Frederiksen
Passion for science

Tone output

Arduino-eksperiment	130135	*Stikord*	tone(), blocking vs. non-blocking funktioner, arrays	
Version	2018-05-18 / HS	*Niveau*	Grundkursus, modul 6	*p.* 1/4

Det lærer du:

- Lav en firkanttone på et udgangsben
- Forstå, hvad blocking og non-blocking funktioner er
- Brug arrays

- Anvend en passiv piezo-buzzer

1 – Tilslut en piezo-buzzer på et breadboard

Forskellige typer buzzere

Start med at montere buzzeren som vist på figuren – bemærk den løse ledning.

Vi skal i dette projekt bruge en piezo-buzzer som en højttaler. Der findes flere typer buzzere, og nogle af dem er uanvendelige til dette formål, så du skal nok teste din buzzer, inden du prøver at programmere:

Forbind det ene ben af buzzeren til GND og forbind en ledning til det andet ben (se nedenfor). Observer, om der er en markering af polaritet (+ eller -). Forbind kortvarigt ledningen til 5 V. Hvis buzzeren summer eller hyler, er det en såkaldt aktiv buzzer, som ikke kan bruges. Hvis den giver et lille skrat eller tick fra sig, er den sandsynligvis OK.

Efter testen forbindes ledningen til et af benene på Arduinoen (f.eks. ben 9).

Skriv hele programmet i setup()-funktionen

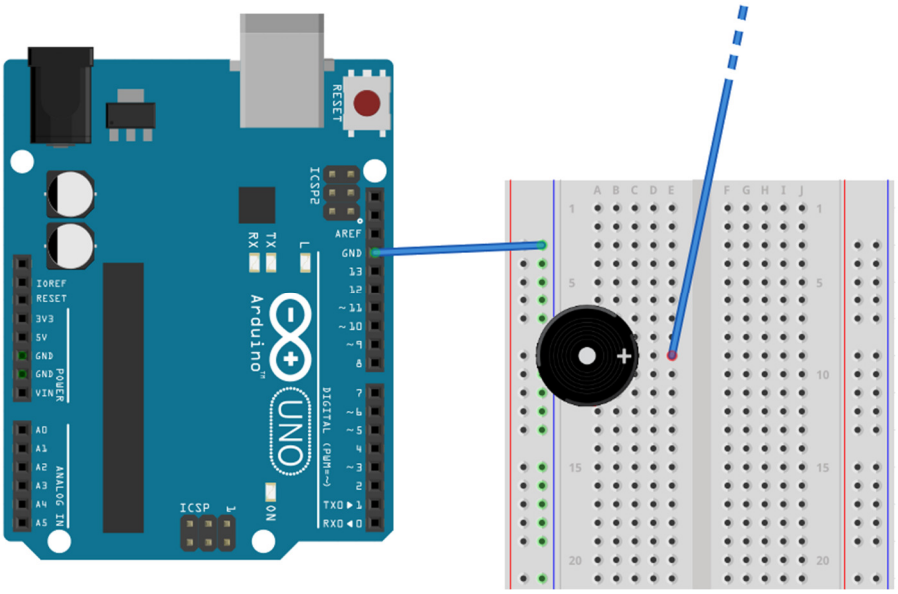
For at kunne høre, hvornår hyleriet begynder og slutter, vil vi ikke skrive programmet i `loop()`, som vi plejer. Ved at skrive det hele i `setup()`, udføres linjerne kun én gang. Tast ind og prøv af:

```
const byte pinOut = 9;

void setup() {
  tone(pinOut,440,250); // Kammertonen (440 Hz) i et kvart sekund (250 ms).
}

void loop() {           // Ingenting hér.
}
```

Bemærk, at der ikke er behov for nogen konfigurering af output-benet.



Den løse ledning sluttes til ben 9 på Arduinoen, når den indledende test er afsluttet.

Udfordring 1

Hvis man vil frembringe en dur-treklang, kan man bruge frekvenserne 440 Hz, 554 Hz, 659 Hz og 880 Hz.

Gem programmet. Prøv at tilføje tre ekstra linjer med kald af `tone()` for at få programmet til at spille en treklang.

Programmet bør altså afspille fire toner med voksende frekvens.

Fungerede det, som du troede? Prøv evt. at gå tilbage til det første program og sammenlign lyden.

2 – Blocking og non-blocking funktionskald

Hvorfor virker det ikke?

De fleste af de funktioner, vi har arbejdet med i de tidligere projekter i grundkurset, starter naturligvis idet de bliver kaldt – og returnerer ikke kontrollen til programmet, før de er afsluttet. Et godt eksempel er `delay()` – der sker lige præcis ikke en bønne, mens vi venter. Denne type funktioner kaldes *blocking*; funktionen blokerer for yderligere programudførelse, indtil den er afsluttet.

Det kan lade sig gøre at lave funktioner, som starter en given opgave, og som returnerer *straks* efter de blev kaldt – hvorefter opgaven kører videre i baggrunden. Dette kaldes en *non-blocking* funktion.

Arduinos `tone()`-funktion er non-blocking. Dine tre første programlinjer med kald til denne funktion når hver især kun lige akkurat at begynde, hvorefter de bliver afbrudt af det *næste* kald af `tone()` . Kun det sidste kald "overlever".

Dette er fantastisk smart i mange situationer – men lige hér er det ikke det, vi ønsker.

Skriv en blocking funktion, som spiller toner

Dette er ikke specielt elegant, men det virker: Tilføj en linje med `delay(300);` efter hvert kald af `tone()` i programmet. Prøv af. Det skulle meget gerne virke…

De 50 ms ekstra pause for hver tone gør det lettere at opfatte de enkelte toner.

OK – nu har vi fat med neglene – så er det tid at sætte lidt struktur på programmet. I stedet for at kalde to forskellige funktioner pr. tone, vil vi skrive en ny funktion `play()` , som så skal kalde `tone()` og `delay()` .

Vi arbejder videre med frekvenser, men millisekunderne skal erstattes af noget mere "musikalsk". Når man arbejder med noder, angives deres varighed som brøkdele af en helnode – vi vil her vælge at regne i sekstendedele. Så må funktionen laves, så den regner om til millisekunder.

Et sted i programmet vil vi notere omregningsfaktoren som en konstant – 100 er en passende faktor:

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

Nu kan varigheden af en helnode angives som 16, en halvnode 8 osv.

Udfordring 2

Skriv en funktion `void play(int freq, int sixteenths)` , der afspiller en tone, som er `sixteenths` lang og har frekvensen `freq` . Varigheden skal kunne justeres ved at ændre `sixteenthsToMillis` til en anden værdi.

Når funktionen bliver kaldt som vist herunder, skal den afspille en treklang (med en lillebitte pause mellem hver tone).

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

(Hvis du har en helt usædvanlig god rytmesans, vil du undervejs i dette projekt **muligvis** opdage, at det bedste resultat opnås, hvis det er den *samlede* varighed af tone + lillebitte pause, som svare til det angivne antal sekstendedele. Det opnås i praksis let ved at beregne en varighed, som bruges direkte i `delay()` , mens `tone()` kaldes med en varighed, som er f.eks. 50 ms kortere.)

3 – Arrays

Hvis vi ville spille en lidt længere melodi på Arduinoen, skal der efterhånden en hel del linjer til. Det kunne måske være mere overskueligt, hvis man blot kunne skrive tonerne i en lang række (og tilsvarende med varighederne).

Vi skal nu se en måde at håndtere denne type data på.

Hidtil har de tal, vi har arbejdet med i programmerne, ligget i hukommelsen som variabler under hvert sit navn. Ved at bruge en datastruktur kaldet et *array* kan en række tal adresseres via ét navn samt et *index*.

Se værktøjs-boks til højre.

Det er vist lige, hvad vi har brug for…

I programstumpen, som vises nedenfor, er der defineret to arrays med hhv. frekvenser og varigheder af en række toner.

Der er ligeledes defineret en variabel `notes`, som angiver længden på de to arrays. Bemærk, at den er givet ved `sizeof(times)`, hvilket fungerer, fordi `times` er et array af *bytes*. Der er en pointe her: Hvis man senere ønsker at tilføje flere toner og varigheder, bliver variablen `notes` automatisk opdateret med den nye længde.

Værktøj: Arrays

Eksempel:

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

Dette definerer et *array* af heltal med længden 4. De enkelte elementer i array'et nummereres fra 0 til 3, nummeret kaldes elementets *index*.

F.eks. har `fr[2]` værdien 659.

Eksempel:

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

Her sættes plads af til et array af `long` (4-bytes-heltal) med længde 7. Elementerne har ikke nogen bestemt værdi fra starten.

Eksempel:

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

Efter disse linjer har `x` værdien 8 (der er 4 elementer à 2 bytes), mens `y` har værdien 28 (7 elementer à 4 bytes).

Hvis hvert element fylder 1 byte, giver `sizeof()` netop array'ets længde.

De enkelte elementer i de to arrays udpeges ved at angive elementets index i [firkantklammer]; nummereringen starter med 0:

Værdien af `freq[0]` er 294. Værdien af `times[7]` er 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 passer godt til en for-løkke med en tælle-variabel `i`, som kan bruges som index. I vores eksempel skal løkken starte med `i=0` og slutte med `i=notes-1`. Dvs. løkke-betingelsen skal være `i<notes`.

Udfordring 3

Skriv programmet færdig, så tonerne i array'et `freq` afspilles med de tilhørende varigheder i `times`. Anvend en `for`-løkke til dette.

4 – Lidt mere om tone( )

Her følger et par løse detaljer om denne funktion:

En af ideerne ved at lave `tone()` som en non-blocking funktion er at kunne give lydsignaler, uden at Arduinoens yderligere opgaver behøver at ligge brak. F.eks. kan man give et advarsels-bip, og fortsætte med at måle, beregne og styre forskellige udgange i den tid, bip'et varer.

Man kan kalde `tone()` med kun én parameter (frekvensen). Så vil tonen lyde indtil den standses ved et kald af funktionen `noTone()`.

Når man ønsker at anvende `tone()`, kan man ikke samtidigt benytte ben 3 eller ben 11 til PWM-signaler. (Se 130125 Arduino som lysdæmper.)

5 – En musikalsk dørklokke

Nået så langt kan du nu frembringe et af den moderne verdens teknologiske mirakler: Den musikalske dørklokke ☺

Der skal tilføjes en afbryder, og det indgangsben, du vælger at bruge, skal konfigureres i `setup()`.

Der skal ikke spilles toner, når Arduinoen tændes, så du skal have flyttet spilleriet væk fra `setup()`-funktionen.

Nede i `loop()` skal du have en `if`-konstruktion, som checker, om knappen er trykket ned. Så snart den er det, skal melodistumpen afspilles.

Udfordring 4

Ud fra de ovenstående ideer skal du skrive programmet til den musikalske dørklokke.

Resultatet skal opføre sig således: Når knappen trykkes ned og slippes, skal melodien afspilles én gang. Hvis knappen holdes nede, skal melodien gentages, indtil knappen er sluppet, og melodien er færdig.