

## PRÁCTICA 7: Buses de comunicación III (I2S)

Ya hemos visto el I2C y el SPI. Por último tenemos el I2S. Este protocolo nos sirve para reproducir archivos de audio digital con la ayuda de la placa, entre otros. El I2S utiliza 3 cables: **SCK o BCLK**, que es el reloj serial, el **Word Select (WS) o Frame Select (FS)**, que diferencia entre el canal izquierdo o el derecho, y por último el **Serial Data (SD)**, que es donde la carga útil se transmite en 2 complementos.

Primero, mostraré el código de ejercicio práctico 1, que es la **reproducción desde memoria interna**.

### Código 1

```
#include "AudioGeneratorAAC.h"
#include "AudioOutputI2S.h"
#include "AudioFileSourcePROGMEM.h"
#include "sampleaac.h"

AudioFileSourcePROGMEM *in;
AudioGeneratorAAC *aac;
AudioOutputI2S *out;

void setup(){
    Serial.begin(115200);

    in = new AudioFileSourcePROGMEM(sampleaac, sizeof(sampleaac));
    aac = new AudioGeneratorAAC();
    out = new AudioOutputI2S();
    out -> SetGain(0.125);
    out -> SetPinout(4,5,6);
    aac->begin(in, out);
}

void loop(){
    if (aac->isRunning()) {
        aac->loop();
    } else {
        aac -> stop();
        Serial.printf("Sound Generator\n");
        delay(1000);
    }
}
```

### Explicación del código

#### Inclusión de librerías

```
#include "AudioGeneratorAAC.h"
#include "AudioOutputI2S.h"
#include "AudioFileSourcePROGMEM.h"
#include "sampleaac.h"
```

Estas líneas incluyen las bibliotecas necesarias para generar y reproducir audio AAC, y para leer el archivo de audio desde la memoria del programa, donde **sampleaac.h** es un archivo de encabezado que contiene los datos del archivo de audio AAC codificados en un arreglo.

## Declaración de variables

```
AudioFileSourcePROGMEM *in;  
AudioGeneratorAAC *aac;  
AudioOutputI2S *out;
```

Se declaran punteros a objetos de las clases **AudioFileSourcePROGMEM**, **AudioGeneratorAAC** y **AudioOutputI2S**. Estos objetos se usarán para manejar la fuente de audio, la generación de audio AAC y la salida de audio a través del protocolo I2S, respectivamente.

## Función *setup()*

```
void setup(){  
    Serial.begin(115200);  
  
    in = new AudioFileSourcePROGMEM(sampleaac, sizeof(sampleaac));  
    aac = new AudioGeneratorAAC();  
    out = new AudioOutputI2S();  
    out->SetGain(0.125);  
    out->SetPinout(4, 5, 6);  
    aac->begin(in, out);  
}
```

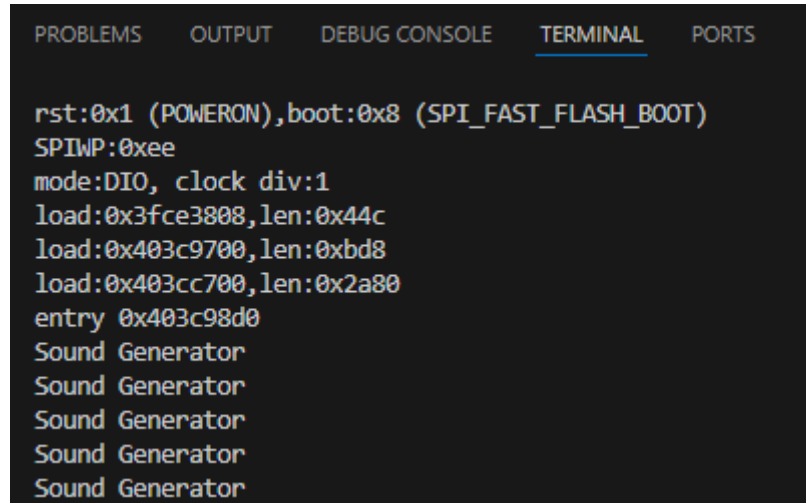
- + **Serial.begin(115200);** : Inicializa la comunicación serie con una velocidad de 115200 baudios.
- + **in = new AudioFileSourcePROGMEM(sampleaac, sizeof(sampleaac));** : Crea un nuevo objeto *AudioFileSourcePROGMEM* que lee los datos de audio desde el arreglo *sampleaac* almacenado en la memoria del programa.
- + **aac = new AudioGeneratorAAC();** : Crea un nuevo objeto *AudioGeneratorAAC* que se encarga de la decodificación y generación del audio AAC.
- + **out = new AudioOutputI2S();** : Crea un nuevo objeto *AudioOutputI2S* que envía el audio decodificado a través del protocolo I2S.
- + **out->SetGain(0.125);** : Ajusta la ganancia de salida del audio a 0.125.
- + **out->SetPinout(4, 5, 6);** : Configura los pines de salida I2S, donde 4 es el pin de datos (Data), 5 es el pin de reloj de bits (Bit Clock), y 6 es el pin de reloj de palabra (Word Clock).
- + **aac->begin(in, out);** : Inicia el proceso de decodificación de audio AAC, utilizando *in* como fuente de datos y *out* como salida de audio.

## Función *loop()*

```
void loop(){  
    if (aac->isRunning()) {  
        aac->loop();  
    } else {  
        aac->stop();  
        Serial.printf("Sound Generator\n");  
        delay(1000);  
    }  
}
```

- + ***if (aac->isRunning()) { aac->loop(); }*** : Verifica si el generador de audio AAC está en funcionamiento. Si está en funcionamiento, llama al método loop() del objeto aac para continuar la reproducción del audio.
- + ***else { aac->stop(); Serial.printf("Sound Generator\n"); delay(1000); }*** : Si el generador de audio no está en funcionamiento (es decir, ha terminado de reproducir el archivo), detiene el generador de audio con aac->stop(), imprime "Sound Generator" en el monitor serie, y espera 1 segundo antes de continuar el bucle.

## Imagen del terminal



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

rst:0x1 (POWERON),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fce3808,len:0x44c
load:0x403c9700,len:0xbd8
load:0x403cc700,len:0x2a80
entry 0x403c98d0
Sound Generator
Sound Generator
Sound Generator
Sound Generator
Sound Generator

```

Ahora, veremos el siguiente código el cual **reproduce un archivo WAVE desde una tarjeta SD externa**. En este caso, utilizaremos el MAX98357A, igual que antes, y un lector de tarjetas SD, que es desde donde obtendremos el archivo .wav

## Código 2

```

#include "Audio.h"
#include "SD.h"
#include "FS.h"

// Digital I/O used
#define SD_CS      39
#define SPI_MOSI   35
#define SPI_MISO   37
#define SPI_SCK    36
#define I2S_DOUT   6
#define I2S_BCLK   4
#define I2S_LRC    5

Audio audio;

void setup(){
  pinMode(SD_CS, OUTPUT);
  digitalWrite(SD_CS, HIGH);
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  Serial.begin(115200);
  SD.begin(SD_CS);

```

```

    audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
    audio.setVolume(10); // 0...21
    audio.connecttoFS(SD, "kick.wav");
}

void loop(){
    audio.loop();
}

// optional
void audio_info(const char *info){
    Serial.print("info      "); Serial.println(info);
}
void audio_id3data(const char *info){ //id3 metadata
    Serial.print("id3data   "); Serial.println(info);
}
void audio_eof_mp3(const char *info){ //end of file
    Serial.print("eof_mp3    "); Serial.println(info);
}
void audio_showstation(const char *info){
    Serial.print("station   "); Serial.println(info);
}
void audio_showstreaminfo(const char *info){
    Serial.print("streaminfo "); Serial.println(info);
}
void audio_showstreamtitle(const char *info){
    Serial.print("streamtitle "); Serial.println(info);
}
void audio_bitrate(const char *info){
    Serial.print("bitrate    "); Serial.println(info);
}
void audio_commercial(const char *info){ //duration in sec
    Serial.print("commercial "); Serial.println(info);
}
void audio_icyurl(const char *info){ //homepage
    Serial.print("icyurl     "); Serial.println(info);
}
void audio_lasthost(const char *info){ //stream URL played
    Serial.print("lasthost   "); Serial.println(info);
}
void audio_eof_speech(const char *info){
    Serial.print("eof_speech "); Serial.println(info);
}
}

```

## Explicación del código

### Inclusión de librerías

```

#include "Audio.h"
#include "SD.h"
#include "FS.h"

```

Estas líneas incluyen las bibliotecas necesarias para la reproducción de audio, el manejo del sistema de archivos y el acceso a la tarjeta

## Definición de pines

```
#define SD_CS      39
#define SPI_MOSI   35
#define SPI_MISO   37
#define SPI_SCK    36
#define I2S_DOUT   6
#define I2S_BCLK   4
#define I2S_LRC    5
```

Estas líneas definen los pines a los que he conectado cada dispositivo. CS, MOSI, MISO y SCK corresponden al lector de tarjetas SD. DOUT, BCLK y LRC, corresponden al MAX98357A.

## Creación del objeto 'Audio'

```
Audio audio;
```

Se crea un objeto *audio* de la clase **Audio**, que se utilizará para manejar la reproducción de audio.

## Función *setup()*

```
void setup(){
  pinMode(SD_CS, OUTPUT);
  digitalWrite(SD_CS, HIGH);
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  Serial.begin(115200);
  SD.begin(SD_CS);
  audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
  audio.setVolume(10); // 0...21
  audio.connecttoFS(SD, "kick.wav");
}
```

- + ***pinMode(SD\_CS, OUTPUT);*** : Configura el pin de selección (CS) de chip de la tarjeta SD como salida.
- + ***digitalWrite(SD\_CS, HIGH);*** : Pone el pin de selección de chip en alto (no seleccionado).
- + ***SPI.begin(SPI\_SCK, SPI\_MISO, SPI\_MOSI);*** : Inicializa el bus SPI con los pines definidos para SCK (reloj), MISO (entrada) y MOSI (salida).
- + ***Serial.begin(115200);*** : Inicializa la comunicación serie con una velocidad de 115200 baudios.
- + ***SD.begin(SD\_CS);*** : Inicializa la tarjeta SD utilizando el pin de selección de chip.
- + ***audio.setPinout(I2S\_BCLK, I2S\_LRC, I2S\_DOUT);*** : Configura los pines de salida I2S para el reloj de bits, reloj de palabra y datos.
- + ***audio.setVolume(10);*** : Establece el volumen de la salida de audio en 10 (en un rango de 0 a 21).
- + ***audio.connecttoFS(SD, "kick.wav");*** : Conecta el sistema de archivos SD y selecciona el archivo de audio *kick.wav* para reproducir.

## Función *loop()*

```
void loop(){
  audio.loop();
}
```

- + **audio.loop()**: Llama al método loop() del objeto audio para manejar la reproducción del audio. Este método debe ser llamado repetidamente para mantener la reproducción.

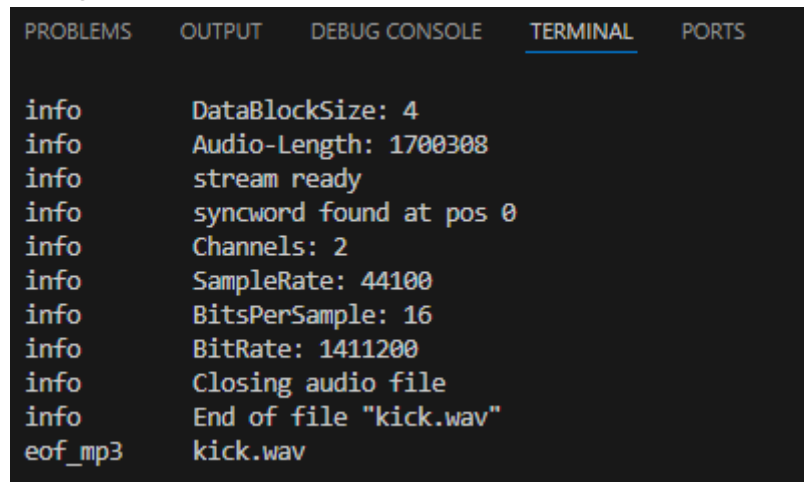
## Funciones opcionales

Estas funciones se utilizan para proporcionar información adicional sobre la reproducción de audio y pueden ser llamadas por la biblioteca Audio.

```
void audio_info(const char *info){
    Serial.print("info      "); Serial.println(info);
}
void audio_id3data(const char *info){ //id3 metadata
    Serial.print("id3data   "); Serial.println(info);
}
void audio_eof_mp3(const char *info){ //end of file
    Serial.print("eof_mp3    "); Serial.println(info);
}
void audio_showstation(const char *info){
    Serial.print("station    "); Serial.println(info);
}
void audio_showstreaminfo(const char *info){
    Serial.print("streaminfo "); Serial.println(info);
}
void audio_showstreamtitle(const char *info){
    Serial.print("streamtitle "); Serial.println(info);
}
void audio_bitrate(const char *info){
    Serial.print("bitrate    "); Serial.println(info);
}
void audio_commercial(const char *info){ //duration in sec
    Serial.print("commercial "); Serial.println(info);
}
void audio_icyurl(const char *info){ //homepage
    Serial.print("icyurl     "); Serial.println(info);
}
void audio_lasthost(const char *info){ //stream URL played
    Serial.print("lasthost   "); Serial.println(info);
}
void audio_eof_speech(const char *info){
    Serial.print("eof_speech "); Serial.println(info);
}
```

Se pueden poner en el código o no, el funcionamiento será el mismo.

## Imagen del terminal



A screenshot of a terminal window with a dark background and light-colored text. At the top, there is a horizontal bar with five tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the terminal displays a series of log messages. Each message consists of a prefix (either 'info' or 'eof\_mp3') followed by a space and then the message content. The messages describe the processing of an audio file named 'kick.wav', including details about data block size, audio length, stream status, syncword detection, channel count, sample rate, bits per sample, and bit rate. The final message indicates the end of the file.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

info      DataBlockSize: 4
info      Audio-Length: 1700308
info      stream ready
info      syncword found at pos 0
info      Channels: 2
info      SampleRate: 44100
info      BitsPerSample: 16
info      BitRate: 1411200
info      Closing audio file
info      End of file "kick.wav"
eof_mp3   kick.wav
```