SISTEMAS ELECTRÓNICOS DIGITALES

PROYECTO MICROS: Reproductor MP4 con letra por pantalla

Repositorio de Github: https://github.com/lunasbaciero/Grupo14_MICROS.git
Archivos de audio y texto para pruebas:
https://upm365-my.sharepoint.com/:f:/g/personal/jorge_sepulveda_alumnos_upm_es/EmYx2k3PnShPvrCNTtCo5AYBiTQ_wLLqjBfnZHZmGGMBzA?e=0qz7dG

GRUPO 14

Sepúlveda Cárcamo, Jorge - 56611
Sánchez-Urán Domínguez, Lucía - 56604
Silveira Baciero, Luna - 56614

ÍNDICE

1. Alcance del proyecto

1.1. Idea inicial

Inicialmente fueron planteadas diversas alternativas, para poder hacer un reproductor de música que además tuviera cierta similitud con un karaoke. Es por ello que se decidió emplear una pantalla LCD mediante la cual se muestre la letra de la canción a la vez que se coordina con la música que se escucha de la salida jack del micro.

Al investigar sobre el tipo de altavoz que podía ser empleado a la salida del micro, encontramos únicamente la opción de poder hacerlo con un altavoz de baja impedancia, y al poner bastante gasto sobre el presupuesto del hardware,se tomó la decisión de emplear la salida del jack para poner unos auriculares por los cuales se escuche la canción que se está reproduciendo.

Para poder acceder a los archivos de audio o texto, creamos las conexiones correspondientes para poder introducir el USB en el puerto del micro USB. La coordinación entre la letra mostrada y la reproducción de música de cada canción se gestionará mediante marcas de tiempo en el texto de las canciones.

Por otro lado, una de las ideas iniciales fue poder introducir el sonido por medio de un micrófono y después que saliera junto con el sonido de la canción. No obstante, al resultar bastante más complejo a la hora de incluso necesitar la implementación de un filtro a la salida del circuito para eliminar el ruido, se decidió no introducir audio adicional a la salida del archivo de audio de la canción.

1.2. Retos y complejidades

En un principio, tratamos de repartirnos el trabajo como en el trabajo de FPGAs pero en este caso, aunque las entidades a realizar fueran repartidas entre cada uno de los componentes, se dependía de forma más directa de cada una de las partes en las funciones a implementar. Además, para poder ajustar la misma configuración de proyecto, fue necesario esperar a juntar las partes para tanto poder comunicar cada una de las partes en función a cómo habían sido llamadas en cada una de las funciones como para poder probar todo el proyecto junto para ajustar la configuración del hardware empleado.

2. Primera parte del proyecto

2.1. Definición de funciones a implementar

2.1.1. Gestión de archivos por medio del USB

2.1.1.1. Montar USB

En esta función se detecta si hay algún USB enchufado en el puerto micro USB del microcontrolador, empleando un adaptador hembra-hembra de USB del cable micro-USB a la unidad que contendrá los archivos necesarios para poder reproducir las canciones y mostrar la letra por la pantalla LCD. En caso de no detectar ningún USB se emite mensaje de error por la pantalla. Se utilizó la función f_mount() de la librería FatFS para reconocer y montar el sistema de archivos del USB, así como lcd_i2c.h para el envio de mensajes a la pantalla LCD.

```
uint8_t MontarUSB(void){
	fd_USB = f_mount(&FatFs, "", 1);  // Monta el dispositivo USB

	switch(fd_USB){
	case FR_INVALID_DRIVE:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "FR_INVALID_DRIVE");
		break;
	case FR_DISK_ERR:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "FR_DISK_ERR");
		break;
	case FR_NOT_READY:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "FR_NOT_READY");
		break;
	case FR_NOT_ENABLED:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "FR_NOT_ENABLED");
		break;
	case FR_NO_FILESYSTEM:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "FR_NO_FILESYSTEM");
		break;
	default:
		lcd_gotoxy(&lcd1, 0, 0);
		lcd_puts(&lcd1, "default");
	}

	if(fd_USB != FR_OK){
```

```
                    //Eviar mensaje por LCD avisando de que no se encuentra un LCD

                    lcd_gotoxy(&lcd1, 0, 1);
                    lcd_puts(&lcd1, "ERROR AL CONECTAR USB");
                    return 1; //Se considera 1 como error al montar el USB (Puede ser interesante cambiarlo
por un enum)
          }
          return 0;
}
```

2.1.1.2. Abrir fichero

Se encarga de abrir el fichero de audio o de texto que se esté pasando, de este modo y de forma igual que con la función anterior, se devuelve 0 en caso de funcionar y sino, se devuelve un mensaje de error. Se utilizó la función f_open() de la librería FatFS para abrir el fichero, así como lcd_i2c.h para el envío de mensajes de error a la pantalla LCD.

```
uint8_t AbrirFichero(char* nombre_fichero, FIL* file){

          fd_USB = f_open(file, nombre_fichero, FA_READ);

          if(fd_USB == FR_OK){
                    return 0;
          }
          //Mensaje de error al abrir archivo

          lcd_gotoxy(&lcd1, 0, 1);
          lcd_puts(&lcd1, "ERROR AL ABRIR ARCHIVO");

          return 2; //Se considera 2 como error al abrir archivo (Puede ser interesante cambiarlo por un enum)
}
```

2.1.1.3. Leer Fichero Texto y Reproducir Fichero Sonido

Mediante esta función, se gestiona la lectura del archivo, en un caso de texto para el catálogo de canciones y la letra de cada una de las canciones, y por otro lado de audio para cada una de las canciones. En caso de producirse algún fallo, se mostrará un mensaje de error por la pantalla. Se utilizó la función f_read() de la librería FatFS para la lectura del fichero, así como lcd_i2c.h para el envío de mensajes de error a la pantalla LCD. Además, es importante mencionar que reproducir fichero de sonido envía la información al DAC usando DMA.

```
uint8_t LeerFicheroTexto(FIL* file, char* buffer){

          fd_USB = f_read(&file, buffer, sizeof(buffer) - 1, &bytesRead);
          if(fd_USB == FR_OK){
                    buffer[bytesRead] = '\0';
                    return 0;
          }
```

```
		lcd_gotoxy(&lcd1, 0, 1);
		lcd_puts(&lcd1, "ERROR AL LEER ARCHIVO");

		return 3; //Se considera 3 como error al leer el archivo (Puede ser interesante cambiarlo por un enum)

}
```

```
uint8_t ReproducirFicheroSonido(){
		static unsigned int bytesAudio = 0;
		static uint16_t buffer_audio[TAM_BUFFER_AUDIO];

		if(dma_transfer_complete_flag){
			fd_USB = f_read(&auidio, buffer_audio, TAM_BUFFER_AUDIO, &bytesAudio);

			if(fd_USB == FR_OK){
				//Si se ha acabado salimos de la función
				if(bytesAudio == 0){
					return 0;
				}
				//Si no, enviamos por dma
				HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, buffer_audio, TAM_BUFFER_AUDIO, DAC_ALIGN_12B_R);
			}

			//MENSAJE ERROR
			lcd_gotoxy(&lcd1, 0, 1);
			lcd_puts(&lcd1, "ERROR AL LEER ARCHIVO");

			return 3;
		}
		return 0;
}
```

2.1.1.4. Inicialización de pantalla LCD

Es necesario inicializar la estructura que emplea la librería lcd_i2c.h que usamos para controlar la pantalla lcd. Para ello tenemos que asignar a una variable de esta estructura el handler de i2c del canal al que está conectada la pantalla, así como la dirección a la que tiene que enviar la información.

```
void init_lcds(void){
		lcd1.hi2c = &hi2c1;
		lcd1.address = 0x4E;
		lcd_init(&lcd1);
}
```

2.1.2. Gestor de información de archivos de texto

2.1.2.1. Generar Catálogo

Esta función es la que envía cada uno de los títulos de las canciones disponibles a la memoria del programa, junto con los nombres del archivo de la letra y del archivo de audio que vayan relacionados con ese título. El catálogo propiamente dicho se guarda en una lista circular, de tal forma que podamos cambiar cómodamente de una canción a otra.

```
catalogo* GenerarCatalogo(char* buffer_catalogo){
	catalogo* cat, *cat_anterior;
	catalogo* primer_cat;

	char n[TAM_STRING];

	//Creamos la lista

	strncpy(n,strtok(buffer_catalogo, "\n"),TAM_STRING);
	while (n != NULL) {

		if(cat == NULL){
			cat = (catalogo*)malloc(sizeof(catalogo));
			strcpy(cat->nombre, "");
			strcpy(cat->nombre_mp4, "./audio/");
			strcpy(cat->nombre_txt, "./letra/");
			cat->siguiente = NULL;
			cat->anterior = NULL;

			primer_cat = cat;
		}
		else{
			cat_anterior = cat;

			cat->siguiente = malloc(sizeof(catalogo));
			cat = cat->siguiente;

			strcpy(cat->nombre, "");
			strcpy(cat->nombre_mp4, "./audio/");
			strcpy(cat->nombre_txt, "./letra/");
			cat->siguiente = NULL;
			cat->anterior = cat_anterior;
		}

		//Asignacion en la lista circular
		strncpy(cat->nombre,n,TAM_STRING);

		strncpy(cat->nombre_mp4,strcat(cat->nombre_mp4,n),TAM_STRING);
		strncpy(cat->nombre_mp4,strcat(cat->nombre_mp4,".mp4"),TAM_STRING);

		strncpy(cat->nombre_txt,strcat(cat->nombre_txt,n),TAM_STRING);
		strncpy(cat->nombre_txt,strcat(cat->nombre_txt,".txt"),TAM_STRING);

	   // Obtener la siguiente línea
	   strncpy(n,strtok(NULL, "\n"),TAM_STRING);
	}

	cat->siguiente = primer_cat;
	primer_cat->anterior = cat;
```

```
        return primer_cat;
}
```

2.1.2.2. Mostrar Catálogo

La función es la encargada de dar la salida por la pantalla LCD de la canción que aparece, se puede avanzar al siguiente título o retroceder al anterior por medio de los pulsadores. Además, también son los encargados de poder seleccionar la canción. Se utilizó la librería lcd_i2c.h para el envío de los nombres a la pantalla LCD.

```
catalogo* MostrarCatalogo(catalogo* cat){

        catalogo* primer = cat;

        if(FLAG_SIGUIENTE){
                cat = cat->siguiente;
                FLAG_SIGUIENTE = 0;
        }

        if(FLAG_ANTERIOR){
                cat = cat->anterior;
                FLAG_ANTERIOR = 0;
        }

        if(cat == NULL){
                cat = primer;
        }

        lcd_gotoxy(&lcd1, 0, 1);
        lcd_puts(&lcd1, cat->nombre);

        return cat;
}
```

2.1.2.3. Enviar Letra

Una vez seleccionada la canción, esta función se encarga de enviar la letra por la pantalla LCD, leyendo el archivo de texto con sus correspondientes marcas temporales sobre el tiempo que deben ser mostradas cada una de las líneas de la canción. Se utilizó la librería lcd_i2c.h para el envío de mensajes a la pantalla LCD.

```
void EnviarLetra(char* buffer_letra){

	static char linea[TAM_STRING] = "";
	char t_deseado_str[TAM_STRING];

	if(t_letra_actual == 0){
		if(linea == "" || nueva_cancion == 1){
			strncpy(linea,strtok(buffer_letra, "\n"),TAM_STRING);
			nueva_cancion = 0;
		}
		else {
			strncpy(linea,strtok(buffer_letra, NULL),TAM_STRING);
		}
		//SACAR EL TIEMPO Y CONVERTIRLO
			strncpy(t_deseado_str,linea,2);
			t_letra_deseado = (t_deseado_str[0]-'0')*10+(t_deseado_str[1]-'0');

		lcd_gotoxy(&lcd1, 0, 1);
		lcd_puts(&lcd1, linea);
	}
	else if(t_letra_actual >= t_letra_deseado){
		t_letra_actual = 0;
	}
}
```

En cuanto a los tiempos, se usa un temporizador de propósito general configurado para devolver una interrupción cada Hz. En cada una de esas interrupciones sumamos 1 al tiempo actual. Es importante por ello activar el temporizador al entrar al estado en el que se reproducirá la canción, y desactivarlo al salir.

```
void TIM6_IRQHandler(void) {
  HAL_TIM_IRQHandler(&htim6);  // Llamada al manejador de interrupción

  t_letra_actual += 1;

}
```

2.1.3. Gestor de pulsadores y potenciómetro

2.1.3.1. Rutina de atención a la interrupción

La pulsación de los botones genera una interrupción que debe ser atendida por la función HAL dedicada a ello: HAL_GPIO_EXTI_Callback. En esta función, tras comprobar que el botón pulsado no genera rebotes mecánicos utilizando HAL_GetTick(), se comprueba qué pin ha generado la interrupción. Como se observa en el código, el pin PA0 se asocia al botón 1, el pin PA1 al botón 2 y el pin PA2 al botón 3. Según el pin que haya generado la interrupción, se realiza una llamada a una función u otra, dado que cada uno de los botones tiene acciones distintas según el estado en el que se encuentre el sistema. Se ha realizado la llamada a funciones en lugar de escribir el código en la rutina de atención a la interrupción dado que no se recomienda estar más del tiempo necesario dentro de esta llamada.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
	static uint32_t t_button = 0;


	if(HAL_GetTick() - t_button > 10){
		if (GPIO_Pin == GPIO_PIN_0){		// Botón 1: Configurado en PA0
			Button1Pressed();
		}
		else if (GPIO_Pin == GPIO_PIN_1){		// Botón 2: Configurado en PA1
			Button2Pressed();
		}
		else if (GPIO_Pin == GPIO_PIN_2){		// Botón 3: Configurado en PA2
			Button3Pressed();
		}
		t_button = HAL_GetTick();
	}
}
```

2.1.3.2. Button pressed

Dado que existen 3 botones a pulsar, se ha creado una función por cada uno de los botones que podrá generar una interrupción, que son las tres funciones a las que se puede llamar desde la rutina de atención a la interrupción explicada anteriormente. Cada una de ellas analiza en qué estado se encuentra el sistema (selección de canción, canción en reproducción, canción en pausa) y realiza las acciones necesarias según el estado y el botón que haya generado la interrupción.

La razón por la que existen tres botones es porque en el estado de selección de canción, es decir, el estado en el que se muestra el catálogo, se debe poder subir y bajar en la lista, así como seleccionar la canción a reproducir.

```
void Button1Pressed(){
        switch(estado){
                case 1:  // Botón 1 en estado 1: Subir en la lista
                        FLAG_ANTERIOR = 1;
                        break;
                default:
                        break;
        }
}
```

En primer lugar, Button1Pressed solo tendrá acción en caso de que el sistema se encuentre en el modo de selección de canción. Su función será la de ascender (retroceder) en la lista de canciones que la componen. Para ello, la función “Mostrar Catálogo”, utiliza la función FLAG_ANTERIOR, por lo que si el sistema se encuentra en el catálogo de canciones y se pulsa el botón 1, se establecerá el valor de dicha bandera a 1 para que el catálogo muestre las canciones anteriores.

Como ya se ha mencionado, este botón no tiene función en ningún otro estado del sistema, por lo que el resto de interrupciones que provoque el botón 1 pasarán por el case “default” saliendo del switch y de la función sin realizar más funciones.

```
void Button2Pressed(){
        switch(estado){
                case 1: // Botón 2 en estado 1: Seleccionar canción
                        nueva_cancion = 1;
                        estado = 2;
                        break;
                case 2: // Botón 2 en estado 2: Pausar canción
                        estado = 3;
                        break;
                case 3: // Botón 2 en estado 3: Reanudar canción (resume)
                        estado = 2;
                        break;
                default:
                        break;
        }
}
```

Por otro lado, Button2Pressed tiene diversas funciones según el estado de la máquina de estados. Si se está mostrando la letra, su función es seleccionar una canción. Si se está reproduciendo una canción, se utilizará para parar. Por último, en caso de que una canción se encuentre parada, servirá para ser reanudada.

Para el caso en el que la máquina se encuentre en la selección de canción, dado que su función es seleccionar la canción, esto implica un cambio en el estado a canción en reproducción, lo que corresponde al estado 2. Por otra parte, la función Enviar Letra requiere que la variable “nueva_cancion” tenga valor 1 para el desarrollo de la misma, por lo que se realiza dicha asignación. Esta misma variable es también utilizada en la máquina de estados en el estado de reproducción de canción.

Para los otros dos casos se puede observar que el objetivo de los botones es el indicado anteriormente pero su implementación no está realizada al completo. Esto se debe a la imposibilidad de finalizar el proyecto a tiempo, por lo que no se ha podido terminar de coordinar el código de cada uno de los integrantes del grupo. Por ejemplo, en el caso de pulsar el botón 2 encontrándonos en una canción en reproducción, sería necesario crear la flag necesaria a otra función para detener la reproducción y cambiar el estado al 3, el estado de canción detenida. Por último, en caso de pulsarse el botón 3 el procedimiento sería parecido, pues el estado debería devolverse al estado 2 y se debería mandar la bandera correspondiente a otra función que determinara que se debe continuar con la reproducción de la canción.

```
void Button3Pressed(){
        switch(estado){
                case 1: // Botón 3 en estado 1: Bajar en la lista
                        FLAG_SIGUIENTE = 1;
                        break;
                case 2: // Botón 3 en estado 2: Regreso a selección de canción
                        estado = 1;
                        f_close(&texto);
                        f_close(&auidio);
                        break;
                default:
                        break;
        }
}
```

Por último, la función Button3Pressed se encarga de avanzar hacia abajo en el catálogo de canciones o de volver al catálogo de canciones al pulsarlo mientras se reproduce la canción. En el segundo caso se emplearía como un stop.

Por ello, si la máquina de estados está en mostrar la lista de canciones, se procede de forma similar al mismo caso para el botón 1: se cambia el valor de la flag que necesita “Mostrar Catálogo”, en este caso FLAG_SIGUIENTE. Si, por otra parte, se encuentra en el estado de canción en reproducción, provoca la salida a selección de canción, por lo que se debe cambiar el estado al de mostrar el catálogo de canciones, es decir, el estado 1 y, por seguridad, cerrar los dos archivos de texto y audio que se encontraban abiertos y en reproducción. Dado que se regresa a la selección de canciones, no es necesario almacenar el punto en el que se encuentra la canción en el momento de pulsar el botón, por lo que no

se deben añadir más funcionalidades. Por último, estando en el estado de canción detenida no se pueden realizar acciones con el botón 3, por lo que igual que en el caso de pulsar el botón 1 en este mismo estado, no se realiza nada con la interrupción.

2.1.3.3. Leer Potenciómetro

```
void LeerPotenciometro(void){
                        HAL_ADC_Start(&hadc1);
                        HAL_ADC_PollForConversion(&hadc1, 1000); // Espera
                        valpot = HAL_ADC_GetValue(&hadc1);
                        setvolumen();
 }
```

Mediante esta función, se lee el valor del potenciómetro y así regular el volumen con la función siguiente. Para ello, se inicializa el manejador del ADC y, a continuación, se lee su valor. Dicho valor será un número entre 0 y 2045 que posteriormente será manejado para modificar la onda del PWM que permitirá modificar el volumen de la pista de audio. Por ello, al finalizar la función se llama a la función encargada de modificar el volumen explicada a continuación.

2.1.3.4. Set volumen

```
void setvolumen(void){
                        float volumen = (float)valpot/4095.0;

                        salida_volumen = (uint32_t)(volumen * 4095);
                        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, salida_volumen);
                        HAL_Delay(10);
}
```

En esta función se ajusta el volumen para poder modificarlo a la salida del audio. Como otras funcionalidades indicadas anteriormente, no se encuentra completamente terminada dado que se desconoce cuál debe ser la conversión para modificar correctamente la salida. Para ello, hubiera sido necesario que funcionara en primer lugar la reproducción de audio y ajustar los valores entonces.

2.1.4. Máquina de estados del programa

En la máquina de estados del programa se va produciendo la secuencia de los estados de la siguiente forma:

- Main menu: no detecta aún USB y sale un mensaje de error por la pantalla LCD. Además, se implementó el indicador de forma que al encontrarse en este estado, se encienda el LED verde correspondiente al PIN PD12.

```
switch(estado){
        case 0: //Main menu
           printf("Main menu\n");
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

                        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);

                         if(MontarUSB() == 0){
                                 estado = 1;
                                 if(AbrirFichero("C:/SED/catalogo.txt", &texto)==0){
                                          error = LeerFicheroTexto(&texto, buffer_texto);
                                          f_close(&texto);

                                          cat = GenerarCatalogo(buffer_texto);
                                 }

                         }
        break;
```

- Menu canciones USB: en este estado, el programa se encarga de transmitir la información del catálogo de canciones por la pantalla LCD, previamente indicando que se encuentra en el estado 1. En este caso se activaría el LED conectado al pin PD13, que sería el naranja.

```
case 1: //Menu canciones USB - Como stop de cancion actual
                        //printf("Songs list\n");
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

                        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);

                        cat = MostrarCatalogo(cat);
                        lcd_puts(&lcd1, "ESTOY EN EL ESTADO 1");
                        LeerPotenciometro();


           //Introducir código LCD y micro
        break;
```

- Play canción seleccionada: tras haberse seleccionado una canción con el pulsador, se pasa al estado de play de la propia canción (estado 2), la cual activa la variable de nueva_cancion. En caso de estar en este estado, se comienza a reproducir el audio y la letra, podemos acceder a estos archivos gracias al vínculo creado con la función abrir fichero. En caso de no poder leerse los archivos, se mostraría un mensaje por pantalla, y en caso de encontrarnos en estado de pausa, se mostraría el paso al estado 3. Al igual que en los otros estados en este estado también se encenderá un LED distinto el cual sería el rojo, conectado al pin PD14.

```
case 2: //Play cancion seleccionada
          //printf("Play song\n");
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);

                    if(nueva_cancion){
                            if(AbrirFichero(cat -> nombre_mp4, &auidio)==0 || AbrirFichero(cat -> nombre_txt,
&texto)==0){
                                    // Apertura correcta de fichero de audio y fichero de texto
                                    if(LeerFicheroTexto(&texto, buffer_texto) == 0){

                                    }
                                            // Lee correctamente el fichero de texto

                                    }
                            else
                                    estado = 3; // Estado de pausa para mostrar el mensaje de error
                            break;
                            }
                    if (ReproducirFicheroSonido() == 0){
                            EnviarLetra(buffer_texto);
                    }
        break;
```

- Pause canción seleccionada: En caso de volver a pulsar el botón para cambiar de estado se pasaría al estado en pausa de la canción ya previamente seleccionada y la cual se estaría reproduciendo en el estado dos. En este caso el LED activado sería el LED azul conectado al pin PD15.

```
case 3: //Pause cancion seleccionada
          //printf("Pause song\n");
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
          //Introducir código LDC y micro
        break;

        default:
        break;
    }
 }
```

2.2. Funcionamiento del programa

Para poder utilizar el reproductor empleado, la forma de gestión sería secuencial de la siguiente manera:

Primero, la pantalla mostraría un mensaje de error al conectar el USB, es decir que estaría en el estado 0 y sería necesario introducir un USB con un archivo de texto que contuviese el listado de canciones y dos carpetas, una para los archivos de audio, y otra para los archivos de texto, teniendo en cuenta que ambos deben llamarse igual que el título de la canción para poder abrir ambos ficheros al seleccionar el título.

A la hora de elegir una de las canciones contenidas en el USB, podemos movernos por dicho listado pulsando los botones uno o tres para poder retroceder hacia la anterior o avanzar hacia la posterior. En caso de querer seleccionar la canción sobre la que estamos situados (aparece el la LCD), podríamos seleccionarla con el botón 2, el cual es el encargado de entrar en el modo de reproducción de la “nueva canción”.

Podemos, una vez dentro de la canción, entrar en modo de pausa, es decir, guardar la posición en cuanto al lector del fichero de texto y del reproductor de audio y posteriormente, reanudar la reproducción. Además, en caso de querer abandonar la canción y volver al menú de la lista de canciones disponibles, podríamos hacerlo pulsando el botón 3. En este caso no se guardaría la posición del fichero de texto o del reproductor de audio puesto que se ejecutaría como un STOP, no como un PAUSE.

En todos y cada uno de los pasos del funcionamiento correcto del programa, en caso de que no funcionase correctamente algunas de las funciones, se muestra el mensaje de error. También, se indica en cada uno de los estados el estado en el que nos encontramos para evitar confusiones.

Finalmente, la idea del programa a implementar en el microprocesador ha sido un Reproductor MP4 con posibilidad de mostrar la letra por pantalla.

2.3. Conclusión final del primer modelo

2.3.1. Resultado final

Durante el desarrollo del proyecto encontramos muchos problemas con la conexión USB. Al intentar establecer la conexión y montar el sistema de archivos, la función responsable de la librería FatFS devolvía FR_DISK_ERR. Este código está asociado a errores irrecuperables en las funciones para montar el sistema de archivos o leer ficheros y se corresponde habitualmente con un error al establecer la conexión con el dispositivo de almacenamiento. Sabiendo esto, intentamos encontrar el error y solucionarlo. A continuación enumeramos las posibles causas que encontramos, así como las medias que se tomaron para intentar solucionarlo:

- **Mala configuración de USB_OTG_FS o USB_HOST**: Se revisó la configuración de la placa y se comparó con proyectos similares que encontramos en foros y páginas web. Se realizó también una verificación con la herramienta de IA ChatGPT. Se encontró un error en configuración de la alimentación del puerto USB. Se solucionó habilitando el Vbus del USB_OTG_FS y el pin PC0 como GPIO_Output. Tras revisar la documentación de la placa se observó que el pin se conecta a un enable activo a baja, por lo que se fuerza al pin a estar siempre a baja. Aún así, el error FR_DISK_ERR persistió.
- **Posibilidad de confusión entre el puerto CN1 (mini USB para alimentación, programación y debugging) y el CN5 (micro USB):** Tras revisar la documentación de la propia placa, se comprobó que el USB_OTG_FS funcionando en modo Host solo se conectará al puerto CN5. Se descarta como posible error.
- **USB corrupto o mal formateado:** Se descartó inmediatamente que el USB estuviese corrupto, al poder leerse en nuestros ordenadores. Se revisó el formato del USB y se comprobó que era FAT32. Tras consultar la documentación de FatFS se comprueba también que el formato con el que trabaja la librería es FAT32. Se descarta como posible error.
- **Ausencia de un reloj externo:** El USB_OTG_FS necesita de un error externo para funcionar. Tras consultar la documentación de la placa, y algunas búsquedas en internet, parece que la STM32F407-D no lleva incluido un reloj externo de 8MHz. Por desgracia, no pudimos adquirir uno a tiempo. Esta parecía la razón más probable del error, sin embargo tras adquirir un reloj de cuarzo de 8MHz no se soluciono el problema. Además, en placas STM32F411 no parece ser necesario, aunque cabe la posibilidad de que si sea necesario en las 407.

Tras todo esto, seguimos sin poder solucionar el error, por lo que el proyecto final no es capaz de leer la memoria USB, pero cuenta con un sistema más detallado que el implementado inicialmente para la detección y notificación de errores.

A causa de no haber podido conectar correctamente el USB, y con ello poder leer el contenido, nos ha sido imposible finalizar el proyecto. Algunas de las implementaciones que

no hemos podido finalizar o comprobar a causa de ello es el funcionamiento de algunas funciones como Mostrar Catálogo, Enviar Letra o Set Volumen, puesto que no hemos podido obtener los datos del USB que nos permitieran mostrar la letra o reproducir el audio.

2.3.2. Objetivos de mejora

Es importante mencionar que con los problemas que surgieron en la conexión USB perdimos tiempo que se habría podido dedicar a acabar o mejorar código existente. Además dificultó mucho (o directamente hizo que no fuese posible) probar y retocar algunas de las funcionalidades ya implementadas. En este apartado se detallarán algunos de esos cambios y mejoras a realizar tras resolver los problemas con la conexión:

En primer lugar, un cambio importante a realizar sería establecer condiciones de regreso al estado 0, correspondiente a no detectar un USB, en caso de que se desconecte. Para esto, se ha consultado la documentación de la librería FatFS y de la STM32F407-D. Tras la consulta se observa que al carecer de una señal asíncrona que nos indique una desconexión, debemos volver a "montar" periódicamente el USB, ya que esta función revisa también si el USB es accesible. Idealmente, se incluiría f_mount() antes de todas las funciones que utilicen el USB. Tras esto, si se ha detectado correctamente se permitiría continuar al programa, y en caso contrario, se le forzará a volver a 0.

Por otra parte, aunque algo menos crítico, se debería haber definido un enumerado para el tratamiento de los errores. En vez de devolver siempre un entero, se podría devolver de esta forma mensajes algo más concretos como OK, ERR_APERTURA y ERR_LECTURA, por ejemplo.

Otra posible mejora a realizar es aprovechar mejor el espacio de la pantalla LCD. La pantalla cuenta con dos líneas de 16 caracteres cada una. Por ello, para poder utilizar todo el espacio con comodidad se puede implementar en el código la siguiente función, encargada de separar el mensaje en dos mensajes de 16 caracteres y enviarlos en orden a cada línea.

```
void lcd_send_two_lines(const char *text) {
    char line1[17] = {0}; // Línea 1 (16 caracteres + caracter nulo)
    char line2[17] = {0}; // Línea 2 (16 caracteres + caracter nulo)

    // Copiar los primeros 16 caracteres a la primera línea
    strncpy(line1, text, 16);

    // Si hay más de 16 caracteres, copiar los siguientes 16 a la segunda línea
    if (strlen(text) > 16) {
        strncpy(line2, text + 16, 16);
    }

    // Mostrar las líneas en el LCD
    lcd_put_cur(0, 0);
    lcd_send_string(line1);

    lcd_put_cur(1, 0);
    lcd_send_string(line2);
}
```

3. Segunda versión

Al no poder implementar de la forma en que queríamos el programa desarrollado, decidimos realizar una segunda parte del proyecto. Esta segunda parte pudimos realizarla gracias a los archivos que nos facilitó nuestro tutor del proyecto, dado que tras ya haber realizado el proyecto e investigaciones sobre cuáles podrían ser los posibles motivos del funcionamiento erróneo del programa, en concreto, de la conexión del USB, no habríamos logrado solventarlo,por lo que nuestro proyecto no era capaz de funcionar de la forma adecuada. Al comenzar a trabajar con los archivos nuevos, tuvimos que emplear una placa nueva dado que la configuración que teníamos disponible estaba hecha para una STM32F411 y tras hacer de nuevo incluso la configuración para nuestra placa, una STM32F407, no habríamos dado no la forma correcta de hacerlo.

Tras implementar el programa con la configuración para la placa STM32F411, decidimos realizar el proyecto en base a ese código. Para empezar, tratamos de implementar las funciones que ya habíamos creado previamente en el primer modelo del proyecto.

En primer lugar, primero nos centramos en poder establecer la sucesión y la transición entre un estado y otro mediante la máquina de estados que ya habíamos realizado. Podría decirse que esta fue la parte menos complicada, puesto que fuimos adaptando los estados que ya establecimos con las inicializaciones correspondientes al nuevo funcionamiento de cada una de las etapas del proyecto.

Posteriormente, decidimos implementar la variación del volumen con el potenciómetro y después, las funciones como leer los archivos de texto con la letra y poder mostrarlas por pantalla. También decidimos hacer los estados de STOP, START y PAUSE de forma que se coordinasen con los que ya teníamos del código del audio. Puede que esta fuera la parte más difícil, ya que al tratar de establecer los estados, tuvimos que modificar los que nosotros ya teníamos para poder coordinarlo bien, y sobretodo, el entender el código que ya teníamos para poder modificarlo sin dejar ninguna de las funcionalidades del programa con algún error de funcionamiento.

3.1. Definición de funciones a implementar

3.1.1. Gestión de archivos por medio del USB

3.1.1.1. Montar USB

Mantendremos la función que utiliza la librería y código que usamos como base y la modificaremos para poder observar el estado el estado de la conexión.

```
void Mount_USB (void)
{
	fresult = f_mount(&USBHFatFS, USBHPath, 1);
}
```

3.1.1.2. Inicialización de pantalla LCD

Una vez más, se mantiene el mismo código de inicialización. Únicamente se cambia el canal I2C que usamos para comunicarnos con el LCD

```
void init_lcds(void){
	lcd1.hi2c = &hi2c2;
	lcd1.address = 0x4E;
	lcd_init(&lcd1);
}
```

3.1.1.3. Buscar ficheros en el directorio

El código que empleamos de base contiene una función que emplea para buscar dentro de la estructura de archivos creada al montar el USB todos los archivos .WAV. Nosotros modificaremos la función para que busque también nuestros .txt con las letras. Utilizamos además las mismas estructuras que utiliza la librería para almacenar los WAV.

```
FRESULT AUDIO_Parse(void)
{
  FRESULT res = FR_OK;
  FILINFO fno;
  DIR dir;
  char *fn;


  res = f_opendir(&dir, USBHPath);
  FileList.ptr = 0;

  if(res == FR_OK)
  {
    while(Appli_state == APPLICATION_READY)
    {
      res = f_readdir(&dir, &fno);
      if(res != FR_OK || fno.fname[0] == 0)
      {
        break;
      }
      if(fno.fname[0] == '.')
      {
        continue;
      }

      fn = fno.fname;

      if(FileList.ptr < FILEMGR_LIST_DEPDTH)
      {
        if((fno.fattrib & AM_DIR) == 0)
        {
          if((strstr(fn, "wav")) || (strstr(fn, "WAV")))
          {
            strncpy((char *)FileList.file[FileList.ptr].name, (char *)fn, FILEMGR_FILE_NAME_SIZE);
            FileList.file[FileList.ptr].type = FILETYPE_FILE;
            FileList.ptr++;
          }
          if((strstr(fn, "txt")) || (strstr(fn, "TXT")))
                  {
                    strncpy((char *)TxtList.file[TxtList.ptr].name, (char *)fn, FILEMGR_FILE_NAME_SIZE);
                    TxtList.file[TxtList.ptr].type = FILETYPE_FILE;
                    TxtList.ptr++;
                  }
        }
      }
    }
  }
  NumObs = FileList.ptr;
  f_closedir(&dir);
  return res;
}
```

3.1.1.4. Lectura de Archivo de Texto

Tomamos como base la función de la versión anterior. Además, modificamos la forma de llenar el buffer. Para mayor comodidad en el procesado de la información posterior, nos aseguraremos de que al llenarlo no dejemos ninguna línea de las canciones a medias, y de devolver al comienzo de la siguiente línea el puntero de fichero.

```
void LeerFicheroTexto(char* buffer){
	UINT bytesRead;
	uint8_t i = 0;
	uint8_t buffer_correcto = 0;

	strcpy(buffer,"");
	fresult = f_read(&filetxt, buffer, 4096, &bytesRead);
	if(fresult == FR_OK){
		//Paramos al final de la última linea completa
		buffer_correcto = 0;
		for(i = bytesRead-1; i>0 && buffer_correcto == 0; i--){
			if(buffer[i]=='\n'){
				buffer[i+1] = '\0';
				fresult = f_lseek(&filetxt,f_tell(&filetxt) + i - (bytesRead));
				buffer_correcto = 1;
			}
		}
	}
}
```

3.1.2. Gestor de información de archivos de texto

3.1.2.1. Enviar Letra

La función para el envío de la letra es muy similar a la de la versión anterior. Seguimos comprobando cuando nos encontramos en la primera línea de una canción, pero añadimos una comprobación de si el buffer está vacío para volver a llenarlo. Además ya no enviamos directamente la información al lcd, sino que lo pasamos por una función que lo separe en líneas de 16 caracteres.

```
void EnviarLetra(char* buffer_letra){
	static char linea[TAM_STRING] = "";
	char t_deseado_str[TAM_STRING];

	if(cambio_linea){
		if(primera_linea == 1){
			strncpy(linea,strtok(buffer_letra, "\n"),TAM_STRING);
			primera_linea = 0;
		}
		else {
			strncpy(linea,strtok(NULL, "\n"),TAM_STRING);
		}

		if(linea[0] == '\0'){
			strcpy(buffer_letra,"");
			LeerFicheroTexto(buffer_letra);
			strncpy(linea,strtok(buffer_letra, "\n"),TAM_STRING);
		}

		if(!strcmp("&",linea)){
			AudioState = AUDIO_STATE_STOP;
			uwVolume = 10;
			AUDIO_OUT_SetVolume(uwVolume);
			return;
		}

		//SACAR EL TIEMPO Y CONVERTIRLO
		strncpy(t_deseado_str,linea,2);
		t_letra_deseado = (t_deseado_str[0]-'0')*10+(t_deseado_str[1]-'0');

		strcpy(linea,linea+2);

		lcd_send_two_lines(linea);
		cambio_linea = 0;
	}
	else if(t_linea >= t_letra_deseado){
		t_linea = 0;
		cambio_linea = 1;
		lcd_gotoxy(&lcd1,0,0);
		lcd_clear(&lcd1);
	}
}
```

3.1.2.2. Dividir Letra en dos líneas para enviar a la pantalla LCD

No solo era necesario enviar la letra a la pantalla LCD sino también hacerlo con formato y tomando en cuenta algunas consideraciones. En primer lugar, la longitud máxima que puede tener una cadena a mostrar en la LCD es de 32 caracteres, 16 por cada línea. Para cada una de las líneas se crea un array.

Al mostrar las letras de las canciones, se leen los nombres completos de los archivos, lo que incluye la extensión ".txt". Para que no se muestre, es necesario sustituir todos los caracteres desde el punto por caracteres nulos, los cuales no se muestran por pantalla. También es necesario sustituir el carácter final de cada línea ('\r') por un el carácter nulo, pues el fin de línea se representa en la LCD con su caracter ASCII correspondiente; mientras que el carácter nulo no muestra nada.

No es condición obligatoria que la línea enviada a la LCD sea mayor de 16 caracteres, por lo que es posible que el final de línea se busque tanto en la primera línea como en la segunda. Es algo no necesario a tener en cuenta en el caso del nombre de las canciones porque no se contempla que la suma del identificador de canción, nombre de artista y nombre de canción ocupe menos de 16 caracteres.

```
void lcd_send_two_lines(const char *text) {
    char line1[17] = {0}; // Línea 1 (16 caracteres + terminador nulo)
    char line2[17] = {0}; // Línea 2 (16 caracteres + terminador nulo)
    int i=0;
    // Copiar los primeros 16 caracteres a la primera línea
    strncpy(line1, text, 16);
    // Si hay más de 16 caracteres, copiar los siguientes 16 a la segunda línea
    if (strlen(text) > 16) {
        strncpy(line2, text + 16, 16);
    }
    while (line2[i] != '\0') {  // Recorrer la cadena
            if (line2[i] == '.') {  // Si encontramos un punto
                line2[i] = '\0';  // Reemplazamos el punto por un carácter nulo
                break;  // Detenemos la búsqueda
            }
            i++;
        }
    while (line1[i] != '\0') {
            if (line1[i] == '\r') {
                line1[i] = ' ';
                break;
            }
            i++;
        }
    while (line2[i] != '\0') {
                if (line2[i] == '\r') {
                    line2[i] = ' ';
                    break;
                }
                i++;
    }
   // Mostrar las líneas en el LCD
    lcd_gotoxy(&lcd1, 0, 0);
    lcd_puts(&lcd1, line1);
    lcd_gotoxy(&lcd1, 0, 1);
    lcd_puts(&lcd1, line2);
}
```

3.1.3. Gestor de pulsadores y potenciómetro

3.1.3.1. Interrupción de los botones

La activación de los botones decidimos hacerla por interrupciones, por un Callback en el cual se activaba una flag u otra dependiendo del botón que estuviera siendo pulsado en ese momento. Como podemos ver, en el caso del estado 2, no solo se activa la flag del botón sino que si queremos pausar o continuar con la reproducción, pausa o continua la cuenta del temporizador encargado de mostrar por pantalla cada una de las líneas el tiempo especificado.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	static uint32_t t_button = 0;
		if(HAL_GetTick() - t_button > 20){

			if (GPIO_Pin == GPIO_PIN_0)
			{
				boton0 = 1;
				//AudioState = AUDIO_STATE_NEXT;
				if(estado == 2){
					if (AudioState == AUDIO_STATE_PLAY)
					{
						AudioState = AUDIO_STATE_PAUSE;
						HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_1);
						boton0=0;
					}

					if (AudioState == AUDIO_STATE_WAIT)
					{
						AudioState = AUDIO_STATE_RESUME;
						HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_1);
						boton0=0;
					}
				}
			}

			else if(GPIO_Pin == GPIO_PIN_1)
			{
				boton1 = 1;


			}

			else if (GPIO_Pin == GPIO_PIN_2)
			{
				boton2 = 1;
			}

		t_button = HAL_GetTick();
		}
}
```

3.1.3.2. Leer Potenciómetro

Una vez más, usamos una función para leer el potenciómetro y convertirlo en un valor de volumen. Además, nuestra librería contiene una función para asignar el volumen al I2C, así como una función para cambiar su nivel. Incluimos la función para cambiar el volumen dentro de la lectura del potenciómetro en lugar de hacer una función adicional. Además, revisamos la función que usa la librería para asignar el volumen y observamos que hay que incluir protecciones para evitar que el nivel de volumen caiga por debajo de 10 o suba por encima de 90.

```
void LeerPotenciometro(void){
        // No es necesario comprobar quién ha generado la interrupción
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 1000);
        valpot = HAL_ADC_GetValue(&hadc1);
        uwVolume = valpot*100/4095;
        if(uwVolume < 10)
                uwVolume = 10;
        else if(uwVolume > 90)
                uwVolume = 90;
        AUDIO_OUT_SetVolume(uwVolume);
}
```

3.1.4. Máquina de estados del programa

Dentro del while(1) del programa principal se encuentra el código mostrado a continuación. Se contemplan 3 estados:

- El sistema se encuentra en el estado 0 si el USB no se encuentra conectado a la placa o bien si no se está dando la comunicación correctamente. Una vez se conecta correctamente, se transiciona al estado 1, se sitúa el “puntero” de la LCD en la primera fila y primera columna de la pantalla para empezar a dibujar en la posición correcta y se borra todo lo que se estaba dibujando en ella previamente: “CONECTE USB”. Además, se monta el USB y se llama a la función AUDIO_StorageParse(), perteneciente a la biblioteca externa utilizada y previamente utilizada.
- El sistema se encuentra en el estado 1 si se muestran los títulos de las canciones leídas del USB. Dentro de este estado se utilizan los tres botones externos.
    - El botón central se utiliza para seleccionar la canción que se esté mostrando en pantalla. De esta forma se pasa el estado siguiente, se borra todo lo que se estuviera mostrando previamente, se cambia el estado del audio a “RESUME” y se activa la bandera de primera línea utilizada en la función EnviarLetra.
    - Los botones izquierdo y derecho se utilizan para desplazarse por la lista de canciones. La lista está configurada como una lista circular.
- El sistema se encuentra en el estado 2 cuando se está reproduciendo la canción seleccionada y se está mostrando la letra de la misma en la pantalla LCD. En este estado, si el USB está montado correctamente, se comienza a reproducir la canción seleccionada mediante la función de la biblioteca externa “AUDIO_PLAYER_Start” e identificada por su identificador. Además, se inicializa el temporizador configurado para realizar una interrupción cada segundo. A continuación, se abre el fichero de la

letra y se lee. Mientras la canción no haya finalizado, se reproduce la canción, se envía la letra a la pantalla LCD y se lee el potenciómetro con el que se modifica el volumen de la canción. Si se pulsa el botón central, se pausa la canción y una nueva pulsación la reanudará. En el caso de que se pause, es necesario silenciar la canción pues se queda un residuo auditivo por la configuración de la biblioteca. Por otra parte, si el audio entra en modo “STOP”, se declara a 1 la variable que define que la canción ha finalizado, se regresa al estado de mostrar los títulos de las canciones, se detiene el temporizador, se devuelve el puntero al inicio del fichero de texto a leer. Además, se devuelven al valor de inicio las variables de las funciones para poder reproducir una nueva canción.

```
switch(estado)
  {
  case 0:
        lcd_gotoxy(&lcd1,0,0);
        lcd_puts(&lcd1, "CONECTE USB");
        if (Appli_state == APPLICATION_READY)
        {
              estado = 1;
              lcd_gotoxy(&lcd1,0,0);
              lcd_clear(&lcd1);
              Mount_USB();
              AUDIO_StorageParse();
        }
        break;

  case 1:
        lcd_send_two_lines((char *)TxtList.file[idx].name);
        if (IsFinished)
              IsFinished = 0;

        //estado = 2;
        if (boton0)
        {
              estado = 2;
              boton0 = 0;
              lcd_gotoxy(&lcd1,0,0);
              lcd_clear(&lcd1);
              AudioState = AUDIO_STATE_RESUME;
              primera_linea = 1;
        }
        if (boton1)
        {
              if(idx>0)
              {
                    idx--;
              }
              else
              {
                    idx = NumObs-1;
              }
              boton1 = 0;
              lcd_gotoxy(&lcd1,0,0);
              lcd_clear(&lcd1);
        }

        if (boton2)
        {
              if(idx<NumObs-1)
              {
                    idx++;
```

```
                }
                else
                {
                   idx = 0;
                }
                boton2 = 0;
                lcd_gotoxy(&lcd1,0,0);
                lcd_clear(&lcd1);
        }

        break;

case 2:

        if(fresult == FR_OK) // USB montado correctamente
        {
                Mount_USB();
                AUDIO_PLAYER_Start(idx);
                HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_1);
                f_open(&filetxt, (char*)TxtList.file[idx].name, FA_READ);
                LeerFicheroTexto(buffertxt);

                if (IsFinished)
                        IsFinished = 0;

                while (!IsFinished)
                {
                        AUDIO_PLAYER_Process(TRUE);
                        EnviarLetra(buffertxt);
                   LeerPotenciometro();
                   if(boton1)
                   {
                        boton1 = 0;
                        AudioState = AUDIO_STATE_STOP;
                        uwVolume = 10;
                        AUDIO_OUT_SetVolume(uwVolume);
                        // Silenciamos el audio porque queda un residuo de la canción al pararla
                   }
                   if (AudioState == AUDIO_STATE_STOP)
                   {
                        IsFinished = 1;
                        estado = 1;
                        HAL_TIM_OC_Stop_IT(&htim3, TIM_CHANNEL_1);
                        f_lseek(&filetxt, 0);
                        f_close(&filetxt);
                        t_letra_deseado=0;
                        t_linea=0;
                        cambio_linea = 1;
                        AudioState = AUDIO_STATE_IDLE;
                   }
                }
        }
        else
                estado = 0;
        break;

case 3:
        break;
}
```

3.2. Funcionamiento del programa

El funcionamiento del programa es muy similar al funcionamiento teórico del primero. Para poder utilizar el reproductor empleado, la forma de gestión sería secuencial de la siguiente manera:

Primero, la pantalla mostraría un mensaje de error al conectar el USB Sería necesario introducir un USB con archivos de texto y audio. Esta vez sin necesidad de ordenarlos en carpetas. No es necesario que tengan exactamente el mismo nombre, pero si es muy recomendable que se encuentren numerados igual para que se ordenen igual en sus respectivos vectores. En caso contrario, podrían no llevar el mismo orden y podría no reproducirse la letra correcta.

A la hora de elegir una de las canciones contenidas en el USB, podemos movernos por dicho listado pulsando los botones uno o tres para poder retroceder hacia la anterior o avanzar hacia la posterior. En caso de querer seleccionar la canción sobre la que estamos situados (aparece el la LCD), podríamos seleccionarla con el botón 2, el cual es el encargado de entrar en el modo de reproducción de la “nueva canción”.

Podemos, una vez dentro de la canción, entrar en modo de pausa, es decir, guardar la posición en cuanto al lector del fichero de texto y del reproductor de audio y posteriormente, reanudar la reproducción. Además, en caso de querer abandonar la canción y volver al menú de la lista de canciones disponibles, podríamos hacerlo pulsando el botón 3. En este caso no se guardaría la posición del fichero de texto o del reproductor de audio puesto que se ejecutaría como un STOP, no como un PAUSE.

3.3. Conclusión final del segundo modelo

Tras el fracaso de la primera versión del proyecto, fue necesario cambiar de placa, implementar las mejoras propuestas en la versión anterior y corregir errores en unos pocos días.

Por ello, y en primer lugar, queremos agradecer a los profesores de laboratorio que nos pudieron prestar una STM32F411, modelo que no sufría problemas en la conexión USB. Esto nos permitió comprobar que el problema principal de la versión anterior se debía no a las posibilidades que sugerimos anteriormente, si no al microcontrolador utilizado.

En segundo lugar, debemos mencionar también la importancia de usar un código de base facilitado también por nuestro tutor en vez de intentar corregir y adaptar la primera versión del proyecto. Esto nos permitió, tras leer y estudiar las librerías y código sobre las que trabajaríamos, ser mucho más rápidos para implementar funcionalidades y corregir errores. Además, nos daba una base sólida desde donde volver a empezar en caso de ser necesario. Por último, aunque algunas de nuestras funciones de la primera versión perdieron relevancia, otras se adaptaron con facilidad al nuevo código encargado de la reproducción de sonido.

Por último, en cuanto al resultado final del proyecto, encontramos tanto funcionalidades implementadas tal y como nos gustaría como otras que sería necesario mejorar. Por un

lado, las modificaciones a la lectura de archivos funcionan de forma correcta, y permiten mayor flexibilidad con la estructura de archivos que en la primera versión. El control del volumen es más sencillo al depender de una variable enviada por protocolo I2S en lugar de utilizar directamente un PWM y la pantalla LCD muestra adecuadamente la información. Por otro lado, encontramos dos errores que desincronizan música y letra al reproducir una canción después de acabar o salir de otra. Después de esta situación, era necesario darle a PAUSE y una vez más a PLAY para que se reproduciera el sonido, que además se ralentiza. Este error parece tener su origen en la librería que utilizamos como base, y que no contemplaba la opción de que se emplease un STOP para salir de una canción. El estado de STOP estaba pensado inicialmente para que se entrase al acabar las canciones disponibles. Tras revisar varias veces ese estado y compararlo con el de PAUSE, no hemos visto diferencias observables, pero existe la posibilidad de que se hayan pasado por alto cambios en alguna variable interna de las funciones de la librería.