

# **PRÁCTICA 5**

## **Índice**

1. Objetivos.
2. Introducción.
3. Desarrollo de la práctica.
4. Conclusión.

## **1. Objetivos**

- Fase 1: MBED, estudio del entorno de desarrollo, utilizar una interrupción periódica y el display LCD.
- Fase 2: Realización de un dispositivo de clase AUDIO de salida, tipo altavoz.
- Fase 3: Añadir al dispositivo anterior efectos de retraso (delay).
- Fase 4: Realización de un dispositivo de clase MIDI de entrada, tipo teclado.
- Fase 5: Realización de un dispositivo de clase MIDI de salida, tipo sintetizador sonidos.

## **2. Introducción**

Esta práctica tiene dos objetivos académicos bien diferenciados:

1 - Profundizar en el conocimiento de los sistemas de desarrollo y sus posibilidades. Para ello se introduce en este texto un sistema de desarrollo “online” denominado MBED. En este curso sólo vamos a trabajar con el sistema MBED en esta práctica 5, en prácticas posteriores utilizaremos el ya conocido STM32CubeIDE. El propósito de introducir MBED, en al menos una práctica de SETR1, es que el/la alumno/a conozca algunas de las singularidades de los sistemas de desarrollo en la nube. Los sistemas de desarrollo online se han vuelto en los últimos años bastante populares, por ser mecanismos de prototipado rápido con los que se consiguen realizar proyectos complejos en relativamente poco tiempo. Como siempre, hay plataformas “online” de desarrollo gratuitas y de pago, la que usaremos en esta práctica es una de las más populares, siendo gratuita.

2-Introducir al microcontrolador como periférico USB del PC. En concreto desarrollaremos una serie de dispositivos USB de clase AUDIO y MIDI.

### **MBED entorno de desarrollo online.**

Los microcontroladores, al igual que el resto de sistemas digitales, han evolucionado hacia mayores prestaciones y posibilidades, pero a costa de una mayor complejidad para el desarrollo del correspondiente software. Afortunadamente el avance en este campo ha sido constante desde que aparecieron los microcontroladores, atrás quedaron los tiempos en el que la única posibilidad era utilizar el ensamblador y depurar a base de LEDs/puertos serie.

Un aspecto importante a considerar es que la mayoría de los desarrolladores al final utilizan los mismos componentes de forma repetida, no parece demasiado lógico que cada vez que se inicia un proyecto tenga que empezarse desde cero. En este sentido los fabricantes de microcontroladores, y de sistemas de desarrollo, también evolucionan proporcionan más facilidades a base de plantillas, librerías y “wizards” generadores de código automáticos (como el del STM32CubeIDE), que le permiten al diseñador no empezar de cero.

En esta práctica vamos a introducir brevemente el funcionamiento de una herramienta que en cierta forma representan en la actualidad el máximo exponente en cuanto a facilidades para el desarrollador para ARM, con el mínimo coste económico (son gratuitas) y sin necesidad de instalación en el PC, simplemente usando un navegador web. Como ya se ha indicado en las clases de teoría, es lógico concluir que prácticamente todos los sistemas de desarrollo de microcontroladores se basan en compiladores cruzados, pero estos IDE en la nube van mucho más allá cuando nos ofrece un compilador remoto. La principal ventaja de un entorno online es el poder disponer de una amplia comunidad de desarrolladores, con una biblioteca de programas y librerías en constante crecimiento. Esto supone que para hacer cualquier

desarrollo contamos con multitud de plantillas, y lo más probable es que ya alguien haya hecho algo parecido a lo que necesitamos, con lo que ya no tenemos que arrancar de cero. Por otra parte, MBED, en particular, proporciona almacenamiento en la nube a nuestros proyectos por tiempo ilimitado, estos además se pueden exportar a entornos locales y seguir el desarrollo con los IDE instalados en nuestros ordenadores.

No todo son ventajas para estas plataformas online gratuitas, también tienen inconvenientes: conlleva un cierto grado de caos, al mismo tiempo que el código que nos proporciona no está exento de “chapuzas” y errores, todo ello inherente a su carácter gratuito y abierto (en el que mucha gente “mete mano” sin control). Pero obviando estos “pequeños” inconvenientes no deja de ser sistemas muy útiles y pioneros en el desarrollo de microcontroladores.

El nuevo sistema de MBED/KEIL STUDIO CLOUD permite no solo utilizar los repositorios de MBED y sus librerías, si no también el código para MCU, cada vez más numeroso, que encontramos en GitHub... y no sólo con las librerías MBED, si no también con las CMSIS standard de ARM, pudiendo enlazarnos a nuestros repositorios en GitHub. Por otra parte todos los fuentes oficiales de Mbed se encuentran ahora en GitHub, esta plataforma ha ido con los años conteniendo cada vez más software público para Microcontroladores, por lo que parece lógico la decisión de ARM de facilitar su uso en el nuevo entorno Keil Studio Cloud. Con lo que contamos con todo el software que a lo largo de los años se ha realizado con las librerías de Mbed, más todo el disponible en GitHub.

MBED es en realidad una plataforma de prototipado rápido, en la que facilitan al desarrollador todo lo necesario para hacer un prototipo de sus ideas en el mínimo tiempo posible. Está auspiciada por ARM, por lo que estos son los microcontroladores con los que se puede trabajar en esta plataforma (soporta todos los Cortex M modernos de casi todos los fabricantes). Para estudiarla, podemos dividir dicha plataforma en tres apartados:

1-Sistema operativos y librerías de apoyo, se dispone de diversas versiones, pero en realidad hay dos líneas fundamentales:

- o MBED-OS para Cortex M (los de este curso). En particular el MBED-OS lo hay con soporte multitarea tiempo real o sin dicho soporte.

- o MBED Linux OS para Cortex A.

2-IDEs y compiladores, dispone de tres tipos:

- MBED Online Compiler, el que lo ha hecho popular, aunque en el año 2022 sufrió un cambio drástico y pasó a llamarse Keil Studio Cloud, precisamente éste es el entorno sobre el que se va a realizar esta práctica.

- MBED Studio, es un IDE para usar en local (en nuestro PC), hasta hace poco no era una herramienta eficiente y fiable, ahora parece que funciona correctamente.

-MBED CLI un sistema local de compilador en línea de comandos, algo primitivo pero que sirve de apoyo a otros proyectos de sistemas de desarrollo (al propio MBED Studio).

3-Hardware. MBED está pensado sólo para desarrollar con microcontroladores tipo ARM Cortex. No obstante, es tan popular dicha arquitectura que podemos encontrar placas soportadas por MBED de la mayoría de fabricantes, tenemos que recordar que ARM diseña el núcleo del microcontrolador, que después cada fabricante lo “coloca” en su producto con diversas características y periféricos. Es por eso que, a pesar de ser exclusivo para la plataforma Cortex, se autodefine como abierto, pues cualquier fabricante puede incluir sus productos en esta plataforma. Con respecto al hardware, la metodología de diseño está basada en “placas ejemplos”, que permiten al diseñador elegir uno de esos ejemplos para hacer su prototipo y posteriormente copiar sus esquemas para integrarlo en su diseño final, es lo que hemos comentado siempre de hacer “la lavadora” con la placa de evaluación (tipo Arduino) y después hacer una placa parecida que sería la que finalmente iría en el producto comercial. El procedimiento es tan sencillo y el coste tan bajo que ha hecho que miles de aficionados usen Mbed para sus proyectos personales sin ningún tipo de interés de fabricación posterior. Las placas de ejemplo directamente soportadas por MBED se pueden dividir en tres tipos:

-Boards. Son las típicas placas de evaluación que diseñan y venden, a bajo precio, los fabricantes de microcontroladores para que los desarrolladores (y estudiantes/aficionados) evalúen sus productos, al mismo tiempo que pueden servir como un primer prototipo en un diseño (de este tipo es la que tenéis prestadas los alumnos). El precio de estas placas suele oscilar de 10 a 50 euros, dependiendo de sus características y de donde se compren. Estas placas de evaluación pueden ser muy diversas, pero todas tienen una característica común, y es la tecnología de grabado y depuración que se emplea en los nuevos Cortex. En MBED se utiliza un mecanismo denominado DAPlink, que es similar y compatible al que se usa por parte de otros fabricantes como pueden ser OPENSDA, STLINK, JLINK, etc. Todos estos depuradores/grabadores se conectan al PC mediante un puerto USB como un dispositivo compuesto (perteneciente a varias clases), de forma que se ve la placa desde el PC como un disco de almacenamiento masivo, como un COM virtual y como un depurador clásico de microcontroladores. La grabación de la memoria FLASH se hace de forma tan fácil como la de cargar la imagen binaria en el disco virtual correspondiente a la placa. Cuando la plataforma MBED compila nuestro programa editado en su IDE, el resultado es un fichero .bin, que se puede descargar directamente en dicho disco virtual, y con eso ya está grabado en la memoria del microcontrolador. Por otra parte, el COM virtual disponible permite su uso como consola de depuración para enviar mensajes al PC mediante “printf”.

-Modules. En esta categoría están ciertas placas de comunicaciones (para IOT).

-Components. Suelen ser placas con periféricos (sensores, actuadores, etc.) que se pueden añadir a las Boards para ampliar sus posibilidades, en algunos casos

son como los “shields de Arduino” y en otros son placas autónomas con su propio microcontrolador.

Mbed siempre ha estado en constante evolución, sus librerías y el mismo entorno de desarrollo han sufrido muchos cambios. Esto supone una cierta complicación para el desarrollador, pues las diferentes revisiones de las librerías y del entorno de desarrollo ha hecho que el mantenimiento de la compatibilidad hacia atrás sea un verdadero infierno, tanto para los “dueños” de MBED como para los diseñadores que pretenden seguir esos cambios. El gran problema del viejo Mbed Online Compiler era la depuración del código, como es natural las limitaciones del tráfico en la red y de los servidores no permitían hacer una depuración en remoto, el sistema de depuración básico que se utilizaba era el del “printf” mediante ese COM virtual mencionado anteriormente. Pero esto cambió en el año 2022, ahora el Keil Studio Cloud permite depuración en remoto con un funcionamiento relativamente bueno.

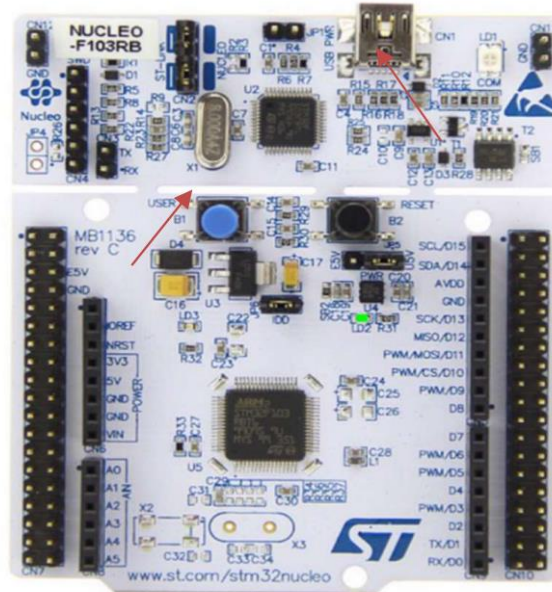
A continuación, se muestra la imagen de una placa de desarrollo de STM, observad que hay una división entre la parte del depurador y la del microcontrolador, la placa está “precortada”, esto es así por si queremos usar la placa en un producto final, desprender la parte del depurador y dejar sólo la del microcontrolador, claro está, que una vez que hagamos eso reprogramar la placa no será tan fácil... aunque se puede (Modo 1 de arranque). Evidentemente la parte superior de depuración es equivalente a las cajas blancas que vimos en el laboratorio, simplemente que en lugar de ser un dispositivo externo se ha añadido a la misma placa. Es esa parte de arriba con el conector USB la que hace esas tres funciones al mismo tiempo:

1-Puerto Com virtual (puede requerir instalar driver o no).

2-Depurador tradicional de microcontroladores (requiere instalar driver).

3-Grabador FLASH como unidad de almacenamiento masivo virtual (como una llave USB, no requiere instalar driver).

Esta puede ser la función más llamativa y la que hace que este tipo de placas en MBED sean muy fáciles de usar. En el Keil Studio Cloud podemos compilar, generar el fichero .bin , descargarlo en nuestro PC y “guardarlo” en la placa como si de una unidad de disco duro se tratara. En realidad, como se ha dicho, es un grabador de FLASH, en el que cada vez que guardamos un fichero .bin lo que hacemos es grabar la memoria FLASH del microcontrolador borrando el anterior código (o fichero .bin), es como un disco duro que nunca se llenase, pero del que tampoco podemos recuperar nada, pues los ficheros .bin que grabamos no se pueden leer por este procedimiento. Este mecanismo era el único que tenía el viejo compilador MBED para grabar el firmware en el MCU, pero el nuevo Keil Studio Cloud puede grabar directamente, e incluso grabar y depurar.



La placa que usamos en este curso no tiene esa estructura física de la imagen anterior, pero su funcionamiento es exactamente el mismo.

## **Dispositivos USB con microcontroladores**

La conexión USB se explica con detalle en las correspondientes clases de teoría, aquí vamos a tratar los conceptos generales mínimos para poder realizar la práctica. Lo primero que nos debemos percatar es que la conexión USB es para el usuario relativamente simple y universal, pero precisamente esas dos características frente al usuario hacen que estos sistemas sean bastantes complejos para el desarrollador... y nosotros somos desarrolladores.

Puntos importantes que conocer de los sistemas USB:

1-Los dispositivos USB se dividen en Host y Device (o dispositivo). El típico Host suele ser nuestro PC, y el Device suele estar implementado con un dispositivo inteligente o complejo (Microcontrolador, FPGA...). El Device se entiende que es un periférico del Host. Existen dispositivos USB que pueden cambiar su “personalidad” entre Host y Device, por ejemplo, los llamados OTG (conector teléfono móvil), que no trataremos en este curso.

2-En general los periféricos de los computadores tradicionales se clasifican según lo que hagan (su función) por “clases”. Los fabricantes de hardware y sistemas operativos para máquinas de propósito general se han puesto de acuerdo para organizar y clasificar el funcionamiento de los periféricos creando lo que se denominan “clases estándar”. Esto implica que para que un periférico pertenezca a una clase estándar debe cumplir las “normas escritas” de dicha clase estándar. Por ejemplo, clases estándar son: HID

(ratones), Almacenamiento masivo (“llaves” memorias USB, disco duro), Audio (tarjetas de sonido)... Los periféricos cuyo funcionamiento no se ajustan a ninguna norma de clase estándar se dicen que son de clase específica de fabricante o vendedor. Por tanto, fijándonos en su clase podemos dividir los periféricos USB en dos tipos:

- Los pertenecientes a clases genéricas (o standard). No precisan añadir drivers, el sistema operativo los reconoce directamente, ya contienen el correspondiente driver de clase. Normalmente son periféricos que requieren algo más de complejidad; pero, por el contrario, al tener ya el driver, son más simples desde el punto de vista del “host”: sólo requieren la aplicación (y en algunos casos ni tan siquiera eso: por ejemplo, ratones, tarjetas de sonido ...). Precisamente en esta práctica vamos a hacer dispositivos de clase Audio (estándar) y el driver del Host que maneja el periférico ya está en el sistema operativo.

- Los pertenecientes a clases específicas de fabricante (o vendedor). Estos dispositivos necesitan que se le proporcione un driver para que funcionen, pues el sistema operativo del “host” carece del mismo por defecto. Los periféricos pueden ser más simples, al no requerir adecuarse a la norma de una clase estándar, pero precisan algo más de software para el “host”: el driver y la aplicación.

3-Los Device USB se caracterizan por autoconfigurarse. Cuando se conecta un periférico USB al PC éste manda su configuración al Host (PC), de forma que este último lo asocia a la clase que corresponda. Cuando enchufamos a nuestro PC una llave de memoria USB ésta es reconocida por el PC como de clase almacenamiento masivo y trata a dicha llave como un “disco”. El proceso por el que el HOST identifica al Device es complejo, pero se basa en que este último manda una serie de datos estructurados al Host, que permiten que el Host sepa lo necesario para clasificar al Device y hacerlo funcionar de forma adecuada. Esos datos estructurados que manda el Device al Host se denominan descriptores, y en nuestro caso serán datos que deben contener nuestros microcontroladores para que cuando se requieran sean enviados al PC. El principal problema para hacer funcionar adecuadamente un Device USB es que dichos descriptores sean correctos (según las normas), cualquier fallo en los descriptores implica que el Host rechace al Device, y nos aparezca el mensaje de “el dispositivo USB no es correcto”, con lo que no podremos usarlo en el PC. En resumen, desde el punto de vista del desarrollador de software USB para microcontroladores hay dos elementos a tener en cuenta:

- Introducir en nuestros programas los descriptores adecuados y el manejo de los mismos.

- Llenar o vaciar los buffers de comunicaciones para enviar o recibir los datos adecuados, estos buffers se ven como EndPoint (EP seguido de un número), que son conexiones virtuales entre el HOST y el device.

4-El Host (PC) siempre es el que inicia y controla la comunicación, es el Máster, y el Device USB es por tanto el Slave. Por convención se denominan a las transferencias

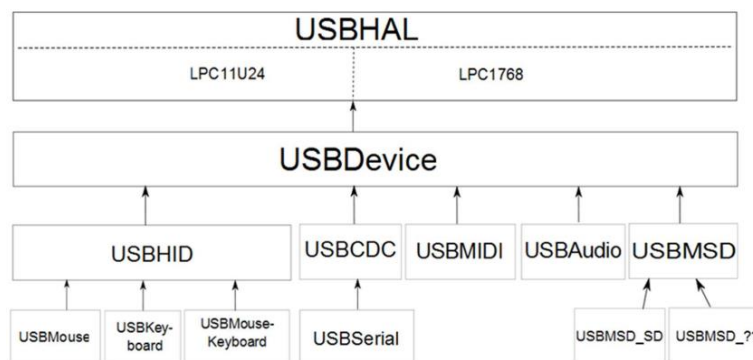
desde el Master al Slave: transferencias tipo OUT, y a los EP por los que se hace dichas transferencias se les denomina EP OUT. De igual forma, por convención, las transferencias desde los Slave (Device) a los Máster (Host) se denominan transferencia tipo IN, y los EP correspondientes como EP IN. Es decir, el carácter IN o OUT de la transferencia lo marca el Máster, si entran o salen los datos del Máster, INDEPENDIENTEMENTE DE QUE ESTEMOS HACIENDO PROGRAMAS PARA EL PC O PARA EL MICROCONTROLADOR. Esto conlleva una enorme confusión cuando somos desarrolladores de programas USB para uC, puesto que en nuestros programas cuando aparece IN en realidad es salida para el microcontrolador y cuando aparece OUT en realidad es entrada para el microcontrolador. Y todavía peor, hay desarrolladores que le cambian el nombre en sus programas y aún es más difícil saber lo que hace... si son EP de salida o de entrada.

5-Los descriptores tienen una estructura jerárquica que se explicara en las clases de teoría, todos los dispositivos USB tienen al menos 4 tipos de descriptores: Device, Configuración, Interface y EndPoint. Puede haber otros tipos de descriptores dependiendo a qué clase pertenezcan.

## **USB en MBED**

Las librerías de MBED que vamos a utilizar en esta práctica es la USBDEVICE, cuya estructura de capas se muestra en la siguiente figura. Además, hay múltiples programas de ejemplo que utilizan dichas librerías y con los que casi se puede hacer todos los dispositivos planteados en esta práctica:

Explicaciones de los diferentes módulos que componen el sistema se encuentra en <http://developer.mbed.org/users/samux/notebook/usbdevice-stack-architecture/>



Puntos importantes a destacar de este firmware:

-Cuatro niveles de capa, las capas USBHAL y USBDEVICE casi siempre la proporcionan los fabricantes, lo que aporta más el MBED son las otras dos capas de dispositivos de clase genérica. La capa USBHAL no conviene tocarla casi nunca, es la que maneja el hardware del microcontrolador directamente, si se necesita cambiar los ENDPOINT o buffers de los mismos puede ser necesario tocar dicha capa. La capa USBDEVICE siempre hay que modificarla en lo que respecta a sus correspondientes descriptores,



sobre todo para modificar los ID, pero el MBED facilita precisamente dichos cambios a la hora de inicializar el device (se pasan los ID como parámetros).

-En el módulo USBDEVICE se encuentra el descriptor de DEVICE y los métodos básicos para leer y escribir los ENDPOINT, que después las sucesivas capas utilizan, transforman y enmascaran: readEP, readEP\_NB, write, writeNB. CONSEJO PARA UTILIZAR USB EN CUALQUIER ENTORNO DE MICROCONTROLADORES DE CUALQUIER FABRICANTE: EN EL FIRMWARE USB SIEMPRE HAY QUE BUSCAR LOS DESCRIPTORES Y LAS FUNCIONES/METODOS QUE NOS PERMITEN ACCEDER A LOS BUFFERS DE LOS ENDPOINT (LEER Y ESCRIBIR EN ELLOS).

-En la tercera capa, la correspondiente a las clases genéricas se encuentra el descriptor de Configuración, que engloba: al descriptor de configuración, al de interface, al de endpoint y a los propios de cada clase.

-Como se puede observar en la figura, el firmware USB básico de MBED no está preparado para hacer dispositivos de clase específica de fabricante. Pero es fácil modificar el firmware (fundamentalmente los descriptors) de cualquier clase genérica y convertirlo en específica de fabricante.

### **Dispositivos USB clase audio**

Los dispositivos de clase audio se dividen en tres subclases: audiostreaming, midistreaming y audiocontrol. La subclase

audiocontrol se encuentra en todos los dispositivos de audio. Desde el punto de vista coloquial son las otras dos

subclases las que dividen a estos periféricos USB en dispositivo AUDIO o dispositivo MIDI. En esta práctica utilizaremos la denominación dispositivo AUDIO para referirnos a los de clase audiostreaming, y dispositivo MIDI para cuando tratemos la clase midistreaming.

La principal diferencia entre ambos tipos de dispositivo radica en el formato de la información sonora que maneja. Un dispositivo USB AUDIO maneja datos correspondientes al muestreo de una señal analógica que representa el sonido (la clásica conversión analógico/digital), mientras que un dispositivo USB MIDI los datos que maneja hacen referencia a las notas e instrumentos musicales (parecido a una partitura).

## **3. Desarrollo de la práctica**

▪ En la fase 1, primero nos registramos en la plataforma. Luego, elegimos nuestra placa para comenzar a trabajar. Tras configurar todo, comprobamos si funciona correctamente escribiendo este código en el “main.cpp”:

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

La primera vez que intentemos ejecutarlo, nos informará que no reconoce nuestra placa. Para resolver este problema, añadimos la librería de mbed al repositorio. También aprovecharemos para incluir las librerías relacionadas con el LCD.

Después de añadirlas, procederemos a inicializar el código teniendo en cuenta la tabla que se nos proporciona en el documento de la práctica:

Nombre actual	Arduino	Pin MCU	Tipo de puerto		Nombre nuevo que hay que poner
ARD_D10	D10	PA2	Output Push Pull	Low	Led_LCD
ARD_D9	D9	PA15	Output Push Pull	Low	E_LCD
ARD_D6	D6	PB1	Output Push Pull	Low	D6_LCD
ARD_D5	D5	PB4	Output Push Pull	Low	D5_LCD
ARD_D8	D8	PB2	Output Push Pull	Low	RS_LCD
ARD_D7	D7	PA4	Output Push Pull	Low	D7_LCD
ARD_D4	D4	PA3	Output Push Pull	Low	D4_LCD

Una vez comprobado que la hemos seguido bien, escribimos el siguiente código

```
#include "mbed.h"
#include "TextLCD.h"

DigitalOut myled(LED1);

DigitalOut ledlcd(D10);
TextLCD lcd(D8, D9, D4, D5, D6, D7);

int main() {

    ledlcd=1;

    while(1) {

        float f = ain.read();

    }
}
```

```

    myled = 1; // LED is ON
    wait(0.5); // 200 ms
    myled = 0; // LED is OFF
    wait(3.0); // 1 sec
    lcd.locate(0,0);
    lcd.printf("Hola mundo");

}

}

```

Toca leer los botones del LCD. Para ello, declarar AnalogIn. Además, usaremos el objeto Ticker intermi para hacer interrupciones periódicas. Para ello modificaremos el “main.cpp”:

```

#include "mbed.h"
#include "TextLCD.h"

DigitalOut myled(LED1);
DigitalOut otroled(LED2);

TextLCD lcd(D8, D9, D4, D5, D6, D7);
DigitalOut ledlcd(D10);

AnalogIn ain(PC_5);

Ticker intermi;

void mifuninterrupt(){
    otroled = !otroled;
}

int main() {

    ledlcd=1;

    intermi.attach(&mifuninterrupt, 0.5);

    while(1) {

        float f = ain.read();

        myled = 1; // LED is ON
        wait(0.5); // 200 ms
        myled = 0; // LED is OFF
        wait(3.0); // 1 sec
        lcd.locate(0,0);
        lcd.printf("%f", f);

    }

}

```

- En la fase 2, hacemos una copia de la práctica anterior. Cogemos como plantilla un programa llamado USBAUDIO\_speaker. Además de añadir la librería USBDevice.

Copiaremos y pegaremos el “main.cpp” del USBAUDIO\_speaker en el nuestro , modificando algunas cosas quedando así:

```
// USBAudio speaker example

#include "mbed.h"
#include "USBAudio.h"

// frequency: 48 kHz
#define FREQ 24000

// 1 channel: mono
#define NB_CHA 1

// length of an audio packet: each ms, we receive 48 * 16bits ->48 * 2 bytes. as there is one
channel, the length will be 48 * 2 * 1
#define AUDIO_LENGTH_PACKET 24 * 2 * 1

// USBAudio (we just use audio packets received, we don't send audio packets to the
computer in this example)
USBAudio audio(FREQ, NB_CHA, 8000, 1, 0x7180, 0x7500);

// speaker connected to the AnalogOut output. The audio stream received over USb will be
sent to the speaker
AnalogOut speaker(PA_5);

// ticker to send data to the speaker at the good frequency
Ticker tic;

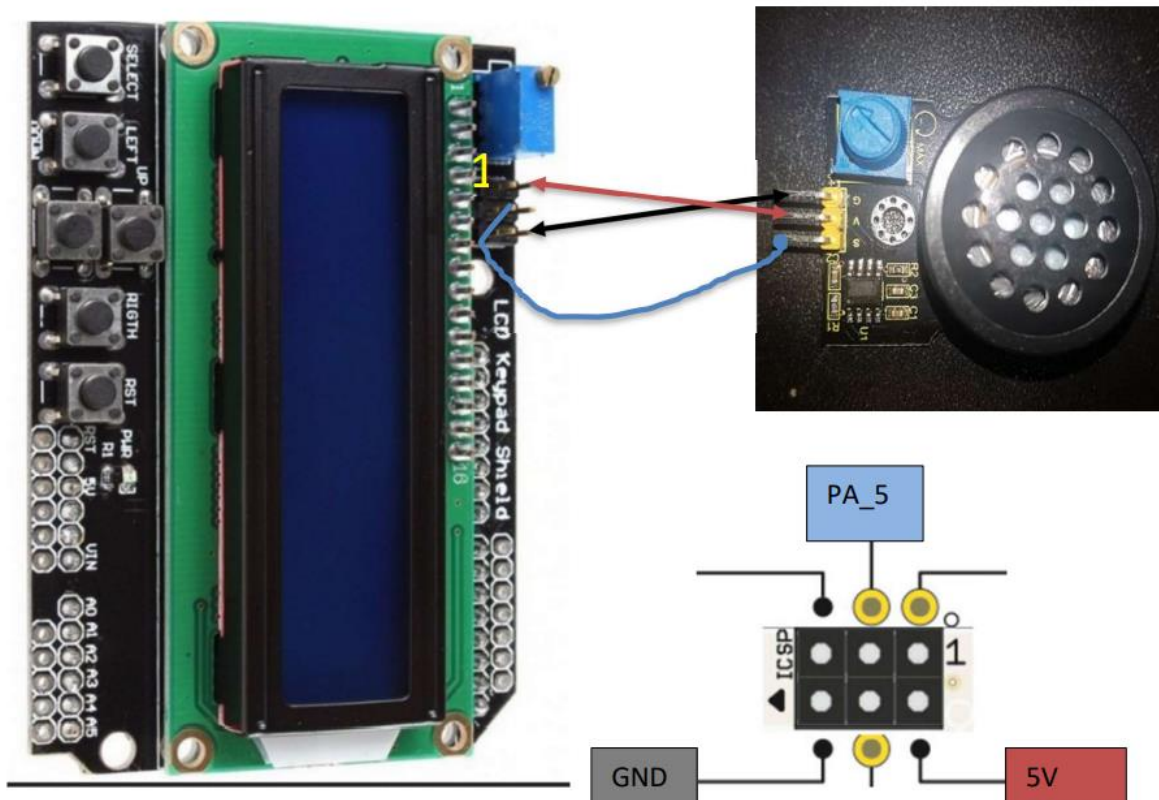
// buffer where will be store one audio packet (LENGTH_AUDIO_PACKET/2 because we are
storing int16 and not uint8)
int16_t buf[AUDIO_LENGTH_PACKET/2];

// show if an audio packet is available
volatile bool available = false;

// index of the value which will be send to the speaker
int index_buf = 0;

// previous value sent to the speaker
uint16_t p_val = 0;
```

Una vez realizado esta serie de pasos, conectaremos el altavoz a nuestra placa basándonos en la imagen proporcionada en la placa:



Luego, conectaremos los dos cables entre la placa y el PC, ya que uno es para escribir el código en la placa y el otro para hacer uso de ella. Tras esto, estaremos listos para usar nuestra placa como dispositivo de audio.

También podremos añadir en el main() del “main.cpp” la siguiente línea de código para variar el efecto de distorsión:

```
// attach a function executed each 1/FREQ s
tic.attach_us(tic_handler, 1000000.0/(float)FREQ);
```

▪ En la fase 3, añadiremos delay al sonido para crear un efecto de eco. Para ello, incluiremos en el “main.cpp” el siguiente código:

```
int16_t buf[AUDIO_LENGTH_PACKET/2];
int16_t mibuf [256][AUDIO_LENGTH_PACKET/2];

//Inicializaciones
float ganancia = 0.;
uint8_t efecto = 0;
uint8_t j = 0;
```

Además de modificar la función `tic_handler()` quedando así:

```
void tic_handler() {
    float speaker_value;

    if (available) {
        //convert 2 bytes in float
        speaker_value = ((float)(buf[index_buf])+ (float) (ganancia*mibuf[(unsigned char)(j-efecto))][index_buf])+(float)
        (ganancia*mibuf[(unsigned char)(j-efecto-1)][index_buf]))/3 ;

        // speaker_value between 0 and 65535
        speaker_value += 32768.0;

        // adjust according to current volume
        speaker_value *= audio.getVolume();

        // as two bytes has been read, we move the index of two bytes
        index_buf++;

        // if we have read all the buffer, no more data available
        if (index_buf == AUDIO_LENGTH_PACKET/2) {
            index_buf = 0;
            available = false;
        }
    } else {
        speaker_value = p_val;
    }

    p_val = speaker_value;

    // send value to the speaker
    speaker.write_u16((uint16_t)speaker_value);
}
```

Por último, modificamos el `while(1)` quedándonos así:

```
while (1) {
    // read an audio packet

    audio.read((uint8_t *)buf);

    for (int i=0;i<24;i++)
        mibuf[j][i]=buf[i];

    j++;

    available = true;
}
```

Después de esto, queda implementar un sistema más ameno usando el LCD. Después de las implementaciones para controlar la ganancia y el efecto, el archivo “main.cpp” quedará así:

```
// USBAudio speaker example

#include "mbed.h"
#include "USBAudio.h"
#include "TextLCD.h"

DigitalOut myled(LED1);

TextLCD lcd(D8, D9, D4, D5, D6, D7);
DigitalOut ledlcd(D10);

AnalogIn ain(PC_5);

Ticker intermi;

// frequency: 48 kHz
#define FREQ 24000

// 1 channel: mono
#define NB_CHA 1

// length of an audio packet: each ms, we receive 48 * 16bits ->48 * 2 bytes. as there is one
// channel, the length will be 48 * 2 * 1
#define AUDIO_LENGTH_PACKET 24 * 2 * 1

// USBAudio (we just use audio packets received, we don't send audio packets to the
// computer in this example)
USBAudio audio(FREQ, NB_CHA, 8000, 1, 0x7180, 0x7500);

// speaker connected to the AnalogOut output. The audio stream received over USB will be
// sent to the speaker
AnalogOut speaker(PA_5);

// ticker to send data to the speaker at the good frequency
Ticker tic;
Ticker pantalla;

// buffer where will be store one audio packet (LENGTH_AUDIO_PACKET/2 because we are
// storing int16 and not uint8)
int16_t buf[AUDIO_LENGTH_PACKET/2];
int16_t mibuf [256][AUDIO_LENGTH_PACKET/2];

// show if an audio packet is available
volatile bool available = false;
```

```

// index of the value which will be send to the speaker
int index_buf = 0;

// previous value sent to the speaker
uint16_t p_val = 0;

//Inicializaciones
float ganancia = 0.;
uint8_t efecto = 0;
uint8_t j = 0;

void mifuninterrupt(){

    float f = ain.read();

    if (f > 0.69 && f < 0.73)
        ganancia = ganancia - 0.5;
    else if (f == 0.0)
        ganancia = ganancia + 0.5;
    else if (f > 0.18 && f < 0.21)
        efecto = efecto - 0.5;
    else if (f > 0.43 && f < 0.48)
        efecto = efecto + 0.5;

}

void botones(){

    float valor = boton.read();

    if ((valor > (float) 0.01) && (valor < (float) 0.2)){

        ganancia = ganancia + (float) 1.0;
        lcd.cls();
        lcd.printf("Efecto: %d, Ganancia: %.1f", efecto, ganancia);

    }

}

// function executed each 1/FREQ s
void tic_handler() {
    float speaker_value;

    if (available) {
        //convert 2 bytes in float
        speaker_value = ((float)(buf[index_buf]) + (float) (ganancia*mibuf[(unsigned char)(j-
efecto)])[index_buf]) + (float)

```



```

(ganancia*mibuf[(unsigned char)(j-efecto-1)][index_buf]))/3 ;

// speaker_value between 0 and 65535
speaker_value += 32768.0;

// adjust according to current volume
speaker_value *= audio.getVolume();

// as two bytes has been read, we move the index of two bytes
index_buf++;

// if we have read all the buffer, no more data available
if (index_buf == AUDIO_LENGTH_PACKET/2) {
    index_buf = 0;
    available = false;
}
} else {
    speaker_value = p_val;
}

p_val = speaker_value;

// send value to the speaker
speaker.write_u16((uint16_t)speaker_value);
}

int main() {
    wait(0.2);

    ledlcd=1;

    intermi.attach(&mifuninterrupt, 0.5);

    // attach a function executed each 1/FREQ s
    tic.attach_us(tic_handler, 1000000.0/(float)FREQ);

    pantall.attach(&botones, 10.0);

    while (1) {
        // read an audio packet
        audio.read((uint8_t *)buf);

        for (int i=0;i<24;i++)
            mibuf[j][i]=buf[i]; //se actualiza la fila con los datos recién llegados

        j++; // se apunta a la siguiente fila

        available = true;
    }
}

```

```

lcd.locate(0,0); //permite elegir fila y columna para la escritura siguiente con printf en el LCD
lcd.printf("Ganancia: %i%%", (int)ganancia*2);

lcd.locate(0,1); //permite elegir fila y columna para la escritura siguiente con printf en el
LCD
lcd.printf("Efecto: %i%%", efecto*100/255);

}
}

```

- En la fase 4, utilizaremos un dispositivo de tipo MIDI de entrada, como por ejemplo, un teclado. Instalaremos el programa VMPK piano virtual siguiendo los pasos de la práctica y lo configuramos. Seleccionamos la pestaña de edición y en el menú desplegable seleccionamos “conexiones”. Debemos de habilitar la entrada MIDI y “copiar la entrada MIDI en la salida”. Como dispositivo de entrada, seleccionamos Mbed Audio y como salida, el generador de sonidos de nuestro ordenador. Usaremos la librería USBMIDI para programar el código. El “main.cpp” queda así tras los cambios:

```

//Hello World example for the USBMIDI library

#include "mbed.h"
#include "USBMIDI.h"
#include "TextLCD.h"

unsigned long botones;
USBMIDI midi;

TextLCD lcd(D8, D9, D4, D5, D6, D7);

AnalogIn boton(PA_3);

AnalogIn p(PC_3);

int main() {

    while(1) {

        botones = boton.read_u16();

        midi.write(MIDIMessage::ProgramChange(100));
        midi.write(MIDIMessage::ControlChange(7, 127*p.read()));

        if (botones > 61000 && botones < 63000) {

            midi.write(MIDIMessage::NoteOn(1));

            wait(0.20);

```

```

midi.write(MIDIMessage::NoteOff(1));

    } else if (botones > 24000 && botones < 26000) {

        midi.write(MIDIMessage::NoteOn(2));

        wait(0.20);

        midi.write(MIDIMessage::NoteOff(2));

    } else if (botones > 8000 && botones < 10000) {

        midi.write(MIDIMessage::NoteOn(3));

        wait(0.20);

        midi.write(MIDIMessage::NoteOff(3));

    } else if (botones > 39000 && botones < 41000) {

        midi.write(MIDIMessage::NoteOn(4));

        wait(0.20);

        midi.write(MIDIMessage::NoteOff(4));
    } else if (botones > 0 && botones < 1000) {

        midi.write(MIDIMessage::NoteOn(5));

        wait(0.2);

        midi.write(MIDIMessage::NoteOff(5));

    }

}







}

}

```

- Para la fase 5, utilizaremos un dispositivo de clase MIDI de salida, tipo sintetizador sonidos. Importamos USBMIDI\_MonoSynth, pero al buscarlo no aparecerá el indicado en el documento de la práctica, que es el de Fuminore ALAI, por lo que no he podido completar esta fase.

6 results found for USBMIDI\_MonoSynth

 <b>Jannes De Brabandere / USBMIDI</b> Added Midi Clock Types	Last updated: 19 Aug 2014 👉 4 👤 4
 <b>Kristen Fernandez / USBMIDI</b> Changed file extensions from .c to .cpp	Last updated: 29 Apr 2016 👉 4 👤 8
 <b>Simon Ford / USBMIDI</b> A library to send and receive MIDI messages over USB using the default USB-MIDI drivers on Win/Mac 📦 library, MIDI, USB, USBMIDI	Last updated: 20 Feb 2011 👉 3 👤 1430
 <b>ADDAC System / USBMIDI</b> A library to send and receive MIDI messages over USB using the default USB-MIDI drivers on Win/Mac 📦 library, MIDI, USB, USBMIDI	Last updated: 23 Feb 2016 👉 5 👤 7
 <b>Chris Arndt / Mbed OS STM32F103_USB MIDI_Switchbox</b> Simple USB-MIDI foot controller	Last updated: 09 Jan 2018 👉 14 👤 44
 <b>Auzmendi Agustin / DISCO_F746NG_USBDevice</b> usbmidi	Last updated: 19 Mar 2020 👉 2 👤 13

Aun así, la realización es muy similar a la fase 4, por lo que una vez escrito el código solo habría que configurar el MIDI en base a las indicaciones.

## **4. Conclusión**

Gracias a las indicaciones de la hoja de prácticas se han podido completar satisfactoriamente todas las fases, quitando la última, que no he podido encontrar el programa “USBMIDI\_MonoSynth”.

Personalmente, la práctica ha sido la más extensa hasta ahora, incluso pesada hasta en ciertos puntos en comparación a las anteriores, dónde notaba todo mucho más fluido. Es cierto que ha sido útil ya que hemos aprendido a programar un altavoz y a configurar nuestra placa como si fuera un piano, pero el entorno de trabajo no me ha gustado demasiado pues me ha dado algunos problemas algo tontos. Si tengo que elegir me quedo el STM32CubeIDE.