圧電ブザー…音を出す

《何ができる？》圧電ブザーは圧電サウンダともいい、電極をつないだ圧電セラミックスの板と金属板を貼り合わせた単純な構造を持つ。電圧をかけることで圧電セラミックスが伸び縮みするが金属板はそのまま。二つの動きの違う板が貼り合わさっているため板同士が音の変化に合わせて高速に反り返り、音を出す。
構造がシンプルでコストをかけずに音を出すことができるが、小さい 2 枚の板の動きによる音ですので、音量が大きくなく音質もよくない。
そのため単純な通知音や、簡単なメロディに用いられている。

音の発生には、PWM（Pulse Width Modulation：パルス幅変調）を用いる。
PWM には、LED の明暗を制御する ledcWrite()関数を用いたが、これとは別に ledcWriteTone()関数や ledcWriteNote()関数を使う。

LED の明暗の制御に使った ledcWrite()関数は、デューティ比 0～255 の数値で出力の増減を変更したが、音程（音の高さ・低さ）を制御するには記述に工夫が必要になる。そのため音を出すために用意されたアナログ（PWM）出力関数を使うほうが良い。

ledcWriteTone()関数では、出したい音の周波数[Hz]を指定してやることで音を出す。
ledcWriteNote()関数では、出したい音の音程（ドレミ…）とそのオクターブ（音階を指定してやることで音を出すため音楽の演奏に向いている。

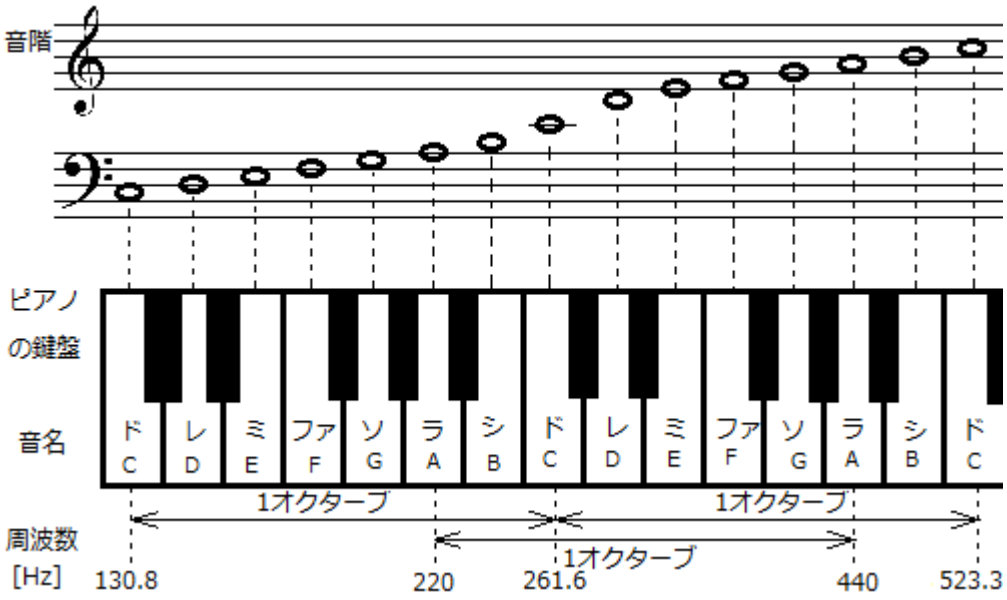
音を出す際に使うアナログ（PWM）出力関数

ledcWriteTone(uint8_t channel, double freq)	
指定した周波数を、デューティ比 50%で出力する。	
【パラメータ】	channel：利用するチャネル。0～15 の値。 ledcAttachPin()で設定したチャネル。 freq：出したい音の周波数[Hz]の値。

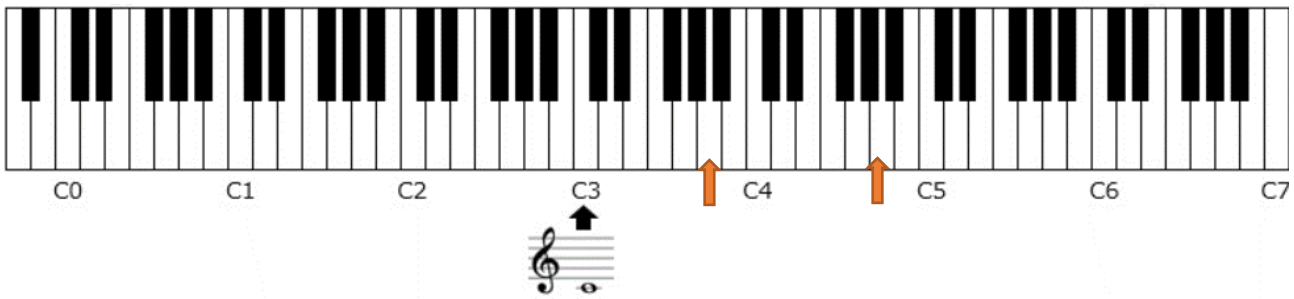
ledcWriteNote(uint8_t channel, note_t note, uint8_t octave)	
指定した音符を出力する。	
【パラメータ】	channel：利用するチャネル。0～15 note：出したい音の音程 octave：出したい音の音階（オクターブ）

出したい音の音階(note)で、以下の通りです。

値	出力
NOTE_C	ド
NOTE_Cs	ド＃
NOTE_D	レ
NOTE_Eb	ミ♭
NOTE_E	ミ
NOTE_F	ファ
NOTE_Fs	ファ＃
NOTE_G	ソ
NOTE_Gs	ソ＃
NOTE_A	ラ
NOTE_Bb	シ♭
NOTE_B	シ



出したい音の音階（octave）は、以下の通りです。



楽譜で五線譜の下にでるドの音符が、オクターブ 3 のド(C)になる。
また、オクターブが 3 のラ(A)が周波数 440[Hz] は ISO で定められた基準周波数で、その上のオクターブが 4 のラは周波数 880[Hz]とあわせて「ピ・ピ・ピ・ピーン」と鳴る NHK の時報でおなじみ。

《サンプル①》ledcWriteTone()関数で周波数を指定して音を出す。

```
#define LEDC_CHANNEL_0     0  //LEDCのPWMチャネル0から15
#define LEDC_TIMER_13_BIT  13 //LEDCタイマーの精度13ビット
#define LEDC_BASE_FREQ     5000 //LEDCのベース周波数 5000Hz

const int buzPin = 23; // 圧電ブザーのピン番号 GPIO IO23pin
const int btnPin = 17; // GPIO IO17pin
int noteDuration = 1000/8;  // 八分音符♪の長さ

void setup() {
  ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_13_BIT);
  ledcAttachPin(buzPin, LEDC_CHANNEL_0);
  pinMode(btnPin, INPUT);
}

void loop() {
  if(digitalRead(btnPin)==HIGH){
    // 低い周波数から高い周波数へ
    for(int i=25;i<120;i++){
      ledcWriteTone(LEDC_CHANNEL_0, 20*i);
      delay(noteDuration);
    }
    // 低い周波数から高い周波数へ
    for(int i=120;i>=25;i--){
      ledcWriteTone(LEDC_CHANNEL_0, 20*i);
      delay(noteDuration);
    }
  }
  //第2引数の周波数を0にすると音が消える
  ledcWriteTone(LEDC_CHANNEL_0, 0);
}
```

演習では、上記サンプルを動かした後、課題用に書きかえる。

`setup()`関数内にある、`ledcSetup()`関数および、`ledAttachPin()`関数は、LED の明暗の制御に使った `ledcWrite()`関数と同じもので、アナログ(PWM)出力したいピンと LEDC のチャネルを結びつけるものです。指定の際の値は、`#define` マクロを使って定数として記述している。

7 行目、20 行目、25 行目の noteDuration 変数は、音の長さを指定している。

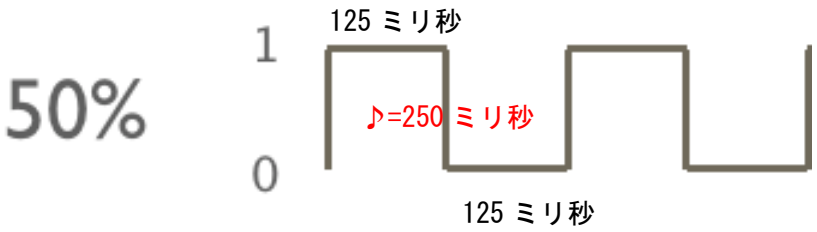
1000[ミリ秒]を 8[分音符]で割った値＝125 ミリ秒となり、120BPM ＝1 分に 4 分音符が 120 個あるテンポに近い値になっている。

60 秒 / 120（個の 4 分音符♩） = 0.5 秒/4 分音符♩1 個

8 分音符♩は 4 分音符♪より半分の長さになるので、1 分当たりの数は倍となって…

60 秒 / 240（個の 8 分音符♪） = 0.25 秒/8 分音符♪1 個 = 250 ミリ秒/8 分音符♪1 個

そうなると、前述の 1000[ミリ秒]を 8[分音符]で割った値＝125 ミリ秒の倍なのですが、「デューティ比 50%で出力」とあり、PWM の矩形波（パルス信号）の凸と凹の長さを示している。



演習

【目標】

圧電ブザーを使って、自由に音が出せるようになる。かんたんな楽譜を見て音符・休符などから音程・長さを割り出してコーディングできるようになる。

【1.　ESP32 と電子工作部品との接続】

1.1. 必要な部品

パーツ名	必要個数
圧電ブザー	1 個
ジャンパーコード	4 本

GPIO ボードに搭載されているので不要

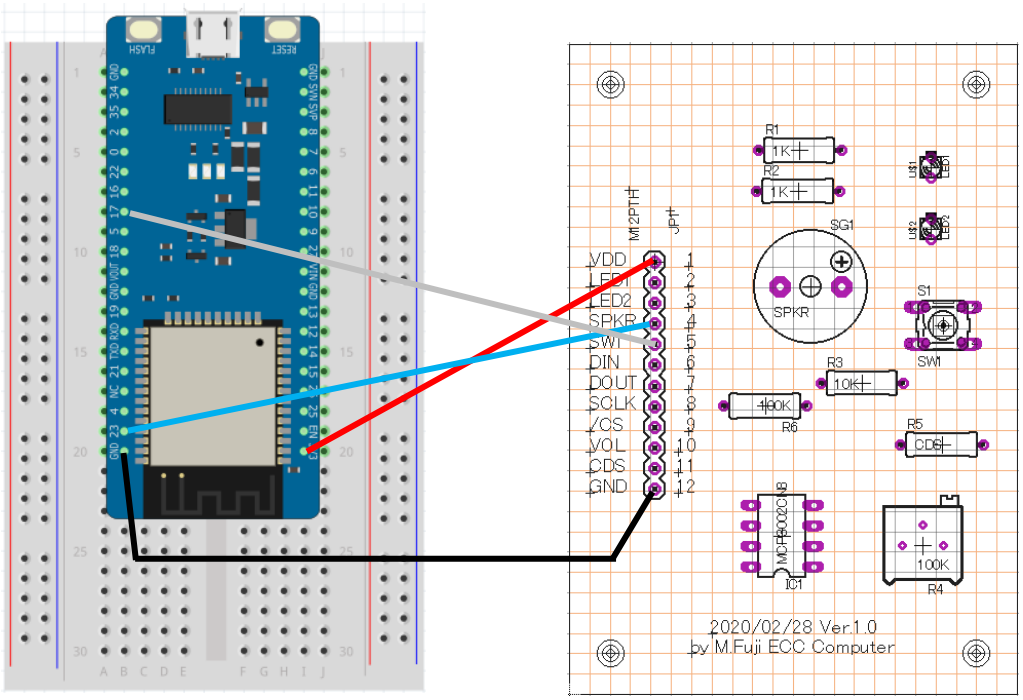
以下の接続を行う。

① ジャンパーコード（赤）ESP32 の 3V3 ― GPIO ボード VDD(1 番ピン)

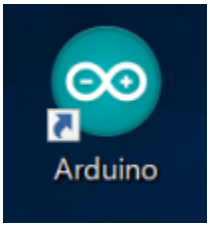
② ジャンパーコード（黒）ESP32 の GND ― GPIO ボード GND(12 番ピン)

③ ジャンパーコード（青）ESP32 の IO23 ― GPIO ボード SPKR(4 番ピン)

④ ジャンパーコード（白）ESP32 の IO17 ― GPIO ボード SW1(5 番ピン)



【2. Arduino スケッチのサンプルプログラムを実行】



デスクトップのアイコンをダブルクリックして
Arduino IDE を起動する。

【課題 14】プロジェクト名「kad14_toneSiren」

まずは、《サンプル①》を打ち込んで、動作確認をする。
音が目まぐるしく変わるサイレン音が流れるのを確認する。
次に、音が出ている最中に、シリアルモニタに周波数の値を表示するプログラムを追加して、値の変化を見られるようにする。

シリアルモニタ

```
500 520 540 560 580 600 620 640 660 680 700 720 740 760 780 800 820 840 860 880 900 920 940 960 980 1000 1020
2400 2380 2360 2340 2320 2300 2280 2260 2240 2220 2200 2180 2160 2140 2120 2100 2080 2060 2040 2020 2000 1980
```

for 文ブロックごとに表示。周波数の値は 1 スペース空けて表示される。
for 文が 1 つ終われば改行を入れる。
次の for 文のブロックでも周波数の値は 1 スペース空けて表示されるようにする。
結果として、500 から 20 おきに 2380 まで表示される行と、
2400 から 20 おきに 500 で表示される行が表示される。
以下は、右スクロールして改行されるところ。

```
1920 1940 1960 1980 2000 2020 2040 2060 2080 2100 2120 2140 2160 2180 2200 2220 2240 2260 2280 2300 2320 2340 2360 2380
1080 1060 1040 1020 1000 980 960 940 920 900 880 860 840 820 800 780 760 740 720 700 680 660 640 620 600 580 560 540 520 500
```

【課題 15】　プロジェクト名「kad15_toneTimeSignal」

Arduino スケッチは、まず前の課題からコピー＆ペーストすればよい。
loop()関数の内容を変更して、スイッチを押したら時報が鳴るようにする。
時報とは、NHK などで 00 分ちょうどに鳴る「ピ・ピ・ピ・ピーン」といったもので、440Hz の基準周波数を元にしている。

ピ	440Hz ラ(A3) 2 分音符
・	16 分休符
ピ	440Hz ラ(A3) 2 分音符
・	16 分休符
ピ	440Hz ラ(A3) 2 分音符
・	16 分休符
ピーン	880Hz ラ(A4) 全音符

時報がなるコードは以降に流用する予定なので、timeSignal()関数を作る。
loop()関数内でタクトスイッチが押されたら、時報が鳴るようにする。
timeSignal()関数は、以下の様な関数頭部と変数を持つようにする。

```
void timeSignal(void){
    double note_half = noteDuration*4;//2分音符
    double note_whole = noteDuration*8;//全音符
    double note_16th = noteDuration/2;//16分休符

    ledcWriteTone(LEDC_CHANNEL_0, 440);
```

←ヒント(^-^)

【課題 16】 プロジェクト名「kad16_noteChime」

ledcWriteNote()関数を使って「ピンポンパンポーン♪ピンポンパンポーン↘」という学校で鳴るチャイムの演奏を行うスケッチを作ってほしい。

以下のスケッチは圧電ブザー演奏のサンプルである。Arduino チュートリアルの Play a Melody using the tone() function と同等のメロディ（"punchline"というメロディ）で、日本ではがっかりしたときの「チャンチャチャチャチャンのチャンチャン！」というメロディ。

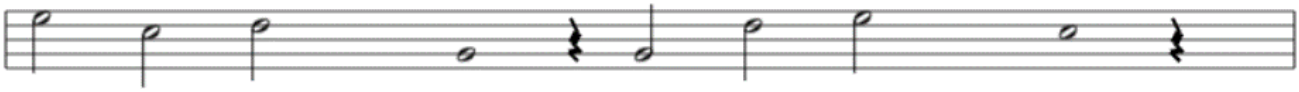
《サンプル②》

```
#define LEDC_CHANNEL_0     0  //LEDCのPWMチャネル0から15
#define LEDC_TIMER_13_BIT  13 //LEDCタイマーの精度13ビット
#define LEDC_BASE_FREQ     5000 //LEDCのベース周波数 5000Hz

const int buzPin = 23; // 圧電ブザーのピン番号 GPIO IO23pin

const int NOTE_NONE = NOTE_MAX; //休符
//各音符の音程（音の高さ）
int melody[] = {
  NOTE_C, NOTE_G, NOTE_G, NOTE_A, NOTE_G, NOTE_NONE, NOTE_B, NOTE_C
};
//各音符の音階（オクターブ）
int noteOctaves[] = {  4, 3, 3, 3, 3, 0, 3, 4 };
//各音符の長さ
int noteDurations[] = {  4, 8, 8, 4, 4, 4, 4, 4 };

void setup() {
  ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_13_BIT);
  ledcAttachPin(buzPin, LEDC_CHANNEL_0);
}

void loop() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    ledcWriteNote(LEDC_CHANNEL_0, (note_t)melody[thisNote], noteOctaves[thisNote]);
    int pauseBetweenNotes = 1000 / noteDurations[thisNote] * 1.30;
    delay(pauseBetweenNotes);
    ledcWriteTone(LEDC_CHANNEL_0, 0); // 演奏を止める。
  }
  delay(2000);
}
```

このソースを参考に、課題では「タクトスイッチを押したら、チャイムの『ピンポンパンポーン♪ピンポンパンポーン↴』というメロディの演奏がされる」プログラムを作成しなさい。

【チャイムの『ピンポンパンポーン♪ピンポンパンポーン↴』というメロディ】

チャイムの楽譜は以下の通り。



ミ E4　ド C4　レ D4　ソ G3　　ソ G3　レ D4　ミ E4　ド C4

音符と休符については以下の表が参考になります。

音符		音符、休符の示す長さ	休符	
表記	名称	全音（休）符の長さを「1」とした場合の比率	表記	名称
	全音符	1		全休符
	二分音符	1/2		二分休符
	四分音符	1/4		四分休符
	八分音符	1/8		八分休符
	十六分音符	1/16		十六分休符

サンプル②の演奏部分では、25 行目が音符・休符の長さを計算するところになっている。課題 16 では、そのままこの行の計算を使ってもよい。

```
24      ledcWriteNote(LEDC_CHANNEL_0, (note_t)melody[thisNote], noteOctaves[thisNote]);
25      int pauseBetweenNotes = 1000 / noteDurations[thisNote] * 1.30;
26      delay(pauseBetweenNotes);
27      ledcWriteTone(LEDC_CHANNEL_0, 0); // 演奏を止める。
```

この 25 行の計算式で、この演奏の BPM を計算すると…。

8 分音符・8 分休符の場合

1000（ミリ秒）/ 8（分音符）=　125 のところ、125 * 1.30 = 162.5 ミリ秒/8 分音符 1 個のパルス

ここからパルス凸凹の 2 倍、4 分音符で長さ 2 倍とすると

(162.5 ミリ秒/8 分音符 1 個 * 2) * 2 = 650 ミリ秒/4 分音符 1 個

1 分に 4 分音符が入る数は 60*1000 ミリ秒/650 ミリ秒/4 分音符 1 個 = 92.30 BPM となる。