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 algunas 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.

Material extra.

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

Para realizar esta práctica, es necesario utilizar una memoria Micro SD en la tarjeta Ophyra y cargar los archivos de programa principal *main.py*, librería *wav_ophyra* y audio (.wav) directamente a esta.

La práctica consistirá en reproducir un archivo .wav a través del amplificador de audio. Para esto, lo dividiremos en 2 partes, primero vamos a explicar la estructura general de la librería *wav_ophyra*, y posteriormente veremos el programa principal y cómo conectar físicamente en la tarjeta el módulo DAC al amplificador.

Librería *wav_ophyra*.

Encabezado del código:

```
1  """
2  ----- REPRODUCTOR DE AUDIO WAV -----
3
4  - WAV_FILE ---> Nombre del archivo .wav
5  - seg --------> Tiempo en segundos para reproducir el audio
6  """
7
8  from pyb import DAC
9  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.

En la siguiente sección creamos la función **play** con dos parámetros de entra (nombre del archivo y los segundos de duración que tendrá la reproducción de audio).

Configuración de audio / objetos y variables:

```
11  def play(WAV_FILE, seg):
12
13      #------------CONFIGURACIÓN DE AUDIO---------------
14      dac = DAC(1, bits=8)        #Inicialización del DAC1 en modo de 8 bits / 8 bits de resolución.
15      SAMPLE_RATE_IN_HZ = 8000    #Tasa de muestreo en Hz (recomendado a 8 KHz o 16 KHz)
16
```

Primero creamos un objeto llamado “dac” de la clase DAC y como argumentos, introduciremos el número 1, para activar el canal 1. Y en el segundo argumento configuraremos el DAC para una resolución de 8 bits. En la siguiente línea creamos un objeto con el nombre “SAMPLE_RATE_IN_HZ” 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 deben olvidar que esta misma tasa de muestreo debe tener el archivo .wav).

Cuerpo del programa:

```
17      print("-------- INICIO DE AUDIO --------")
18      lon = 44  #Se ignora el encabezado del .WAV
19      for i in range(seg+1):
20          wav = open(WAV_FILE, "rb") #Apertura del archivo de audio
21          pos = wav.seek(lon)
22
```

Agregamos un mensaje de inicio para que se muestre en la terminal. En la siguiente línea declaramos una variable “lon” 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 “lon”) y lo asignamos al objeto “pos”.

Declaración de buffers de datos:

```
23          #---- Declaración de buffers de datos ----
24          wav_samples = bytearray(SAMPLE_RATE_IN_HZ)  #Matriz de bytes equivalentes a 1 seg de reproducción
25          wav_samples_mv = memoryview(wav_samples)
```

Creamos un objeto con el nombre “wav_samples” de tipo bytearray de un tamaño igual a “SAMPLE_RATE_IN_HZ” 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:

```
17      print("-------- INICIO DE AUDIO --------")
18      lon = 44  #Se ignora el encabezado del .WAV
19      for i in range(seg):
20          wav = open(WAV_FILE, "rb") #Apertura del archivo de audio
21          pos = wav.seek(lon)
22
23          #---- Declaración de buffers de datos ----
24          wav_samples = bytearray(SAMPLE_RATE_IN_HZ)  #Matriz de bytes equivalentes a 1 seg de rep
25          wav_samples_mv = memoryview(wav_samples)
26
27          #---- Inicio de reproducción del archivo ----
28          try:
29              num_read =wav.readinto(wav_samples_mv) #Se escriben los datos del .WAV al buffer de
30              if num_read == 0:
31                  pos = wav.seek(lon) #Se ignora el encabezado del .WAV
32              else:
33                  dac.write_timed(wav_samples_mv[:num_read], SAMPLE_RATE_IN_HZ, mode=DAC.CIRCULAR)
34          except (KeyboardInterrupt, Exception) as e:
35              print("caught exception {} {}".format(type(e).name, e))
36
37          print("Reproduciendo... " + str(i) +" seg")
38          lon = lon + SAMPLE_RATE_IN_HZ  #Se mueve el cursor para leer el siguiente segundo del ar
39          wav.close() #Se cierra el archivo de audio
40          delay(1000)  #Espera 1 seg, tiempo para reproducir el sonido en el amplificador
41
42      print("--------- FIN DE AUDIO ---------")
43      dac.deinit()  #Se desactiva el DAC
```

Dentro de *try*, usando el método “readinto” leemos los datos del “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 modulo DMA con el método "write_timed". Como lo habíamos visto en la práctica anterior, este método requiere tres parámetros de configuración para su funcionamiento.

1. Primer parámetro: espacio de memoria RAM o bufer donde se encuentran los datos que se enviaran al DAC.
2. Segundo parámetro: temporizador que indica el momento en el que el DMA toma un dato nuevo y lo envía hacia el DAC.
3. 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 circulo infinito de recolección de datos. Recordemos que en el modo circular los datos son tomados del buffer 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.

```
37          print("Reproduciendo... " + str(i) +" seg")
38          lon = lon + SAMPLE_RATE_IN_HZ  #Se mueve el cursor para leer el siguiente segundo del
39          wav.close() #Se cierra el archivo de audio
40          delay(1000)  #Espera 1 seg, tiempo para reproducir el sonido en el amplificador
41
42      print("--------- FIN DE AUDIO ---------")
43      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, número de segundos) que han transcurrido desde el inicio. En la siguiente línea sumamos "SAMPLE_RATE_IN_HZ" a "lon" 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 termino el tiempo de reproducción.

Programa principal - ejemplo.

Para el programa principal haremos que se reproduzca un archivo .wav en amplificador de audio de la tarjeta. En la primera línea vamos a importar la librería *wav_ophyra* que vimos anteriormente, esta librería no permitirá hacer la mayor parte del trabajo.

Configuración de Audio:

```
1   import wav_ophyra  #Importamos la libreria del reproductor WAV
2
3   #-------------CONFIGURACIÓN DE AUDIO---------------
4   WAV_FILE = "test1.wav"        #Archivo .WAV a reproducir
5   seg = 120                    #Segundos que tendra la reproducción del archivo .WAV
6
7   #Agregamos un bucle while para ciclar la reproducción del audio
8   n = 0
9   while n < 1:
10      wav_ophyra.play(WAV_FILE,seg)
```

Ingresamos el nombre del archivo .wav y lo guardamos en el objeto “WAV_FILE”. En la siguiente línea ingresamos los segundos que tendrá la reproducción del audio y se guardaran en el objeto “seg”.

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 librería “wav_ophyra” y pasamos como parámetros el nombre del archivo y los segundos de duración.

Configuración física de la tarjeta.

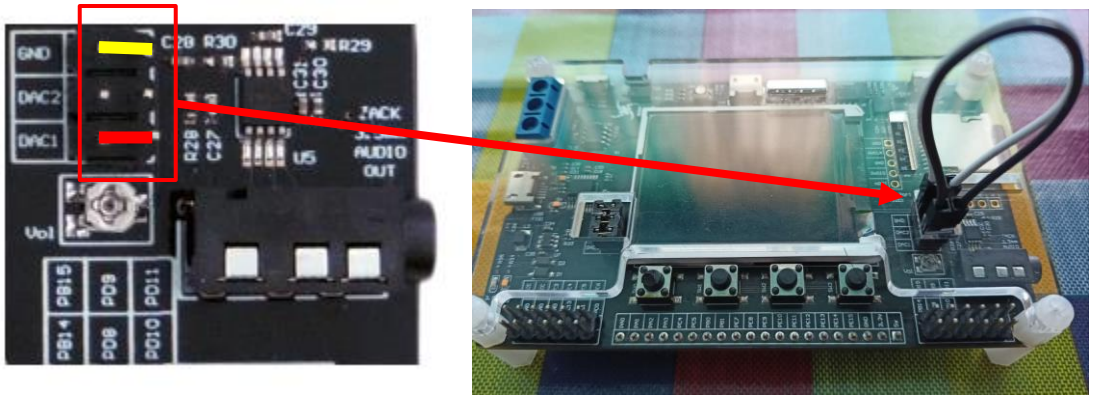

Figura 1. Configuración física entre el Jack de 3.5 mm del amplificador de audio y el módulo DAC en la tarjeta Ophyra.

Adicionalmente a la configuración en el código del programa, será necesario conectar el Jack de 3.5 mm 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 figura 2. **Recuerde que no se debe conectar el DAC1 y DAC2 al mismo tiempo**.