Implementación de protocolo I2S en tarjeta de desarrollo Ophyra usando Micropython

Jorge Corichi Herrejón
Miguel Angel Fierro Gutiérrez

Septiembre 2021

1. Introducción

La tarjeta de desarrollo Ophyra cuenta con un micrófono digital MP45DT02. Este micrófono devuelve la información en forma de pulsos de densidad modulada. Es posible recuperar datos desde el micrófono utilizando un protocolo de comunicación síncrono, en tanto que el micrófono requiere de una señal de reloj. Es posible realizar esto bien con SPI o con I2S, siendo este último un protocolo de comunicación especializado para la transmisión de datos de audio.

Debido a la arquitectura y disposición física actual de la tarjeta Ophyra,únicamente es posible hacer uso del I2S para la recuperación de datos. El presente reporte se enfoca en los cambios que deben hacerse dentro de los archivos de compilación de Micropython para poder hacer uso de un módulo especializado y del hardware pertinente. Primeramente se abordarán los cambios realizados en estos archivos y, posteriormente, se muestra a detalle el uso de la librería.

Cabe mencionar que esta funcionalidad se encuentra catalogada como *Tech Preview*, por lo que la estabilidad y funcionalidad del módulo *machine_i2s* no están probadas por completo.

2. Compilación de Micropython

La activación de las funcionalidades de I2S dependen de dos modificaciones dentro del arhchivo *mpconfigboard.h*, usado para compilar el firmware de Micropython específico de la tarjeta Ophyra. Al agregar la línea

```
#define MICROPY_HW_ENABLE_I2S (1)
```

la tarjeta Ophyra podrá utilizar el módulo *machine_I2S*. No obstante, también deben activarse los buses de manera individual para usarlos con el módulo previamente mencionado. Si bien el microcontrolador de la tarjeta Ophyra cuenta con los buses I2S2 e I2S3 de acuerdo con su hoja de especificaciones, el micrófono se encuentra acoplado al bus de I2S2, por lo que se ha agregado la línea

```
#define MICROPY_HW_I2S2 (1)
```

Esto permitirá el uso del identificador adecuado usado en al crear instancias de la clase I2S, que se explica más a detalle en siguientes secciones. Con la adición de estas líneas, se debe compilar el firmware como se explica en el *Manual de configuración del entorno de construcción de MicroPython en Windows 10* y en el *Manual de compilación de módulos en C (usermod) para la tarjeta Ophyra*

3. Importando la librería I2S

Para comprobar que la compilación del firmware es correcta, se deben probar todos los módulos de interés que se hayan especificado en el archivo *mpconfigboard.h*. En este caso, la librería *machine_i2s*.

Importar la librería desde Micropython sólo requiere del comando

```
from machine import I2S
```

ya sea en un archivo extensión *.py* o bien, en una terminal de *PuTTY* en comunicación serial abierta con la Ophyra. En caso de mostrar un error al importar, se debe verificar nuevamente la compilación del firmware.

Si no se muestra ningún error se puede probar generar una instancia de la clase I2S, esto se hace para comprobar que el bus de I2S2 se está identificando correctamente. En caso de obtener un error en este paso, se debe verificar una vez más la compilación del firmware.

Una vez realizadas las pruebas correspondientes, la librería está lista para usarse y se tiene acceso a todos los métodos de la clase, entre ellos: *I2S.init*, *I2S.readinto*, *I2S.shift*, etc. Cuya funcionalidad puede ser revisada a detalle en el siguiente enlace.

4. Uso de la librería I2S

Para activar la comunicación a través de I2S, se deben configurar ciertos parámetros dentro de la inicialización del módulo. Utilizando *I2S.init*, se da de alta el identificador del bus, en este caso 2, así como los pines en los cuales se encuentran las conexiones al micrófono, así como la tasa y tamaño de muestreo, el formato del audio y tamaño del buffer interno. Estos argumentos se pueden consultar más a detalle en la documentación oficial de Micropython en cuanto a la Clase I2S.

Debe distinguirse que este modulo permite recepción y envío de datos a través de I2S. Para la escritura se utiliza el método *I2S.write()*, mientras que para recibir datos se utiliza *I2S.readinto()*. Esta última función debe recibir un objeto que permita el uso del protocolo buffer, como lo es un *bytearray* o un *array*.

5. Recopilación de audio

El micrófono MP45DT02 de la tarjeta Ophyra recibe audio del ambiente y da como salida una cadena de bits de acuerdo al formato PDM. PDM es una forma de modulación en la que la densidad de 1's o 0's en la cadena de bits es directamente proporcional a la amplitud de la señal de audio.

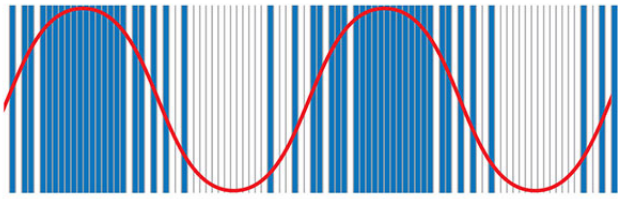

Figura 1: Representación del formato PDM donde los sectores azules corresponden a 1's y los sectores blancos a 0's

Para escuchar audio directamente del amplificador de audio de la tarjeta es necesario realizar la conversión del formato PDM al formato PCM antes de escribir los datos al DAC. El algoritmo de conversión consta de 2 pasos generales: Decimación y Filtrado

5.1. Decimación

El proceso de decimación consiste en la reducción de muestras a razón de un factor M. Esta reducción de muestras permite contrarrestar el sobremuestreo generado para la adquisición de señal PDM, en tanto que rechaza cierta cantidad de muestras dependiendo del factor de decimación. Este proceso se realiza sobre un stream de bits y el factor M se escoge con base a la tasa de muestreo y la frecuencia del reloj suministrada al micrófono. A continuación se presenta una tabla que puede

servir de referencia para el cálculo del factor de decimación. Una vez la cadena bits suministrada por el micrófono ha sido decimada, se puede proceder al filtrado de la señal.

Decimation factor	PDM clock frequency	PCM sample rate
128	1.024 MHz	8 kHz
	2.048 MHz	16 kHz
	3.072 MHz	24 kHz
80	1.280 MHz	16 kHz
64	1.024 MHz	16 kHz
	2.048 MHz	32 kHz
	3.072 MHz	48 kHz
48	768 kHz	16 kHz
32	512 kHz	16 kHz
24	384 kHz	16 kHz
16	256 kHz	16 kHz

Figura 2: Factores de decimación y frecuencias correspondientes

5.2. Filtrado

Debido a que el formato PCM suele devolver datos correspondientes a sonidos de altas frecuencias, es necesario aplicar un filtro para atenuar estas muestras que son consideradas como ruido y dejar pasar únicamente los datos correspondientes al rango auditivo, por lo que se implementa un filtro pasabajas. Dado que es un filtro digital, se hace uso de un Filtro de Respuesta Finita, donde los coeficientes de este se modifican para obtener el comportamiento de un filtro pasa bajas. El cálculo de estos coeficientes se realiza a través de herramientas de software, como TFilter. Los coeficientes deben extraerse en formato entero para la implementación. Propiamente para la implementación se utiliza un módulo de Python diseñado para Micropython desarrollado por Peter Hinch. El repositorio en el que se puede encontrar se encuentra en la siguiente enlace.Dentro de este repositorio, se pueden encontrar diversos filtros, sin embargo, todos hacen uso de *fir.py*, llamando a la función de este módulo usando distintos coeficientes que modifican el comportamiento del Filtro de Respuesta Finita.

5.3. Implementación

A continuación se muestra la propuesta de implementación del algoritmo previamente mencionado:

```
while True:
  audio_in.readinto(mic_samples_mv) #Lectura de microfono al buffer
  delay(100)
  for i in range(len(mic_samples)):
    str_chain += str(bin(mic_samples[i]).replace("0b","")) #Se separa cada byte en
    su cadena de bits, se quita el '0b' de cada cadena de bits

  for j in range(len(str_chain)): #Se recorre la cadena de bits
    if(j % 64 == 0):
      str_chain = str_chain[:j]+str_chain[j+8:] #Por cada 64 bits, se ignora todo un
     byte de datos
    else:
      if(len(str_chain[j:j+8]) == 8): #Asegura que el tama o sean 8 bits
        decimalValue = int("0b" + str_chain[j:j+7]) #Convierte el byte a un valor
    decimal

        if(k > 100): #Condicion no necesaria, se utiliza con fines de debuggeo para
    visualizar las muestras y no saturar memoria
          break
        else:
```

```
17          filteredValueBuff[j] = fir(data, coeffs, decimalValue) #Env o de los
    valores decimales al filtro pasabajas
18          j+=1
19          k+=1
20        else:
21          continue
22    delay(50)
23    dac.write_timed(filteredValueBuff, SAMPLE_RATE_IN_HZ, mode = DAC.NORMAL) #
      Escritura de datos filtrados al DAC
24    str_chain = "" #Se limpia la cadena de bits
```

En las líneas 2 a 5, se ejecuta la lectura de I2S llenando un buffer de bytes, el cual se procesa para su manejo como una cadena de bits.

A continuación, de las líneas 7 a 12, se realiza la decimación con factor de 64. Por cada 8 bytes (64 bits) de información entrante, se recorta 1 byte de la cadena de datos. Cada byte de la nueva cadena se transforma a número decimal correspondiente a la amplitud de la señal. Los datos decimales se envían al Filtro de Respuesta Finita en la línea 17.

Finalmente, el buffer con datos filtrados se envía al DAC estableciendo los parámetros adecuados en la línea 23 y se limpia la cadena para comenzar el ciclo nuevamente.

A. Reproducción de un archivo .WAV

Utilizando la tarjeta SD, es posible reproducir archivos de audio directamente sobre la Ophyra y escuchar el archivo a través del amplificador de audio. Para realizar esto, se debe cargar un archivo .WAV a la memoria externa y establecer ciertos parámetros en la configuración del DAC.

Usando el modo de 8 bits y una frecuencia de salida de 8kHz, es posible obtener una señal de audio a través del jack de la tarjeta Ophyra.

El archivo WAV consiste de dos partes: el encabezado y los datos. Al saltar a la posición 44 del archivo, se almacena la información en bytes usando un bytearray. Posteriormente, cada uno de estos bytes se mandan a través del DAC.

Cabe mencionar que el tamaño del bytearray creado para alojar los datos del archivo de audio deberá ser, generalmente, de gran tamaño. Para la reproducción de un archivo de 21 milisegundos, se registró que se requería de un bytearray de tamaño 40000. Por lo tanto, se deberá tomar esto en consideración al elegir el archivo a reproducir.

```
1  from machine import I2S
2  from machine import Pin
3  from pyb import DAC
4
5  dac = DAC(1, bits=8) # Inicializacion del DAC en modo de 8 bits
6
7  # ======= AUDIO CONFIGURATION =======
8  WAV_FILE = "file_name.wav" # Archivo .WAV a reproducir
9  WAV_SAMPLE_SIZE_IN_BITS = 8
10
11 SAMPLE_RATE_IN_HZ = 8000 # Tasa de muestreo a 8kHz
12
13 wav = open(WAV_FILE, "rb") # Apertura del archivo de audio
14 pos = wav.seek(44)
15
16 # Declaracion de buffers de datos
17 wav_samples = bytearray(40000)
18 wav_samples_mv = memoryview(wav_samples)
19
20 #Inicio de reproduccion del archivo
21 print("==========  START PLAYBACK ==========")
22 try:
23         num_read = wav.readinto(wav_samples_mv) # Se escriben los datos del .WAV al
       buffer de datos
24         if num_read == 0:
25             pos = wav.seek(44) # Se ignora el encabezado del .WAV
```

```
        else:
            dac.write_timed(wav_samples_mv[:num_read], SAMPLE_RATE_IN_HZ, mode=DAC.
    CIRCULAR) # Se escriben los datos del buffer al DAC

except (KeyboardInterrupt, Exception) as e:
    print("caught exception {} {}".format(type(e).name, e))

wav.close() # Se cierra el archivo de audio
print("Done")
```