Tænd med en kontakt

Arduino-eksperiment	130115	*Stikord*	Logiske udtryk, if, else, digitalRead(), kommentarer	
Version	2018-05-18 / HS	*Niveau*	Grundkursus, modul 3	*p.* 1/4

Det lærer du:

- Læs tilstanden af en indgang med `digitalRead()`
- Regn med sand og falsk
- Styr programmets gang med betingelser

- Brug pull-up modstande
- Få kontrol over kontakt-prel

1 – Tilslut en kontakt på et breadboard

Trykkontakter

Der findes mange størrelser og typer på trykkontakter. Prøv at placere kontakten i opstillingen, som vist herunder. Den viste type afbryder har 4 ben, som er forbundet to og to. Anbragt som vist dannes der herved forbindelse mellem rækkerne hen over "gabet" i midten – det er ikke noget vi bruger her, men er vigtigt at vide.

Hvis ikke opstillingen virker, må du undersøge dine kontakter med et ohmmeter (uden Arduino tilsluttet). Det er meningen, at når knappen er trykket ned, skal kontakten lave forbindelse mellem række 7 og række 9 på breadboardet (nogle få ohms modstand). Når knappen er oppe, skal der være afbrudt (mange MegaOhm eller overflow).

Programmering

Indtast nedenstående program:

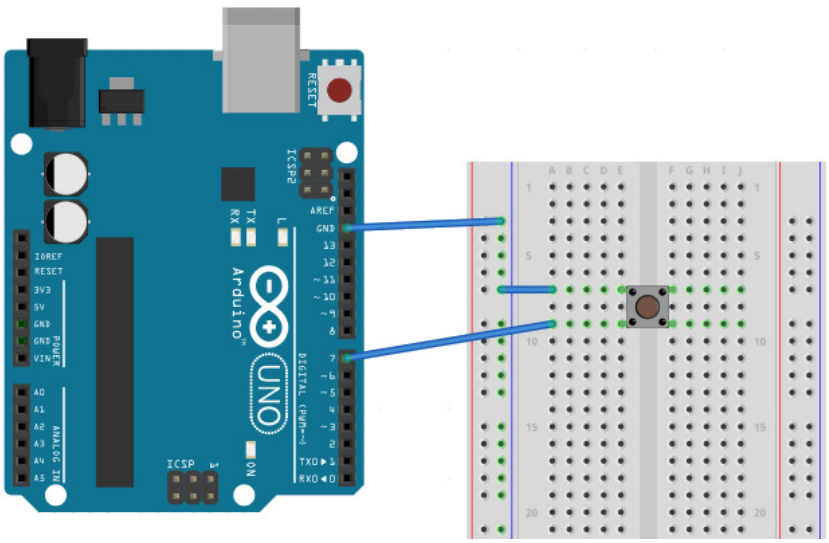
```
const byte pinLed = 13;
const byte pinSwitch = 7;

void setup() {
  pinMode(pinLed, OUTPUT);
  pinMode(pinSwitch, INPUT_PULLUP);
}

void loop() {
  if ( digitalRead(pinSwitch)==LOW ) {  // Kontakten trykket ned
    digitalWrite(pinLed, HIGH);
  } else {
    digitalWrite(pinLed, LOW);
  }
}
```

Hvis alting virker, skal den indbyggede lysdiode lyse, så længe kontakten er trykket ned.

Programmet gennemgås på næste side.



Den første nyhed kommer i `setup()` med linjen `pinMode(pinSwitch, INPUT_PULLUP);` Ben 7 bliver her gjort til input, og der tilsluttes internt en modstand, som forsøger at trække indgangen op på 5 V (= `HIGH`). Det kaldes en pull-up modstand. Modstanden er så stor, at afbryderen uden problemer kan trække udgangen ned på 0 V (= `LOW`), når knappen trykkes ned.

Næste nyhed er konstruktionen med

```
if ( … ) { … } else { … }
```

Disse 5 linjer virker således: Hvis knappen er trykket ned, skal der tændes for lysdioden – ellers skal der slukkes.

Betingelsen i runde parenteser (…) bruger funktionen `digitalRead()` til at læse tilstanden af indgangsbenet `pinSwitch` og sammenligner resultatet med `LOW`.

Det dobbelte lighedstegn "==" skal læses "er lig med" – i modsætning til "=", som læses "sættes lig med".

I oversigtsform gennemgås `if`-konstruktionen i værktøjsboksen her til højre.

En tredje nyhed er **kommentar**en

```
// Kontakten trykket ned
```

Alt fra "//" til linjeskiftet er en *kommentar* og betyder lige præcis ingenting for programmets udførelse. Men for mennesker, som skal **læse** programmet, kan gode kommentarer være meget værdifulde for at kunne forstå programmet.

Værktøj: Betinget udførelse

Eksempel:

```
if (i == 4) {
  digitalWrite(pinLed, HIGH);
} else {
  digitalWrite(pinLed, LOW);
}
```

Hvis variablen `i` har værdien 4, tændes lysdioden, ellers slukkes den.

Forklaring:

`if ( `***A*** ` ) { `***B*** ` } else { `***C*** ` }`

A Et udtryk, som er enten sandt eller falsk

B Kommandoer, som udføres, hvis ***A*** er sandt

C Kommandoer, som udføres, hvis ***A*** er falsk

Hvis ***B*** eller ***C*** kun består af én kommando, kan krølleparenteserne { og } udelades.

Hvis der ikke skal udføres noget, når ***A*** er falsk, kan `else` og `{ `***C*** ` }` udelades

Eksempel 2:

```
if (i==4) digitalWrite(pinLed,HIGH);
```

Hvis variablen `i` har værdien 4, tændes lysdioden.

En anden type kommentarer bruger start- og slut-markeringerne /* og */ . Denne type kan strække sig over flere linjer.

2 – Program med aflæsning af tastetryk

En opgave pr.kontakt

Udfordring 1

Lav programmet om, så kontakten tænder lysdioden, som derefter skal forblive tændt i 2 sekunder, selv om knappen slippes – og derefter slukke.

(Hvis knappen stadig er trykket ned, skal lysdioden tænde igen med det samme.)

Udfordring 2

Tilslut endnu en kontakt til Arduinoen, f.eks. til ben 6.

Tilføj en linje til `setup()` , så dette ben også bliver en indgang med pull-up modstand.

Lysdioden skal ikke længere styres på tid, men skal tændes ved tryk på den ene afbryder og slukkes ved et tryk på den anden.

Flere opgaver til én kontakt

Indtil nu har kontakterne hver haft én – helt afgrænset – opgave. Ofte vil man bruge samme knap til flere opgaver, som f.eks. at steppe ned igennem en række menupunkter, et nyt for hvert tryk.

Vi vil bruge et meget simpelt eksempel: Et tryk på en knap skal tænde lysdioden, det næste tryk skal slukke den – osv. Du vil opdage, at der er flere lærerige detaljer at undersøge undervejs i denne opgave.

Umiddelbart har vi følgende at tænke over:

Hvordan kan programmet vide, om vi er nået til at tænde eller slukke?

Hvordan kan programmet vide, om det er et "nyt" tryk på kontakten og ikke bare et længerevarende?

Første forsøg på at løse opgaven kunne se således ud:

```
const byte pinLed = 13;
const byte pinSwitch = 7;
bool ledState=LOW;

void setup() {
  pinMode(pinLed,OUTPUT);
  pinMode(pinSwitch,INPUT_PULLUP);
}

void loop() {
  while(digitalRead(pinSwitch)==HIGH);

  ledState = !ledState;
  digitalWrite(pinLed,ledState);

  while(digitalRead(pinSwitch)==LOW);
}
```

Programmet holder styr på, om det næste tastetryk skal tænde eller slukke, ved hjælp af en *boolsk variabel* `ledState` (også kaldet en *logisk* variabel). Den starter med at være sat til `LOW` , hvilket er det samme som `false` .

Se værktøjs-boksen øverst til højre.

De to linjer

```
ledState = !ledState;
digitalWrite(pinLed,ledState);
```

bevirker, at variablen `ledState` først ændres til det modsatte, derefter bruges den til at tænde eller slukke lysdioden.

For at holde styr på ét tastetryk ad gangen, rummer programmet to `while` gentagelsesløkker. Se værktøjs-boksen til højre.

Den første løkke

```
while(digitalRead(pinSwitch)==HIGH);
```

bliver ved med at lave – ingenting(!) – *så længe knappen ikke er trykket ned*. Der står ingen kommandoer efter parentesen med betingelsen, kun et semikolon. Det kaldes en tom løkke.

Værktøj: Logisk (Boolsk) variabel

Eksempel:

```
bool itIsSeven;
int j = 5;
itIsSeven = (j == 7);
```

Variablen `itIsSeven` kan have værdierne `false` eller `true`. Da heltalsvariablen `j` er 5, bliver udtrykket `(j == 7)` falsk. Efter tredje linje er værdien af `itIsSeven` derfor `false` .

Boolske variabler kan sammensættes med operatorerne `&&` (*og*), `||` (*eller*), samt `!` (*ikke*) .

Eksempel:

```
ledState = !ledState;
```

Denne linje tildeler variablen `ledState` den *modsatte* værdi af, hvad den havde før.

Værktøj: Gentagelses-løkke med `while`

Eksempel:

```
int sum = 0;
int j = 0;
while (j<11) {
  sum = sum + j
  j++;
}
```

Denne programstump beregner summen af tallene fra 0 til 10.

Forklaring:

`while ( A ) { B }`

A Betingelsen ***A*** udregnes **før hvert** gennemløb af løkken. Hvis udtrykket er sandt, udføres ***B*** . (Ellers fortsættes til næste kommando efter while-løkken)

B En eller flere kommandoer. Hvis der kun er én, kan de krøllede parenteser { } udelades. – Eventuelt kan kommandoen være tom, så der kun står et semikolon.

Så snart betingelsen ikke længere er opfyldt, må det betyde, at knappen er trykket ned. Derefter udføres de tidligere omtalte linjer, som skifter lysdiodens tilstand.

Så når vi til den nederste while-løkke:

```
while(digitalRead(pinSwitch)==LOW);
```

Programmet bliver hængende i denne løkke, *så længe knappen er trykket ned*. Først når knappen slippes igen, går programmet videre med næste tur i `loop()`-funktionen – og ender i den første løkke igen.

Det kan jo nærmest ikke gå galt, vel?

Udfordring 3

Afprøv programmet i praksis!

Kontroller omhyggeligt, at lysdioden skifter for *hvert* tryk. Eller (hvad der er det samme), at lysdioden ender i samme tilstand, *hver gang* du har trykket to gange.

3 – Debouncing

Kontakt-prel

Hvad var resultatet af undersøgelsen i *Udfordring 3*?

Med mindre du er heldig at have nogle meget specielle kontakter, vil du nå frem til, at det fungerer – *sommetider*! Men at der er mange tilfælde, hvor lysdioden f.eks. bare giver et mikroskopisk glimt, hvor den skulle tænde. Andre gange reagerer programmet tilsyneladende ikke.

Du har opdaget fænomenet *kontakt-prel* (engelsk *switch bounce*):
Når du trykker én gang på kontakten, hopper metaldelene inde i kontakten fra hinanden igen et par gange, inden de falder til ro. Og når du slipper, slæber kontaktdelene også lidt mod hinanden, så der kan komme kortvarig kontakt efter den allerførste afbrydelse. Selv om *du* kun trykker én gang, tænder kontakten mange gange.

- Hvad gør man så?

Software debouncing

Der er flere teknikker, man kan anvende. Vi vil lave en helt simpel løsning i software:

Problemet er begrænset til et ganske kort tidsrum efter kontakten er enten trykket ned eller sluppet. Du kan derfor bare bede programmet tage en slapper lige efter hver af de to `while`-løkker. Brug `delay()` til dette.

Udfordring 4

Indsæt de nævnte linjer med `delay()` . Definér en konstant `debounceTime` til at rumme ventetiden for `delay()`-funktionerne. Prøv f.eks. først med 50 millisekunder.

Afprøv programmet.

Prøv at justere på `debounceTime` – hvor lidt kan du nøjes med? Programmet skal stadig fungerer helt stabilt!

Teknikken med at fjerne virkningen af kontakt-prel kaldes på engelsk *debouncing*. Det kan både ske i software og i hardware, hvor man ændrer det elektriske kredsløb.