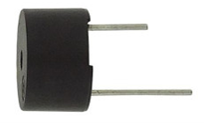
**Buzzer**

**J. Lemaire**

**(Pierrefeu August 2021)**

# Introduction

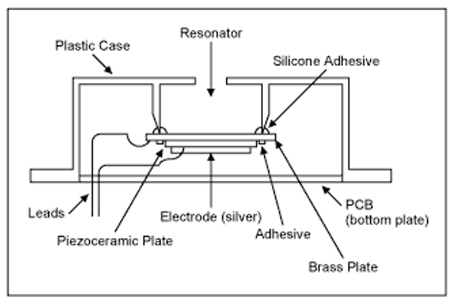
Considering a piezoelectric buzzer :



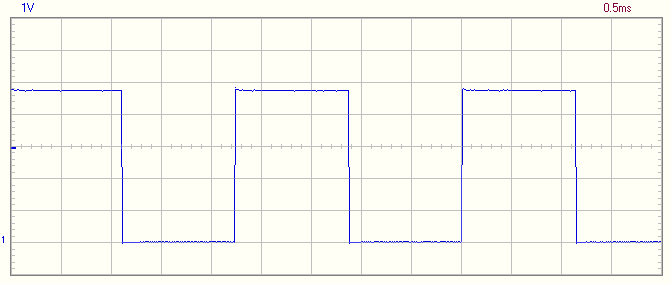
i tried to understand how it produces sounds and to improve the **tone** function, using an Arduino Uno and its **Timer2** interrupts**.**

# Piezoelectric buzzer

This electronic component produces sounds when powered by a rectangular periodic signal[[1]](#footnote-2) (inversion of the piezoelectricity principle, discovered by Jacques and Pierre Curie, back in 1880[[2]](#footnote-3))



The **note** of the sound depends on the frequency of the signal ; for instance :



**A4** (freq = **440**Hz => period ≅ 2,27ms)[[3]](#footnote-4)

The frequencies of the other notes are obtained from A4, applying a constant factor[[4]](#footnote-5) :

where , the note index, is a positive or negative integer and

Hence the frequency is multiplied or divided by 2 on stepping 12 successive notes (an octave).

In the European music theory MIDI notation, the notes are named and grouped in octaves, from Cx to Bx :

|  |  |  |
| --- | --- | --- |
| **Note** | **Index** | **Freq (Hz)** |
| … | … |  |
| C4 | -9 | 261,63 |
| C#4 | -8 | 277,18 |
| D4 | -7 | 293,66 |
| D#4 | -6 | 311,13 |
| E4 | -5 | 329,63 |
| F4 | -4 | 349,23 |
| F#4 | -3 | 369,99 |
| G4 | -2 | 392,00 |
| G#4 | -1 | 415,30 |
| A4 | **0** | **440,00** |
| A#4 | 1 | 466,16 |
| B4 | 2 | 493,88 |
| C5 | 3 | 523,25 |
| C#5 | 4 | 554,37 |
| D5 | 5 | 587,33 |
| D#5 | 6 | 622,25 |
| E5 | 7 | 659,26 |
| F5 | 8 | 698,46 |
| F#5 | 9 | 739,99 |
| G5 | 10 | 783,99 |
| G#5 | 11 | 830,61 |
| A5 | 12 | 880,00 |
| A#5 | 13 | 932,33 |
| B5 | 14 | 987,77 |
| … | … |  |

In the following sketches, the note frequencies (rounded to integers) will be defined in the **Pitches.h** file :

…

#define NOTE\_C4 262

#define NOTE\_CS4 277

#define NOTE\_D4 294

#define NOTE\_DS4 311

#define NOTE\_E4 330

#define NOTE\_F4 349

#define NOTE\_FS4 370

#define NOTE\_G4 392

#define NOTE\_GS4 415

#define NOTE\_A4 440

#define NOTE\_AS4 466

#define NOTE\_B4 494

#define NOTE\_C5 523

#define NOTE\_CS5 554

#define NOTE\_D5 587

#define NOTE\_DS5 622

#define NOTE\_E5 659

#define NOTE\_F5 698

#define NOTE\_FS5 740

#define NOTE\_G5 784

#define NOTE\_GS5 831

#define NOTE\_A5 880

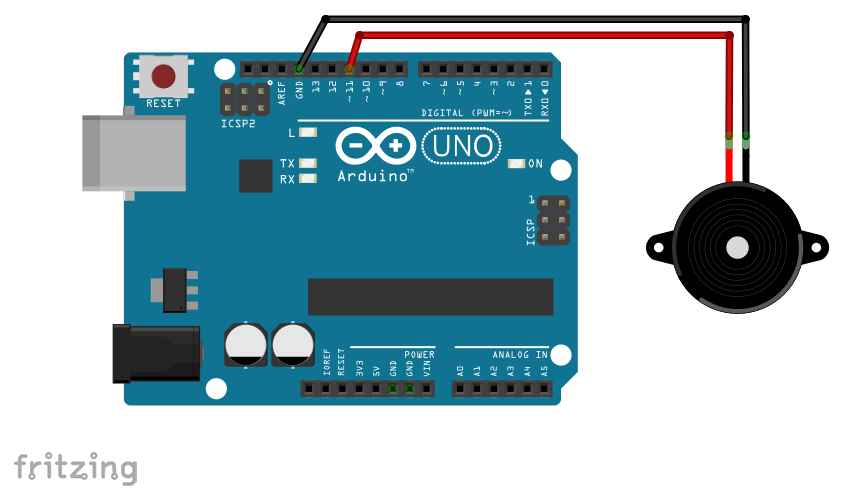
#define NOTE\_AS5 932

#define NOTE\_B5 988

…

# Simple Arduino sketches to play a melody

We consider such a buzzer, connected on the pins **11** and **GND** of an Arduino Uno :



To play a **note**, the Arduino cores offers a simple function, **tone[[5]](#footnote-6) [[6]](#footnote-7)**, with the following parameters :

* pin
* frequency (Hz)
* duration (ms)

Hence, to play the A4 note during 2s, just call :

tone**(**8**,** 440**,** 2000**)**

Let us consider now a melody, that is a succession of **notes**[[7]](#footnote-8), with an indication of their **durations**[[8]](#footnote-9) :



// TestBuzzer1.ino

// Adapted from https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody

#include "Pitches.h"

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

Note melody**[]** **=** **{{**NOTE\_C4**,**250**},{**NOTE\_G3**,**125**},{**NOTE\_G3**,**125**},{**NOTE\_A3**,**250**},{**NOTE\_G3**,**250**},{**0**,**250**},** **{**NOTE\_B3**,**250**},{**NOTE\_C4**,**250**}};**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

**for** **(**int i**=**0**;** i**<**nbPlay**;** i**++)**

**{**

**for** **(**int j**=**0**;** j**<**nbNote**;** j**++)**

**{**

tone**(**11**,** tNote**[**j**].**freq**,** tNote**[**j**].**len**);**

delay**(**tNote**[**j**].**len**+**50**);** // Blocking !!

**}**

**if** **(**i **<** nbPlay**-**1**)** delay**(**200**);**

**}**

**}**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

delay**(**100**);**

unsigned long tm **=** micros**();**

play**(**melody**,** **sizeof(**melody**)/**4**,** 2**);**

Serial**.**print**(**"test1 : "**);** Serial**.**print**(**micros**()-**tm**);** Serial**.**println**(**" us"**);**

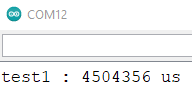
**}**

void loop**()**

**{**

**}**

Obviously, this sketch is blocking because of the delays used here to separate the notes :



These blocking delays can be drastically reduced, using an **elapsedMillis[[9]](#footnote-10)** timer :

// TestBuzzer2.ino

// Asynchonous tone

#include "Pitches.h"

#include <elapsedMillis.h>

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

Note melody**[]** **=** **{{**NOTE\_C4**,**250**},{**NOTE\_G3**,**125**},{**NOTE\_G3**,**125**},{**NOTE\_A3**,**250**},{**NOTE\_G3**,**250**},{**0**,**250**},** **{**NOTE\_B3**,**250**},{**NOTE\_C4**,**250**}};**

elapsedMillis \_timer**;**

unsigned int \_tick**;**

Note**\*** \_tNote**;**

unsigned int \_idNote**,** \_nbNote**,** \_idPlay**,** \_nbPlay**;**

bool okUpdate**,** okPlay **=** **false;**

unsigned long tm**,** delta**;**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

\_tick **=** 0**;**

\_tNote **=** tNote**;**

\_idNote **=** 0**;**

\_nbNote **=** nbNote**;**

\_idPlay **=** 0**;**

\_nbPlay **=** nbPlay**;**

okPlay **=** **false;**

**}**

void updatePlay**()**

**{**

**if** **(**\_timer **>=** \_tick **&&** \_idPlay **<** \_nbPlay**)**

**{**

\_timer **=** 0**;**

tone**(**11**,** \_tNote**[**\_idNote**].**freq**,** \_tNote**[**\_idNote**].**len**);**

\_tick **=** \_tNote**[**\_idNote**].**len **+** 50**;**

\_idNote**++;**

**if** **(**\_idNote **==** \_nbNote**)**

**{**

\_idNote **=** 0**;**

\_idPlay**++;**

**if** **(**\_idPlay **<** \_nbPlay**)** \_tick **+=** 200**;**

**else** okPlay **=** **true;**

**}**

okUpdate **=** **true;**

**}**

**else** okUpdate **=** **false;**

**}**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

delay**(**100**);**

delta **=** 0**;**

tm **=** micros**();**

play**(**melody**,** **sizeof(**melody**)/**4**,** 2**);**

delta **+=** micros**()-**tm**;**

**}**

void loop**()**

**{**

tm **=** micros**();**

updatePlay**();**

**if** **(**okUpdate**)** delta **+=** micros**()-**tm**;**

**if** **(**okPlay**)**

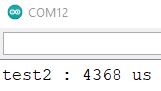
**{**

Serial**.**print**(**"test2 : "**);** Serial**.**print**(**delta**);** Serial**.**println**(**" us"**);**

okPlay **=** **false;**

**}**

**}**



However, even if the implementation of the **tone** function by **Brett Hagman** is non blocking[[10]](#footnote-11) because it uses timer interrupts, these delays can be reduced again using the more compact code of the **NewTone** library from **Tim Eckel**[[11]](#footnote-12):

// TestBuzzer3.ino

// Asynchonous tone using NewTone from Tim Eckel

#include "Pitches.h"

#include <elapsedMillis.h>

#include "NewTone.h"

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

Note melody**[]** **=** **{{**NOTE\_C4**,**250**},{**NOTE\_G3**,**125**},{**NOTE\_G3**,**125**},{**NOTE\_A3**,**250**},{**NOTE\_G3**,**250**},{**0**,**250**},** **{**NOTE\_B3**,**250**},{**NOTE\_C4**,**250**}};**

elapsedMillis \_timer**;**

unsigned int \_tick**;**

Note**\*** \_tNote**;**

unsigned int \_idNote**,** \_nbNote**,** \_idPlay**,** \_nbPlay**;**

bool okUpdate**,** okPlay **=** **false;**

unsigned long tm**,** delta**;**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

\_tick **=** 0**;**

\_tNote **=** tNote**;**

\_idNote **=** 0**;**

\_nbNote **=** nbNote**;**

\_idPlay **=** 0**;**

\_nbPlay **=** nbPlay**;**

okPlay **=** **false;**

**}**

void updatePlay**()**

**{**

**if** **(**\_timer **>=** \_tick **&&** \_idPlay **<** \_nbPlay**)**

**{**

\_timer **=** 0**;**

NewTone**(**11**,** \_tNote**[**\_idNote**].**freq**,** \_tNote**[**\_idNote**].**len**);**

\_tick **=** \_tNote**[**\_idNote**].**len **+** 50**;**

\_idNote**++;**

**if** **(**\_idNote **==** \_nbNote**)**

**{**

\_idNote **=** 0**;**

\_idPlay**++;**

**if** **(**\_idPlay **<** \_nbPlay**)** \_tick **+=** 200**;**

**else** okPlay **=** **true;**

**}**

okUpdate **=** **true;**

**}**

**else** okUpdate **=** **false;**

**}**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

delay**(**100**);**

delta **=** 0**;**

tm **=** micros**();**

play**(**melody**,** **sizeof(**melody**)/**4**,** 2**);**

delta **+=** micros**()-**tm**;**

**}**

void loop**()**

**{**

tm **=** micros**();**

updatePlay**();**

**if** **(**okUpdate**)** delta **+=** micros**()-**tm**;**

**if** **(**okPlay**)**

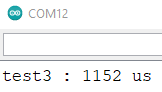
**{**

Serial**.**print**(**"test3 : "**);** Serial**.**print**(**delta**);** Serial**.**println**(**" us"**);**

okPlay **=** **false;**

**}**

**}**



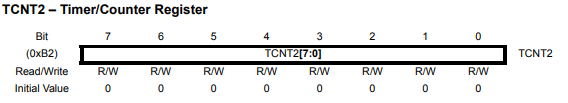
But, this library which uses **timer1** interrupts cannot be called when this timer is already reserved by other components such as servos. And because **timer0** is reserved by the **millis()**, **micros()** and **delay()** functions, it’s a interesting, with such Arduino, to replace **timer1** by **timer2** in the **NewTone** library. It would be a good place to understand how this timer works and to optimize the code.

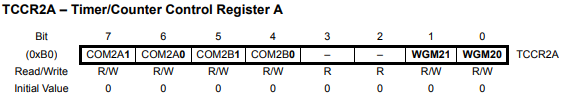
# A sound library using timer2

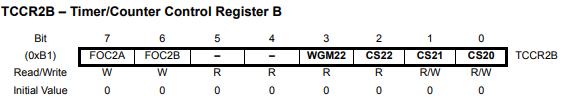
Here, the main difficulty consists in understanding how **timer2** works and can be programmed using its registers, because the datasheet of the **ATMega328P**[[12]](#footnote-13) is a bit obscur… Fortunately there is a very clear article on this subject, from **Ken Shiriff** with further editing by **Paul Badger**, **Secrets of Arduino PWM**[[13]](#footnote-14), which is highly recommended read !

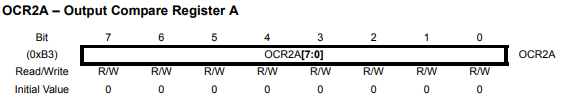
Let us begin with the needed characteristics of **timer2**.

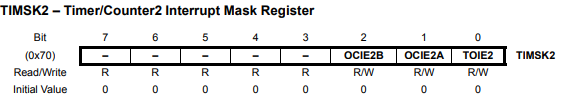
* Registers :











* **Timer2** is a **8 bits** timer, which means that it can count from 0 to 255 at maximum, or inversely from 255 at maximum to 0. The count values are stored in the **TCNT2** register.
* Hence, 2 types of cycles are proposed :
  + **Short cycles** with 256 steps at maximum (0=>1, 1=>2, …, 254=>255, 255=>0)
  + **Long cycles** with 510 steps at maximum (0=>1, 1=>2, …, 254=>255, 255=>254, 254=>253, …, 1=>0) :
* The CPU clock drives this timer with a frequency (**F\_CPU)** of **16MHz** by default. Hence an a step takes **0.0625 µs** and a maximum short cycle takes 256 x 0.0625 = **16µs**.
* To reduce the frequency (or enlarge the period) of a step, a **prescaler** can be used with a possible factor equal to 1, 8, 32, 64, 128, 256 or 1024, according to the 3 bits of **CS2** :

|  |  |  |  |
| --- | --- | --- | --- |
| **CS2** | **Prescaler** | **Freq step (kHz)** | **Period step (µs)** |
| 001 | 1 | 16000 | 0,0625 |
| 010 | 8 | 2000 | 0,5 |
| 011 | 32 | 500 | 2 |
| 100 | 64 | 250 | 4 |
| 101 | 128 | 125 | 8 |
| 110 | 256 | 62,5 | 16 |
| 111 | 1024 | 15,625 | 64 |

* And to adjust the frequency of a cycle, its top value (255 at maximum hence) can be changed, using the **OCR2A** register ; a value of N will lead to :
  + **Short cycles** with N+1 steps (0=>1, 1=>2, …, N-1=>N, N=>0), with a frequency of 16000000/(prescaler x (N+1)) Hz :
  + **Long cycles** with 2N steps (0=>1, 1=>2, …, N-1=>N, N=>N-1, N-2=>N-3, …, 1=>0), with a frequency of 16000000/(prescaler x 2N) Hz :
* The **timer2** can generate **PWM** signals on pins 11 (**OC2A**) and 3 (**OC2B**) : their form is precised by the 2 bits of **COM2A** and the 2 bits of **COM2B**, of the control register **TCCR2A**. Just observe that **00** disables the PWM output and that **01** enables a PWM output for the pin 11, with a **50% duty cycle** when the bit **WMG22** of the **TCCR2B** is set.
  + For the **short cycles**, the signal is called **Fast PWM** :
  + For the **long cycles**, the signal is called **PWM phase-correct** :

Observe that the PWM frequency is always the half of the cycle frequency. Hence, for the long cycle case, the **PWM phase-correct frequency** is :

**freqS = 16000000/prescaler/4/N Hz**

* The other bits of **WGM2** precise the type of cycle :

|  |  |  |
| --- | --- | --- |
| **WGM2** | **Cycle** | **OC2A update at** |
| 101 | Long | TOP |
| 111 | Short | 0x00 |

* Otherwise, different interrupts can be fired by the **timer2**, and specially the **compare A interrupts** when the **TCNT2 = OCR2A**. Their frequency is equal to the cycle frequency and they are enabled on setting the **OCIE2A** bit of the **TIMSK2** register.

***In conclusion, to disable the PWM signal on pin 3, enable a F frequency PWM signal on pin 11 with 50% duty cycle and fire compare A interrupts, we have to :***

* Select a **prescaler** setting **CS2** such that **N** = **16000000/prescaler/4/F** become <256
* **OCR2A** = **N**
* **TCCR2A** = (1<<**COM2A0**) | (1<<**WGM21**)
* **TCCR2B** = (1<<**WGM22**) | **CS2**
* **TIMSK2** = (1<<**OCIE2A**)

***To disable the PWM signals and the compare A interrupts :***

* **TCCR2A** = 0
* **TIMSK2** = 0

In the following sketch, the function **ISR(TIMER2\_COMPA\_VECT)** implements the interrupts treatment which only consists in stopping the tone process after the required duration.

// TestBuzzer4.ino

// Asynchonous tone on pin 11 using Timer2 and PWM phase-correct with top in OCRA

#include "Pitches.h"

#include <elapsedMillis.h>

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

Note melody**[]** **=** **{{**NOTE\_C4**,**250**},{**NOTE\_G3**,**125**},{**NOTE\_G3**,**125**},{**NOTE\_A3**,**250**},{**NOTE\_G3**,**250**},{**0**,**250**},** **{**NOTE\_B3**,**250**},{**NOTE\_C4**,**250**}};**

elapsedMillis \_timer**;**

unsigned int \_tick**;**

Note**\*** \_tNote**;**

unsigned int \_idNote**,** \_nbNote**,** \_idPlay**,** \_nbPlay**;**

unsigned long endTime**;**

bool okUpdate**,** okPlay **=** **false;**

unsigned long tm**,** delta**;**

void tone**(**unsigned int freq**,** unsigned int len **=** 0**)**

**{**

// End time

**if** **(**len **>** 0**)** endTime **=** len **+** millis**();**

**else** endTime **=** 0xFFFFFFFF**;**

**if** **(**freq **>** 0**)**

**{**

// CS2 and OCR2A

unsigned CS2**;**

unsigned int fact**;**

**if** **(**freq **<** 60**)** **{**CS2 **=** 7**;** fact **=** 3906**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 1024

**else** **if** **(**freq **<** 120**)** **{**CS2 **=** 6**;** fact **=** 15625**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 256

**else** **if** **(**freq **<** 240**)** **{**CS2 **=** 5**;** fact **=** 31250**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 128

**else** **if** **(**freq **<** 480**)** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 64

**else** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** round**(**fact**/**freq**);}** // Prescaler = 64

// Enable PWM phase-correct on pin 11,with the previous prescaler, top in OCR2A and 50% duty cycle

TCCR2A **=** **(**1**<<**COM2A0**)** **|** **(**1**<<**WGM20**);**

TCCR2B **=** **(**1**<<**WGM22**)** **|** CS2**;**

// Enable COMPA interrupt

TIMSK2 **=** **(**1**<<**OCIE2A**);**

**}**

**}**

void myNoTone**()**

**{**

TCCR2A **=** 0**;** // Disable PWM on pin 11

TIMSK2 **=** 0**;** // Disable the timer interrupt.

**}**

ISR**(**TIMER2\_COMPA\_vect**)**

**{**

**if** **(**millis**()** **>=** endTime**)** myNoTone**();** // Stop tone

**}**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

\_tick **=** 0**;**

\_tNote **=** tNote**;**

\_idNote **=** 0**;**

\_nbNote **=** nbNote**;**

\_idPlay **=** 0**;**

\_nbPlay **=** nbPlay**;**

okPlay **=** **false;**

**}**

void updatePlay**()**

**{**

**if** **(**\_timer **>=** \_tick **&&** \_idPlay **<** \_nbPlay**)**

**{**

\_timer **=** 0**;**

tone**(**\_tNote**[**\_idNote**].**freq**,** \_tNote**[**\_idNote**].**len**);**

\_tick **=** \_tNote**[**\_idNote**].**len **+** 50**;**

\_idNote**++;**

**if** **(**\_idNote **==** \_nbNote**)**

**{**

\_idNote **=** 0**;**

\_idPlay**++;**

**if** **(**\_idPlay **<** \_nbPlay**)** \_tick **+=** 200**;**

**else** okPlay **=** **true;**

**}**

okUpdate **=** **true;**

**}**

**else** okUpdate **=** **false;**

**}**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

delay**(**100**);**

delta **=** 0**;**

pinMode**(**11**,** OUTPUT**);**

tm **=** micros**();**

play**(**melody**,** **sizeof(**melody**)/**4**,** 2**);**

delta **+=** micros**()-**tm**;**

**}**

void loop**()**

**{**

tm **=** micros**();**

updatePlay**();**

**if** **(**okUpdate**)** delta **+=** micros**()-**tm**;**

**if** **(**okPlay**)**

**{**

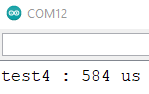
Serial**.**print**(**"test4 : "**);** Serial**.**print**(**delta**);** Serial**.**println**(**" us"**);**

okPlay **=** **false;**

**}**

**}**

Results :

****

**Remarks**

* The reduction of the delays, relatively to the **NewTone** library, is obtained on using **unsigned int** (an not **unsigned long**) to evaluate the **N** value of **OCR2A** with the following factor :

**fact = 16000000/prescaler/4**

Hence, the lowest acceptable prescaler is 64 because **fact** need then to be < 65536 and here, the **PWM phase-correct** option is better than the **Fast PWM** option, for which **fact** should be practically twiced.

* To obtain the best approximation **freqS** of the note frequency, **freq**, the lowest possible prescaler is searched so that :

**N = fact/freq < 256**

The following Excel table shows these values and the errors (%) on the note frequencies :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **note** | **freq (Hz)** | **CS2** | **prescal** | **fact** | **N** | **freqS (Hz)** | **error (%)** |
| C1 | 33 | 7 | 1024 | 3906 | 118 | 33 | 0,0 |
| C#1 | 35 | 7 | 1024 | 3906 | 111 | 35 | 0,0 |
| D1 | 37 | 7 | 1024 | 3906 | 105 | 37 | 0,0 |
| D#1 | 39 | 7 | 1024 | 3906 | 100 | 39 | 0,0 |
| E1 | 41 | 7 | 1024 | 3906 | 95 | 41 | 0,0 |
| F1 | 44 | 7 | 1024 | 3906 | 88 | 44 | 0,0 |
| F#1 | 46 | 7 | 1024 | 3906 | 84 | 46 | 0,0 |
| G1 | 49 | 7 | 1024 | 3906 | 79 | 49 | 0,0 |
| G#1 | 52 | 7 | 1024 | 3906 | 75 | 52 | 0,0 |
| A1 | 55 | 7 | 1024 | 3906 | 71 | 55 | 0,0 |
| Bb1 | 58 | 7 | 1024 | 3906 | 67 | 58 | 0,0 |
| B1 | 62 | 6 | 256 | 15625 | 252 | 62 | 0,0 |
| C2 | 65 | 6 | 256 | 15625 | 240 | 65 | 0,0 |
| C#2 | 69 | 6 | 256 | 15625 | 226 | 69 | 0,0 |
| D2 | 73 | 6 | 256 | 15625 | 214 | 73 | 0,0 |
| D#2 | 78 | 6 | 256 | 15625 | 200 | 78 | 0,0 |
| E2 | 82 | 6 | 256 | 15625 | 190 | 82 | 0,0 |
| F2 | 87 | 6 | 256 | 15625 | 179 | 87 | 0,0 |
| F#2 | 92 | 6 | 256 | 15625 | 169 | 92 | 0,0 |
| G2 | 98 | 6 | 256 | 15625 | 159 | 98 | 0,0 |
| G#2 | 104 | 6 | 256 | 15625 | 150 | 104 | 0,0 |
| A2 | 110 | 6 | 256 | 15625 | 142 | 110 | 0,0 |
| Bb2 | 117 | 6 | 256 | 15625 | 133 | 117 | 0,0 |
| B2 | 123 | 5 | 128 | 31250 | 254 | 123 | 0,0 |
| C3 | 131 | 5 | 128 | 31250 | 238 | 131 | 0,0 |
| C#3 | 139 | 5 | 128 | 31250 | 224 | 139 | 0,0 |
| D3 | 147 | 5 | 128 | 31250 | 212 | 147 | 0,0 |
| D#3 | 156 | 5 | 128 | 31250 | 200 | 156 | 0,0 |
| E3 | 165 | 5 | 128 | 31250 | 189 | 165 | 0,0 |
| F3 | 175 | 5 | 128 | 31250 | 178 | 175 | 0,0 |
| F#3 | 185 | 5 | 128 | 31250 | 168 | 186 | 0,5 |
| G3 | 196 | 5 | 128 | 31250 | 159 | 196 | 0,0 |
| G#3 | 208 | 5 | 128 | 31250 | 150 | 208 | 0,0 |
| A3 | 220 | 5 | 128 | 31250 | 142 | 220 | 0,0 |
| Bb3 | 233 | 5 | 128 | 31250 | 134 | 233 | 0,0 |
| B3 | 247 | 4 | 64 | 62500 | 253 | 247 | 0,0 |
| C4 | 262 | 4 | 64 | 62500 | 238 | 262 | 0,0 |
| C#4 | 277 | 4 | 64 | 62500 | 225 | 277 | 0,0 |
| D4 | 294 | 4 | 64 | 62500 | 212 | 294 | 0,0 |
| D#4 | 311 | 4 | 64 | 62500 | 200 | 312 | 0,3 |
| E4 | 330 | 4 | 64 | 62500 | 189 | 330 | 0,0 |
| F4 | 349 | 4 | 64 | 62500 | 179 | 349 | 0,0 |
| F#4 | 370 | 4 | 64 | 62500 | 168 | 372 | 0,5 |
| G4 | 392 | 4 | 64 | 62500 | 159 | 393 | 0,3 |
| G#4 | 415 | 4 | 64 | 62500 | 150 | 416 | 0,2 |
| A4 | 440 | 4 | 64 | 62500 | 142 | 440 | 0,0 |
| Bb4 | 466 | 4 | 64 | 62500 | 134 | 466 | 0,0 |
| B4 | 494 | 4 | 64 | 62500 | 127 | 492 | 0,4 |
| C5 | 523 | 4 | 64 | 62500 | 120 | 520 | 0,6 |
| C#5 | 554 | 4 | 64 | 62500 | 113 | 553 | 0,2 |
| D5 | 587 | 4 | 64 | 62500 | 106 | 589 | 0,3 |
| D#5 | 622 | 4 | 64 | 62500 | 100 | 625 | 0,5 |
| E5 | 659 | 4 | 64 | 62500 | 95 | 657 | 0,3 |
| F5 | 698 | 4 | 64 | 62500 | 90 | 694 | 0,6 |
| F#5 | 740 | 4 | 64 | 62500 | 84 | 744 | 0,5 |
| G5 | 784 | 4 | 64 | 62500 | 80 | 781 | 0,4 |
| G#5 | 831 | 4 | 64 | 62500 | 75 | 833 | 0,2 |
| A5 | 880 | 4 | 64 | 62500 | 71 | 880 | 0,0 |
| Bb5 | 932 | 4 | 64 | 62500 | 67 | 932 | 0,0 |
| B5 | 988 | 4 | 64 | 62500 | 63 | 992 | 0,4 |
| C6 | 1047 | 4 | 64 | 62500 | 60 | 1041 | 0,6 |
| C#6 | 1109 | 4 | 64 | 62500 | 56 | 1116 | 0,6 |
| D6 | 1175 | 4 | 64 | 62500 | 53 | 1179 | 0,3 |
| D#6 | 1245 | 4 | 64 | 62500 | 50 | 1250 | 0,4 |
| E6 | 1319 | 4 | 64 | 62500 | 47 | 1329 | 0,8 |
| F6 | 1397 | 4 | 64 | 62500 | 45 | 1388 | 0,6 |
| F#6 | 1480 | 4 | 64 | 62500 | 42 | 1488 | 0,5 |
| G6 | 1568 | 4 | 64 | 62500 | 40 | 1562 | 0,4 |
| G#6 | 1661 | 4 | 64 | 62500 | 38 | 1644 | 1,0 |
| A6 | 1760 | 4 | 64 | 62500 | 36 | 1736 | 1,4 |
| Bb6 | 1865 | 4 | 64 | 62500 | 34 | 1838 | 1,4 |
| B6 | 1976 | 4 | 64 | 62500 | 32 | 1953 | 1,2 |
| C7 | 2093 | 4 | 64 | 62500 | 30 | 2083 | 0,5 |
| C#7 | 2217 | 4 | 64 | 62500 | 28 | 2232 | 0,7 |
| D7 | 2349 | 4 | 64 | 62500 | 27 | 2314 | 1,5 |
| D#7 | 2489 | 4 | 64 | 62500 | 25 | 2500 | 0,4 |
| E7 | 2637 | 4 | 64 | 62500 | 24 | 2604 | 1,3 |
| F7 | 2794 | 4 | 64 | 62500 | 22 | 2840 | 1,6 |
| F#7 | 2960 | 4 | 64 | 62500 | 21 | 2976 | 0,5 |
| G7 | 3136 | 4 | 64 | 62500 | 20 | 3125 | 0,4 |
| G#7 | 3322 | 4 | 64 | 62500 | 19 | 3289 | 1,0 |
| A7 | 3520 | 4 | 64 | 62500 | 18 | 3472 | 1,4 |
| Bb7 | 3729 | 4 | 64 | 62500 | 17 | 3676 | 1,4 |
| B7 | 3951 | 4 | 64 | 62500 | 16 | 3906 | 1,1 |
| C8 | 4186 | 4 | 64 | 62500 | 15 | 4166 | 0,5 |
| C#8 | 4435 | 4 | 64 | 62500 | 14 | 4464 | 0,7 |
| D8 | 4699 | 4 | 64 | 62500 | 13 | 4807 | 2,3 |
| D#8 | 4978 | 4 | 64 | 62500 | 13 | 4807 | 3,4 |
| E8 | 5274 | 4 | 64 | 62500 | 12 | 5208 | 1,3 |
| F8 | 5588 | 4 | 64 | 62500 | 11 | 5681 | 1,7 |
| F#8 | 5920 | 4 | 64 | 62500 | 11 | 5681 | 4,0 |
| G8 | 6272 | 4 | 64 | 62500 | 10 | 6250 | 0,4 |
| G#8 | 6645 | 4 | 64 | 62500 | 9 | 6944 | 4,5 |
| A8 | 7040 | 4 | 64 | 62500 | 9 | 6944 | 1,4 |
| Bb8 | 7459 | 4 | 64 | 62500 | 8 | 7812 | 4,7 |
| B8 | 7902 | 4 | 64 | 62500 | 8 | 7812 | 1,1 |

* Using these choices, the approximations are very good for the low frequencies (< 480 Hz). After, we empirically reduced the errors on using the formula :

**N = round(fact/freq)[[14]](#footnote-15)**

But, considering the very poor sonic quality of the buzzer, the difference isn’t crucial !

* In view to simplify the code and I opted to take advantage of a PWM signal generated by **timer2** and in consequence to limit the connection of the buzzer to the pin **11**. If this constraint isn’t acceptable, the solution consists in toggling the pin value in the ISR, as proposed in the **Tone** and the **NewTone** libraries : with our structure of code, the modifications are obvious and the results are very similar, but just a little worse.
* Obviously, the **Compare A interrupts** can be avoided and the test to stop the tone process performed using a tick in the loop function. But it’s simpler to use such an interrupt in view to encapsulate all the tone process in a class.
* Inversely, the **Compare A interrupts** can be used to encapsulate completely the **updatePlay** treatment, as proposed in the following **Buzzer** class :

// Buzzer.h

#pragma once

#include "Arduino.h"

#include "Pitches.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Buzzer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class to manage asynchonous tone on pin 11 using Timer2 CompA interrupts

and PWM phase-correct with top in OCRA

\*/

extern "C" void TIMER2\_COMPA\_vect**();**

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

class Buzzer

**{**

public**:**

Buzzer**();**

void tone**(**Note note**);**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**);**

bool isPlaying**()** const **{return** \_isPlaying**;}**

private**:**

unsigned int \_idNote**,** \_nbNote**;**

unsigned int \_idPlay**,** \_nbPlay**;**

bool \_okTone**;**

Note **\***\_tNote**;**

unsigned long \_endTime**;**

void funcISR**();**

bool \_isPlaying**;**

friend void TIMER2\_COMPA\_vect**();** // To use private funcISR() in the ISR

**};**

// Buzzer.cpp

#include "Buzzer.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ISR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Buzzer**\*** pBuzzer**;** // As static

ISR**(**TIMER2\_COMPA\_vect**)**

**{**

pBuzzer**->**funcISR**();**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Buzzer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Buzzer**::**Buzzer**()**

**{**

pBuzzer **=** **this;**

pinMode**(**11**,** OUTPUT**);**

\_isPlaying **=** **false;**

// WGM2 = 0b101 => Phase-correct PWM with TOP in OCR2A

TCCR2A **=** **(**1**<<**WGM20**);**

TCCR2B **=** **(**1**<<**WGM22**);**

**}**

void Buzzer**::**tone**(**Note note**)**

**{**

// End time

**if** **(**note**.**len **>** 0**)** \_endTime **=** millis**()** **+** note**.**len**;**

**else** \_endTime **=** 0xFFFFFFFF**;**

unsigned int freq **=** note**.**freq**;**

**if** **(**freq **>** 0**)**

**{**

// CS2 and OCR2A

unsigned CS2**;**

unsigned int fact**;**

**if** **(**freq **<** 60**)** **{**CS2 **=** 7**;** fact **=** 3906**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 1024

**else** **if** **(**freq **<** 120**)** **{**CS2 **=** 6**;** fact **=** 15625**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 256

**else** **if** **(**freq **<** 240**)** **{**CS2 **=** 5**;** fact **=** 31250**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 128

**else** **if** **(**freq **<** 480**)** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 64

**else** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** round**(**fact**/**freq**);}** // Prescaler = 64

// Enable PWM output on pin 11 with 50% duty cycle (Toggle OC2A on compare match)

bitSet**(**TCCR2A**,** COM2A0**);**

// Set prescaler using CS2

TCCR2B **&=** 0xF8**;** // Clear CS2 bits

TCCR2B **|=** CS2**;** // Set CS2 bits

// Enable COMPA interrupt

TIMSK2 **=** **(**1**<<**OCIE2A**);**

// Tone flag

\_okTone **=** **true;**

**}**

**}**

void Buzzer**::**play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

\_tNote **=** tNote**;**

\_idNote **=** 0**;**

\_nbNote **=** nbNote**;**

\_idPlay **=** 0**;**

\_nbPlay **=** nbPlay**;**

**if** **(**\_nbNote **>** 0**)**

**{**

\_isPlaying **=** **true;**

tone**(**\_tNote**[**0**]);**

**}**

**}**

void Buzzer**::**funcISR**()**

**{**

**if** **(**millis**()>=**\_endTime**)**

**{**

**if** **(**\_okTone**)**

**{**

bitClear**(**TCCR2A**,** COM2A0**);** // Stop PWM output on pin 11 (but keep COMPA interrupt active)

\_okTone **=** **false;** // Tone flag

\_endTime **+=** 50**;** // 50ms delay after tone

**if** **(**\_idNote **==** \_nbNote**-**1**)** \_endTime **+=** 200**;**

**}**

**else**

**{**

\_idNote**++;** // Next note

**if** **(**\_idNote **==** \_nbNote**)**

**{**

\_idNote **=** 0**;**

\_idPlay**++;**

**}**

**if** **(**\_idPlay **<** \_nbPlay**)** tone**(**\_tNote**[**\_idNote**]);** // Tone call

**else**

**{**

TIMSK2 **=** 0**;** // Stop COMPA interrupt

\_isPlaying **=** **false;**

**}**

**}**

**}**

**}**

With all that staff, the sketch is very simple now :

// TestBuzzer5.ino

// Test the buzzer using a Buzzer class

#include "Buzzer.h"

Buzzer buzzer**;**

Note melody**[]** **=** **{{**NOTE\_C4**,**250**},{**NOTE\_G3**,**125**},{**NOTE\_G3**,**125**},{**NOTE\_A3**,**250**},{**NOTE\_G3**,**250**},{**0**,**250**},** **{**NOTE\_B3**,**250**},{**NOTE\_C4**,**250**}};**

void setup**()**

**{**

buzzer**.**play**(**melody**,** 8**,** 2**);**

**}**

void loop**()**

**{**

**}**

NB : Because the interrupts are stopped in the ISR, here the **millis()** function will not return a correct value at the end of the ISR, so itsn’t easy to evaluate precisely the execution time of the ISR. But the results should be identical to that of **TestBuzzer4.ino.**

1. Such a signal is a particular PWM signal with a 50% duty cycle. [↑](#footnote-ref-2)
2. The piezoelectric effect is the generation of an electric charge when a crystalline structure is submitted to a pressure force (<https://www.electronicdesign.com/power-management/article/21801833/what-is-the-piezoelectric-effect> for instance). [↑](#footnote-ref-3)
3. The classical **La** in latine notation. [↑](#footnote-ref-4)
4. <https://en.wikipedia.org/wiki/Musical_note#Note_frequency_(hertz)> [↑](#footnote-ref-5)
5. <https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/> [↑](#footnote-ref-6)
6. <https://www.arduinolibraries.info/libraries/tone> [↑](#footnote-ref-7)
7. With a silence (null frequency). [↑](#footnote-ref-8)
8. Adapted from <https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody> [↑](#footnote-ref-9)
9. <https://www.arduino.cc/reference/en/libraries/elapsedmillis/> [↑](#footnote-ref-10)
10. Cf . <Arduino folder>/hardware/arduino/avr/cores/arduino/Tone.cpp [↑](#footnote-ref-11)
11. <https://bitbucket.org/teckel12/arduino-new-tone/wiki/Home> [↑](#footnote-ref-12)
12. <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf> [↑](#footnote-ref-13)
13. <https://www.arduino.cc/en/pmwiki.php?n=Tutorial/SecretsOfArduinoPWM> [↑](#footnote-ref-14)
14. Nearest integer. [↑](#footnote-ref-15)