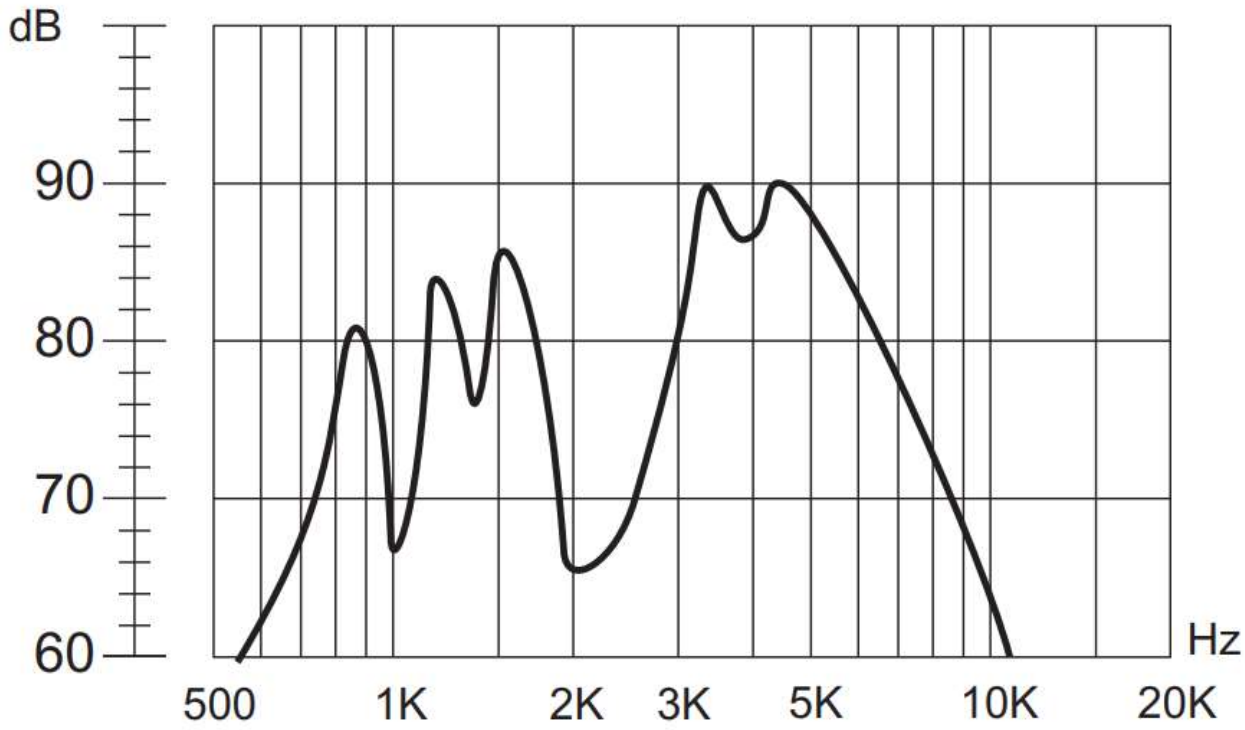
Пьезоэлектрический излучатель. Тестирование

Пьезоэлектрический динамик (он же зуммер, beeper, buzzer) является механическим устройством. Для того чтобы извлечь звук, необходимо заставить его мембрану колебаться. Сделать это можно подав напряжение на его выводы, а затем снять его. Например, что бы получить ноту Ля в первой октаве нужно сделать так 440 раз за секунду.

Зуммеры обладают разными характеристиками, и низкие частоты (скажем, ниже 1 кГц) могут и не воспроизвести совсем. Более того, на некоторых частотах могут быть провалы. Проведём эксперимент.



Пример характеристики, датчик HPM14A

Если же мы будем посылать не меандр, а использовать, скажем, ШИМ, мы сможем вдобавок регулировать громкость заполнением импульса.

Поставим перед собой простую задачу: заставим нашу «пищалку» извлечь линейно нарастающий звук от 0 до 20 кГц. Записав его, можно построить характеристику. Составим заголовочный файл, добавив пару функций для работы с громкостью.

```
// buzzer.h
void buzeer_init(const uint8_t vol);
void buzzer_set_volume(const uint8_t vol);
void buzzer_change_volume(int16_t diff);
void buzzer_test(const uint16_t freq);
```

Напишем функцию инициализации используя код из прошлого раздела (см. Широтно-импульсная модуляция).

```
#define PRESCALER       (SystemCoreClock / 1e6)

static volatile uint8_t volume = 0;

void buzeer_init(const uint8_t vol) {
    volume = vol;

    // RCC
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    // PA2 as output, alternative, push-pull
    GPIOA->CRL &= ~GPIO_CRL_CNF2;
    GPIOA->CRL |= GPIO_CRL_CNF2_1;
    GPIOA->CRL &= ~GPIO_CRL_MODE2;
    GPIOA->CRL |= GPIO_CRL_MODE2_1;

    // TIM2, timer 2, channel 3
    TIM2->CR1 &= ~TIM_CR1_DIR; // upcounting
    TIM2->CR1 &= ~TIM_CR1_CMS; // edge-align mode
    TIM2->PSC = PRESCALER;     // 64 => 1 us
    TIM2->ARR = 0;             // 0 Hz
    TIM2->CCR3 = 0;            // 0% duty
    TIM2->CCER |= TIM_CCER_CC3E | TIM_CCER_CC3P; // 3 channel, low level
    TIM2->CCMR2 &= ~TIM_CCMR2_CC3S;              // output
    TIM2->CCMR2 = TIM_CCMR2_OC3M_2 | TIM_CCMR2_OC3M_1; // PWM1 mode
}
```

Для запуска и отключения таймера воспользуемся двумя `inline`-функциями. Так как они будут точно встраиваемыми, то определение можно поместить в заголовочный файл.

```
// buzzer.h
__attribute__((always_inline)) inline void buzzer_turn_on(void) {
    TIM2->ARR  = 0;
    TIM2->CCR3 = 0;
    TIM2->CR1 |= TIM_CR1_CEN;
}

__attribute__((always_inline)) inline void buzzer_turn_off(void) {
    TIM2->CR1 &= ~TIM_CR1_CEN; // turn off timer
}
```

Для тестирования напишем небольшую функцию, которая устанавливает нужную нам частоту и громкость:

```
#define SET_FREQ(x)     TIM2->ARR  =  (1e6 / x); \
                        TIM2->CCR3 = ((1e6 / x) * volume) >> 8

void buzzer_test(const uint16_t freq) {
    SET_FREQ(freq);
}

void buzzer_set_volume(const uint8_t vol) {
    volume = vol;
}
```

Последнее что нужно сделать — дописать функцию для энкодера. Она будет очень простая.

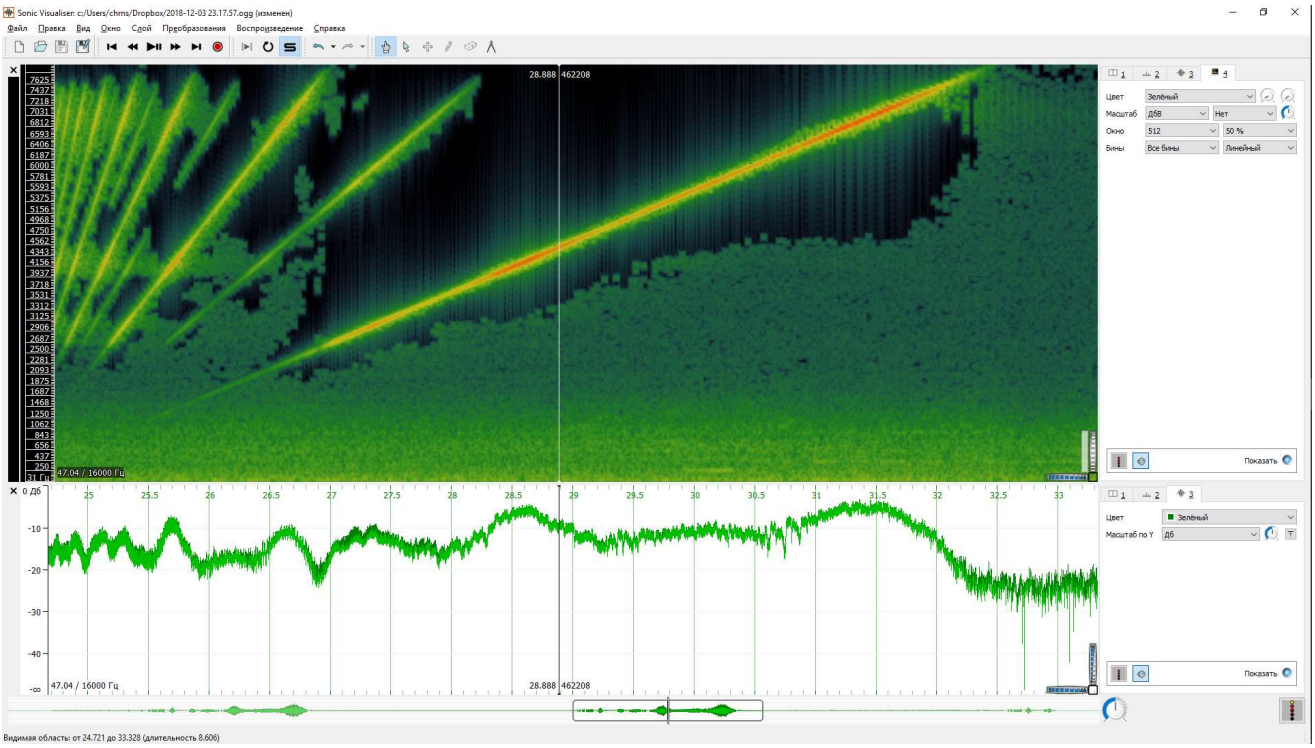
```
void buzzer_change_volume(int16_t diff) {
    volume += diff;
}
```

> Обратите внимание, максимальная громкость достигается при 50% заполнении ШИМ. Любое отклонение от 50%, в любую сторону, приведёт к понижению уровня громкости. Так же в текущей реализации есть проблема: присмотритесь к макросу `SET_FREQ()` .

Запустим тестовую программу и запишем звук.

```
int main(void) {
    mcu_init(); // --> encoder_init(buzzer_change_volume);
                // --> buzzer_init(127);
    while (1) {
        buzzer_turn_on();
        for (uint32_t freq = 0; freq < 20000; freq += 10) {
            buzzer_test(freq);
            delay(10000);
        }
        buzzer_turn_off();
    }
}
```

Ниже приведен скриншот аудиозаписи (записанной на планшет) в программе **Sonic Visualiser**.



Как видите характеристика не очень радужная. Но что имеем, то имеем.

Код урока можно найти на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**