Tone-output

Arduino-experiment	130135-EN	*Trefwoorden*	tone(), blokkerende vs. niet-blokkerende functies, arrays		
Versie	2018-07-05 / HS	*Niveau*	Basiscursus, module 6	*p.*	1/4

Wat je leert:

- Maak een blokgolf op een output pin
- Begrijp blokkerende en niet-blokkerende functies
- Gebruik arrays
- Gebruik een passieve piëzo-zoemer

1 – Sluit een piëzo-zoemer aan op een breadboard

Verschillende soorten zoemers

Monteer de zoemer zoals afgebeeld – let op de niet-aangesloten draad.

In dit project zullen we de piëzo-zoemer als luidspreker gebruiken. Er zijn twee soorten zoemers, waarvan er slechts één voor dit doel kan worden gebruikt. Het is dus aan te raden om de zoemer te testen voordat je verder gaat:

Sluit één pin van de zoemer aan op GND en sluit een draad aan op de andere pin (zoals afgebeeld). Kijk of er een polariteitsmarkering (+ of -) aanwezig is. Sluit de draad kortstondig aan op 5 V. Als de zoemer zoemt of piept is het een zogenaamde actieve zoemer die niet kan worden gebruikt. Als hij een kleine tik of rammel geeft is het waarschijnlijk OK.

Na de test wordt de draad verbonden met één van de genummerde Arduino-pins (bijvoorbeeld pin 9).

Schrijf het volledige programma in de setup()-functie

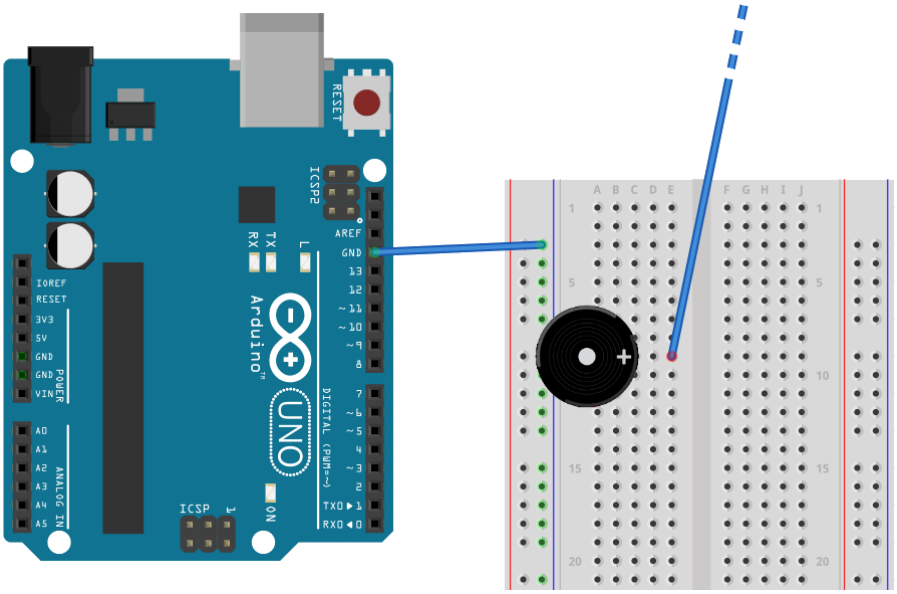
Om te kunnen horen wanneer het zoemen begint en stopt zullen we het grootste deel van het programma niet in `loop()` schrijven, zoals we gewend zijn te doen. Door alles in `setup()` te schrijven worden de regels slechts één keer uitgevoerd. Invoeren en testen:

```
const byte pinOut = 9;

void setup() {
  tone(pinOut,440,250); // Concert pitch (440 Hz) a quarter of a second (250 ms).
}

void loop() {            // Nothing here
}
```

Zoals je ziet is er geen configuratie van de output-pin. Dit is ook niet nodig.



Als je klaar bent met de eerste test, sluit je de draad aan op Arduino-pin 9.

Uitdaging 1

Om een grote drieklank te produceren, kun je de frequenties 440 Hz, 554 Hz, 659 Hz en 880 Hz gebruiken.

Sla het programma op. Probeer drie extra regels die `tone()` aanroepen toe te voegen om het programma een drieklank te laten spelen.

Het programma moet vier noten met toenemende frequentie spelen.

Werkt het zoals je had verwacht? Wellicht wil je teruggaan naar het oorspronkelijke programma en het geluid vergelijken.

2 – Blokkeren en niet-blokkeren van functieoproepen

Waarom werkt het niet?

De meeste functies die we in de vorige modules hebben gebruikt, starten (natuurlijk) wanneer ze worden aangeroepen – en geven pas de controle terug aan het programma als ze klaar zijn. Een goed voorbeeld is `delay()` – er gebeurt precies niets terwijl we wachten. Dit type wordt een *blokkerende* functie genoemd; de functie blokkeert verdere uitvoer van het programma totdat het einde van de functie wordt bereikt.

Het is mogelijk om functies te schrijven die een bepaalde taak starten, maar dan direct terugkomen nadat ze zijn aangeroepen – waarna de taak op de achtergrond verder gaat. Dit wordt een *niet-blokkerende* functie genoemd.

De `tone()`-functie van Arduino is niet-blokkerend. De eerste drie regels in je programma beginnen elk slechts – waarna ze worden onderbroken door de *volgende* aanroep van `tone()` . Alleen de laatste aanroep “overleeft”.

Dit is in veel situaties een prachtig slimme functionaliteit – maar hier is het ongewenst.

Schrijf een blokkeringsfunctie die tonen afspeelt

Dit is niet bijzonder elegant, maar het werkt wel: voeg een regel met `delay(300);` toe na elke aanroep van `tone()` in het programma. Testen maar. Het zou moeten werken…

De extra 50 ms pauze voor elke toon maakt het makkelijker om de afzonderlijke tonen te onderscheiden.

OK – we beginnen het onder de knie te krijgen – tijd om een beetje structuur aan het programma toe te voegen. In plaats van het aanroepen van twee verschillende functies voor elke toon, zullen we een nieuwe functie `play()` schrijven die dan `tone()` en `delay()` moet aanroepen.

We blijven werken met frequenties, maar de milliseconden moeten worden vervangen door iets “muzikalers”. Als je werkt met noten geef je de duur ervan aan als fracties van een hele noot – hier hebben we ervoor gekozen om te tellen in zestienden. Dit moet worden omgezet in milliseconden door de functie die we schrijven.

Ergens in het programma definiëren we de conversiefactor als een constante – de waarde 100 is geschikt.

```
const int sixteenthsToMillis = 100;
```

Nu kan de duur van een hele noot worden opgegeven als 16, een halve noot als 8, enz.

Uitdaging 2

Schrijf een functie `void play(int freq, int sixteenths)` die een toon speelt die `sixteenths` lang is en een frequentie van `freq` heeft. De duur moet worden aangepast door `sixteenthsToMillis` te veranderen in een andere waarde.

Bij het aanroepen zoals hieronder getoond, moet het resultaat een grote drieklank zijn (met een kleine vertraging tussen de tonen).

```
play(440, 2);
play(554, 2);
play(659, 2);
play(880, 6);
```

(Als je een extreem goed gevoel voor ritme hebt, *kun* je het gevoel hebben dat de beste resultaten worden bereikt door de gecombineerde duur van toon + kleine vertraging te laten overeenkomen met de opgegeven duur. In de praktijk kan dit worden gedaan door de berekende duur direct in `delay()` te gebruiken terwijl `tone()` wordt aangeroepen met een duur die bijvoorbeeld 50 ms korter is.)

3 – Arrays

Als we met de Arduino een langere melodie willen spelen, zijn er nogal wat regels nodig. Het zou een beter overzicht geven als we de tonen gewoon in een lange rij konden schrijven (evenals de duur).

We zullen nu een manier introduceren om met dit soort gegevens om te gaan.

Tot nu toe werden de getallen die we in programma's hebben gebruikt in variabelen geplaatst, die elk een specifieke naam hebben. Door gebruik te maken van een datastructuur die een *array* wordt genoemd, wordt een rij getallen aangesproken met behulp van één naam en een *index*.

Zie de toolbox rechts.

Dit is precies wat we nodig hebben…

In het hieronder getoonde stukje code zijn twee arrays gedefinieerd met resp. de frequenties en de bijbehorende duur voor een aantal tonen.

Er is ook een variabele `notes`, die de lengte van de twee arrays aangeeft. Merk op dat het wordt gegeven als `sizeof(times)`, wat werkt omdat `times` een array van *bytes* is. Er is hier een punt: als je later meer noten en looptijden wilt toevoegen, wordt deze variabele automatisch bijgewerkt met de nieuwe lengte.

Toolbox: Arrays

Voorbeeld:

```
const int fr[] = {440,554,659,880};
```

Dit definieert een *reeks (array)* integers met de lengte 4. De afzonderlijke array-elementen zijn genummerd van 0 tot 3, dit getal wordt de *index* van het element genoemd.

`fr[2]` heeft bijvoorbeeld de waarde 659.

Voorbeeld:

```
long a[7];
```

Deze reserveert ruimte voor een array van `long` (4 byte integers) met lengte 7. De elementen hebben ongedefinieerde beginwaarden.

Voorbeeld:

```
int x = sizeof(fr);
int y = sizeof(a);
```

Na deze regels heeft `x` de waarde 8 (er zijn 4 elementen van 2 bytes), terwijl `y` de waarde 28 heeft (d.w.z. 7 elementen van 4 bytes).

Als elk element 1 byte in beslag neemt, geeft

De afzonderlijke elementen in de twee arrays worden geïdentificeerd door de index van het element tussen **[** rechte haakjes **]** te specificeren. De nummering begint bij 0.

De waarde van `freq[0]` is 294. De waarde van `times[7]` is 6.

```
const int freq[] =   {294,294,294,392,392,440,440,588,494,392};
const byte times[] = {  1,  3,  1,  4,  4,  4,  4,  6,  2,  3};
const byte notes = sizeof(times);
```

Arrays worden goed verwerkt in een `for` loop met een loopvariabele `i` die als index kan worden gebruikt. In dit voorbeeld moet de loop beginnen met `i=0` en eindigen met `i=notes-1`. Dit betekent dat de loopvoorwaarde `i<notes` is.

Uitdaging 3

Maak het programma af. De tonen in de `freq`-array moeten worden afgespeeld met de bijbehorende tijdsduur in de `times`-array. Gebruik hiervoor een `for` loop.

4 – Meer details over tone( )

Hier zijn nog een paar willekeurige details over deze functie…

Een van de ideeën achter de keuze om `tone()` een niet-blokkerende functie te maken is om geluidssignalen te kunnen uitzenden zonder de resterende taken van de Arduino te onderbreken. Je kunt bijvoorbeeld een waarschuwingspiep versturen en doorgaan met het meten, berekenen en controleren van verschillende uitgangen gedurende de tijd dat de pieptoon duurt.

Je kunt de `tone()` met slechts één parameter (de frequentie) aanroepen. Dit zal de toon laten klinken totdat deze wordt gestopt door het aanroepen van de functie `noTone()`.

Wanneer je `tone()` gebruikt, kun je niet tegelijkertijd pin 3 of 11 voor PWM-signalen gebruiken. (Zie 130125-EN Arduino als dimmer.)

5 – Een muzikale deurbel

Zo ver als je nu gekomen bent, ben je vast in staat om een van de technologische wonderen van de moderne wereld te creëren: de muzikale deurbel ☺

Je moet een drukknopschakelaar toevoegen en de pin die je gaat gebruiken configureren in `setup()`.

De melodie moet niet beginnen wanneer de Arduino wordt ingeschakeld, dus je moet het spelen weghalen uit de `setup()` functie.

In `loop()` heb je een `if`-constructie nodig die controleert of de knop is ingedrukt. Zodra dat het geval is, moet de korte melodie spelen.

Uitdaging 4

Schrijf op basis van bovenstaande ideeën het programma voor de muzikale deurbel.

Als het klaar is, moet het zich zo gedragen: wanneer de knop wordt ingedrukt en losgelaten, moet de melodie één keer worden afgespeeld. Als de knop ingedrukt wordt gehouden, moet de melodie worden herhaald totdat de knop wordt losgelaten en de melodie voorbij is.