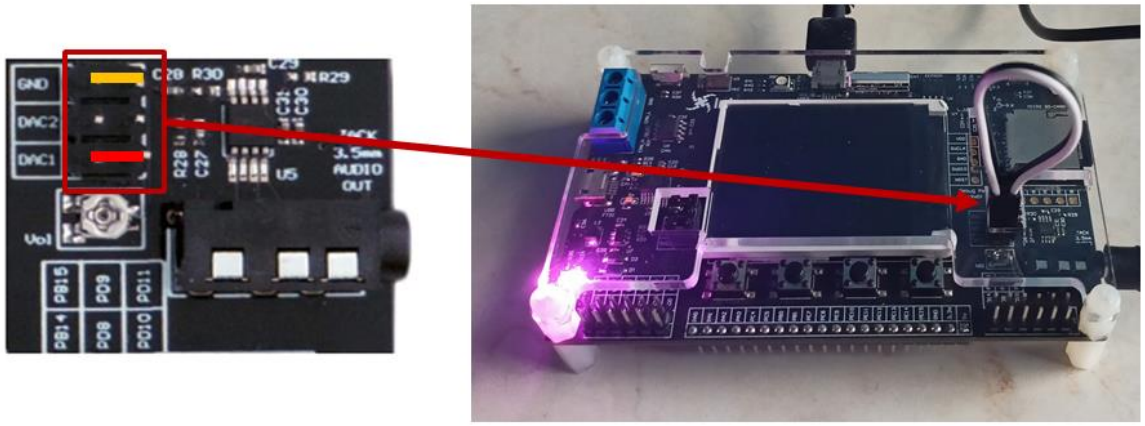



Figura 1. Configuración física entre el Jack de 3.5 mm del amplificador de audio y el módulo DAC en la tarjeta Ophyra.

Iniciaremos con las conexiones físicas en la tarjeta. Adicionalmente a la configuración en el código del programa, será necesario conectar el Jack de 3.5 mm (amplificador de audio) a los pines del módulo DAC. Para hacer esto se pueden utilizar dos cables jumper hembra-hembra y conectar los pines **GND** y **DAC1**, como se muestra en la imagen. **Recuerde que no se debe conectar el DAC1 y DAC2 al mismo tiempo**.

**Paso 2.** Descargar la librería y los archivos adicionales para la práctica.

Para realizar esta práctica, es necesario utilizar una memoria Micro SD en la tarjeta Ophyra. En el siguiente enlace encontrara una carpeta .zip con la librería y los archivos necesarios para realizar esta práctica.

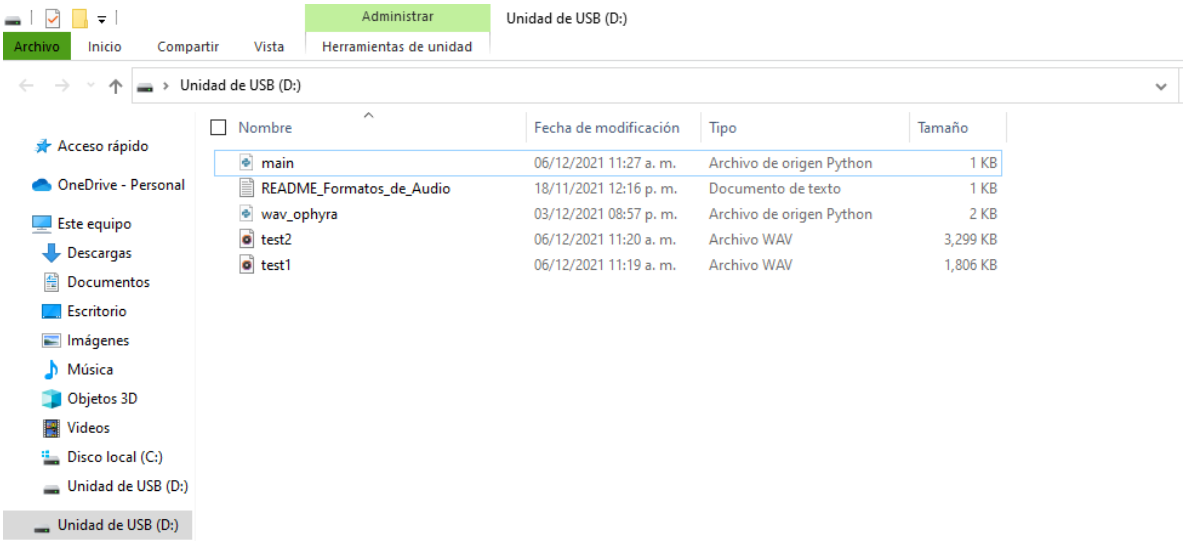
https://github.com/Intesc-Ingenieria/Micropython-AUDIO-Ophyra/tree/main/amplificador_ophyra

**Contenido de la carpeta.**

- Librería wav_ophyra
- 2 archivos de audio .wav con diferente tasa de muestreo.
- Archivo de texto con información de los audios .wav (es importante que lea esta información).

**Paso 3**. Conectar la tarjeta Ophyra a la computadora y cargar los archivos en la memoria Micro SD (utilizar el puerto USB-OTG de la tarjeta).

Al final tendremos cinco archivos como se muestra en la imagen, incluyendo el programa principal "**main.py**".



**Paso 4.** Realizar el código para el programa principal.

Abrimos el archivo "**main.py**" e iniciamos con la programación. Primero importamos la clase ***wav*** de la librería ***wav_ophyra*** y agregamos un pequeño encabezado que se mostrara al iniciar el programa.

```
from wav_ophyra import wav

print("   ---------------------------------")
print("   | REPRODUCTOR DE AUDIO - OPHYRA |")
print("   ---------------------------------")
```

En el siguiente apartado estará la configuración del audio. Primero agregamos el objeto ***archivo*** y le asignamos el nombre del audio .wav que vamos a utilizar, en este caso será el "test1.wav". En la línea siguiente asignamos el número de segundos que tendrá la reproducción, esto se colocará en el objeto ***seg***. Después vamos a seleccionar el DAC que utilizaremos, puede ser el DAC1 o DAC2, no olvide que debe ser el mismo que en el **paso 1** se conectó de forma física en la tarjeta, esto se guardará en el objeto ***canal***. Y, por último, creamos un objeto con el nombre ***muestreo*** el cual será nuestra tasa de muestreo en Hz, aquí recomendamos dejarlo en 8 KHz o 16 KHz, pero se puede modificar dependiendo de las necesidades del usuario (sólo no debe olvidar que la tasa de muestreo debe ser la misma en el archivo .wav), en este caso será de 8 KHz.

```
#-------------CONFIGURACIÓN DE AUDIO---------------
archivo = "test1.wav"     #Archivo .WAV a reproducir
seg = 120                 #Segundos que tendra la reproducción
del archivo .WAV
canal = 1                 #Inicialización del DAC (DAC1 o DAC2)
muestreo = 8000           #Tasa de muestreo en Hz (recomendado a
8 KHz o 16 KHz)
```

Para hacer que el audio se repita una vez terminado el tiempo, agregamos un bucle *while*. Dentro del bucle, llamamos a la función ***play*** de la clase ***wav*** y pasamos como parámetros el nombre del archivo, los segundos de duración, canal que usaremos (DAC1) y tasa de muestreo del archivo. Para visualizar mejor el inicio y final de la reproducción, agregamos un mensaje antes y después de llamar a la función.

```
#Agregamos un bucle while para ciclar la reproducción del audio
while 1:
    print("\n -------- INICIO DE AUDIO --------")
    wav.play(archivo,seg,canal,muestreo)
    print("--------- FIN DE AUDIO ---------")
```

Así debe quedar el programa "**main.py**".

```
from wav_ophyra import wav

print("   ---------------------------------")
print("   | REPRODUCTOR DE AUDIO - OPHYRA |")
print("   ---------------------------------")

#-------------CONFIGURACIÓN DE AUDIO---------------
archivo = "test1.wav"  #Archivo .WAV a reproducir
seg = 120                #Segundos que tendra la reproducción
del archivo .WAV
canal = 1               #Inicialización del DAC (DAC1 o DAC2)
muestreo = 8000         #Tasa de muestreo en Hz (recomendado a
8 KHz o 16 KHz)

#Agregamos un bucle while para ciclar la reproducción del audio
while 1:
    print("\n -------- INICIO DE AUDIO --------")
    wav.play(archivo,seg,canal,muestreo)
    print("--------- FIN DE AUDIO ---------")
```

**Paso 5.** Abrir la terminal PuTTY y ejecutar el programa con **ctrl+d** en el teclado.

Si no recuerda como configurar la terminal PuTTY puede ingresar en el siguiente enlace https://youtu.be/3YFkl92axRU y dirigirse a la sección “Configuración PuTTY”.

## 3.3. Resultados.

En el siguiente enlace puede ver los resultados de la práctica, así como una explicación rápida de todo lo que vimos en este tutorial.


https://www.youtube.com/watch?v=ARA1vfev_-E

# 4. Conclusiones.

Gracias a la librería ***wav_ophyra*** es posible desarrollar varias aplicaciones que requieran salida de audio. Hay dos aspectos principales a tener en cuenta, el primero es relacionado al módulo DAC, la configuración física en la tarjeta y la configuración en el programa deben ser las mismas. Y segundo, el formato del audio .wav debe ser el mismo al que se describe en el código del programa (resolución, tasa de muestreo y formato mono canal).

**Nota:** para ser leído, el archivo .WAV también debe tener los mismos parámetros que se mencionaron anteriormente (número de bits de resolución, tasa de muestreo y formato mono canal). Esto puede configurarse al momento de convertir el archivo de audio al formato .WAV, en el siguiente enlace tenemos un ejemplo de un convertidor de audio en línea gratuito para hacer este trabajo.

https://audio.online-convert.com/es/convertir-a-wav