***Rapid Embedded Systems***

***Design and Programming Course***

**LAB 4**

**Timer and Pulse-Width-Modulation**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc35610125)

[1.1 Lab overview 1](#_Toc35610126)

[2 Requirements 1](#_Toc35610127)

[2.1 Software and Hardware 1](#_Toc35610128)

[3 Hardware Setup 2](#_Toc35610129)

[3.1 Pin Layout 2](#_Toc35610130)

[4 Software interfaces 3](#_Toc35610131)

[4.1 Timer Interface 3](#_Toc35610132)

[4.2 Ticker interface 3](#_Toc35610133)

[4.3 Software functions 4](#_Toc35610134)

[5 Designing an Audio Player 5](#_Toc35610135)

[5.1 Define the Music Note and Beat Length 5](#_Toc35610136)

[5.2 Application code 6](#_Toc35610137)

[6 Additional references 6](#_Toc35610138)

# Introduction

## Lab overview

In this lab, we will design an audio player that will play a melody saved beforehand. The volume of the music as well as its speed can be controlled by two buttons.

By the end of this lab you will get some insight and practical experience with the Mbed API for Timers and Tickers.

# Requirements

## Software and Hardware

In this lab, we will be using the following hardware and software:

* **Mbed Studio**, an IDE designed to streamline development and prototyping using Mbed enabled microcontrollers and development boards

**OR**

**Mbed online compiler**, which can be found here: <https://ide.mbed.com/compiler>

* **NUCLEO-F401RE**, or another suitable Mbed OS 5 compatible development board. A full list of compatible devices can be found here: <https://os.mbed.com/platforms>, note that outputs may have to be reconfigured for devices following a different standard.
* A breadboard, 2 x 100Ω resistors, 2 x 10kΩ potentiometers, 1 x speaker (ABI-001-RC), 3 x RGB LED.

The code skeleton, which includes some guidelines for implementing the task in 5.1 and 5.2, should be found in the same folder as this manual

# Hardware Setup

## Pin Layout

In this experiment, we are going to use the Nucleo F401RE Board, for which pin descriptions can be found below:



Figure 1: The NUCLEO F401RE board pin descriptions

On a breadboard, connect the 2 buttons according to the diagram on the left.  Your buttons should pull the GPIO pin low.

|  |  |
| --- | --- |
| **Pin** | **Pin name in Mbed API**  |
| Button 1 | D4 |
| Button 2 | D7 |
| PWM speaker | D3 |
| Red LED | D5 |
| Green LED  | D6 |
| Blue LED | D9 |

Then connect the buttons, the RGB LEDs and the speaker to their respective pin on the board as defined in the table below.

# Software interfaces

## Timer Interface

We can use the timer interface to create, start, stop and read a timer for measuring precise times (better than millisecond precision), for example:

Timer t;

int main() {

 t.start();

 printf("Hello World!\n");

 t.stop();

 printf("The time taken was %f seconds\n", t.read());

}

You can independently create, start and stop any number of Timer objects simultaneously. The member functions of the API can be found in 4.3.

## Ticker interface

We can use the Ticker interface to set up a recurring interrupt; it calls a function repeatedly and at a specified rate, for example

Ticker flipper;

DigitalOut led1(LED1);

DigitalOut led2(LED2);

void flip() {

 led2 = !led2;

}

int main() {

 led2 = 1;

 flipper.attach(&flip, 2.0); // the address of the function to be attached (flip) and the interval (2 seconds)

 // spin in a main loop. flipper will interrupt it to call flip

 while(1) {

 led1 = !led1;

 wait(0.2);

 }

}

You can create any number of Ticker objects, allowing multiple outstanding interrupts at the same time. The function can be a static function, a member function of a particular object or a Callback object. This can be done using the member functions of this API which can be found in 4.3.

## Software functions

You can find in the table below the member function of the Timer Interface and Ticker interface which were introduced in this lab, as well as the some member function of the PwmOut Interface which will be helpful for this lab.

|  |  |
| --- | --- |
| **Function name** | **Description**  |
| ***Timer functions*** |
| **void start ()** | Start the timer |
| **void stop ()** | Stop the timer |
| **void reset ()** | Reset the timer to 0 |
| **float read ()** | Get the time passed in seconds |
| **int read\_ms ()** | Get the time passed in milliseconds |
| **int read\_us ()** | Get the time passed in microseconds |
| ***Time Ticker functions*** |
| **void attach (void(\*fptr)(void), float t)** | Attach a function to be called by the Ticker, specifying the interval in seconds |
| **void attach (T \*tptr, void(T::\*mptr)(void), float t)** | Attach a member function to be called by the Ticker, specifying the interval in seconds |
| **void attach\_us (void(\*fptr) (void), unsigned int t)** | Attach a function to be called by the Ticker, specifying the interval in micro-seconds |
| **void attach\_us (T \*tptr, void (T::\*mptr) (void), unsigned int t)** | Attach a member function to be called by the Ticker, specifying the interval in micro-seconds |
| **void detach ()** | Detach the function.  |
| **static void irq (uint32\_t id)** | The handler registered with the underlying timer interrupt. |
| ***PWM output functions*** |
| **Void write (float value)** | Set the output duty-cycle, specified as a percentage (float)  |
| **float read ()** | Return the current output duty-cycle setting, measured as a percentage (float)  |
| **void period (float seconds)** | Set the PWM period, specified in seconds (float), keeping the duty cycle the same |
| **void period\_ms (int ms)** | Set the PWM period, specified in milli-seconds (int), keeping the duty cycle the same |
| **void period\_us (int us)** | Set the PWM period, specified in micro-seconds (int), keeping the duty cycle the same |
| **void pulsewidth (float seconds)** | Set the PWM pulse width, specified in seconds (float), keeping the period the same |
| **void pulsewidth\_ms (int ms)** | Set the PWM pulse width, specified in milli-seconds (int), keeping the period the same |
| **void pulsewidth\_us (int us)** | Set the PWM pulse width, specified in micro-seconds (int), keeping the period the same |
| **PwmOut & operator= (float value)** | An operator shorthand for write ()  |
| **PwmOut & operator float ()** | An operator shorthand for read ()  |

# Designing an Audio Player

In this lab task, you will design an audio player that uses a timer, PWM, and interrupts.

## Define the Music Note and Beat Length

To play music, it is better to define a list of notes that maybe used to play the music, including the music notes (the frequency of the sound, e.g. do, re, mi, fa, so, la, si) and the beat length (the time that the current note lasts for, such as whole note, half note). To do that, you need to adjust the period of the PWM so that it can produce a correct frequency for each music note, and the period of play to produce the desired beat.

After defining the music note and beat length, you can then make a note array which will produce a simple piece of music, for example the following sheet gives the notes and beats of “Jingle bells”





In this first task, on the skeleton code, you should:

* Define a basic musique note, for example:

# define Do 0.5

# define Re 0.45

# define Mi 0.4

# define Fa 0.36

# define So 0.33

# define La 0.31

# define Si 0.3

# define No 0

* Define a basic beat note, for example:

# define b1 0.5 //whole note

# define b2 0.25 //half note

…

* Define a basic beat note, for example:

float note []={ Mi, No, Mi …

float beat [] ={ b3, b3, b3 …

…

## Application code

In this task, you will be completing the code skeleton using the functions presented in 4.3. You can make use of the following guidelines:

* Define the digital inputs (two buttons) and PWM outputs (speaker and LEDs)
* Define the time ticker
* Write the time ticker ISR, which will be periodically triggered after each single music note
	+ Update the PWM frequency to the next music note
	+ Update the beat length for the next music note (reconfigure the tick interrupt time)
	+ Update the brightness of the LEDs to reflect the melody change
		- The red LED intensity should represent the note being played.
		- The green LED intensity should represent the beat’s length.
		- The blue LED intensity should represent the volume. It also should blink at the same speed of the melody.
	+ The input of the first button will be used to adjust the volume to 80% when it is pressed and 20% when it is not,
	+ The input of the second button will be used to adjust the speed of the song by adding a delay of 0.5 seconds when the button is pressed.
* In the main program
	+ Initialize the time ticker
	+ Sleep and wait for interrupts

# Additional references

**Documentation of Timer API**

<https://os.mbed.com/docs/mbed-os/v5.13/apis/timer.html>

**Documentation of Ticker API**

<https://os.mbed.com/docs/mbed-os/v5.13/apis/ticker.html>

**Documentation of PwmOut API**

<https://os.mbed.com/docs/mbed-os/v5.13/apis/pwmout.html>