Kontaktprel / Debounce

Med dette dokument er det forsøgt at skabe lidt klarhed over kontaktprel, hvad det er, og hvordan man kan omgå ulemperne. Der er både gennemgået hardwareløsninger og software-løsninger.

Der bruges ikke delay() som får programmet til at hænge, - men funktionen millis().

Det betyder, at processoren ikke hænger, og derfor kan tjekke en knap – eller flere, - for hver gennemløb i loop().

Rettelser og forslag modtages gerne!!!

Links til dokumentet:

Kontakt-typer, Kontaktprel, Hardwareløsninger, RC-led, RC-led med smittrigger,

FF, AND_Gate, 555,

Software-løsninger, Ting, der skal overvejes, Simpel ON/Off, kode-håndtering af kontaktprel,

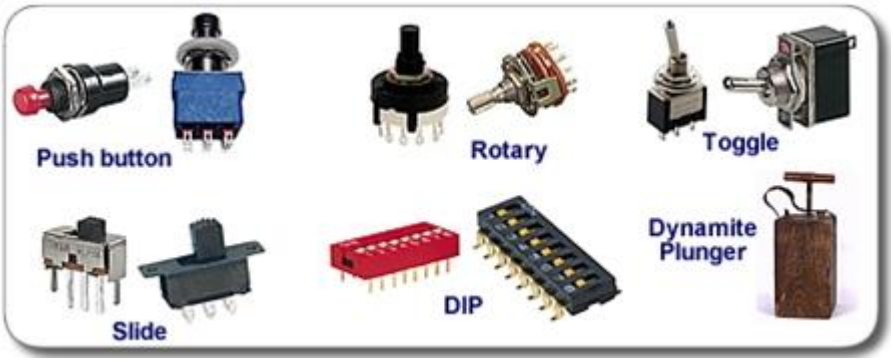
Pseudokode, Kodeeksempler, Pin-interrupts, Debounce flere pins,

Kontakttyper:

Knapper, - eller buttons – eller switch-es fås i forskellige udformninger. Til forskellige formål.

Der findes et hav af komponenter, beregnet til at slutte eller afbryde en strøm.

Her vises et udvalg:



http://www.beavisaudio.com/techpages/Switches/

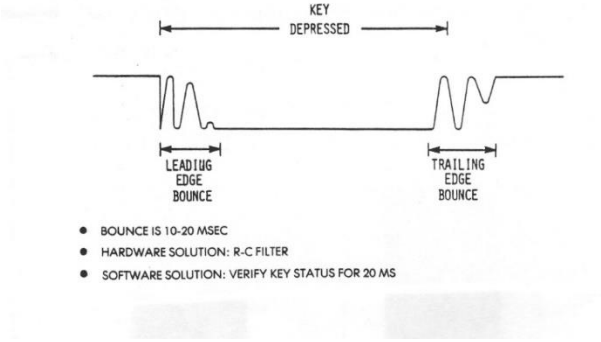
Kontakter kan opdeles efter deres funktion: Se fx mit dok om forskellige kontakttyper her:

Kontaktprel - Contactbounce

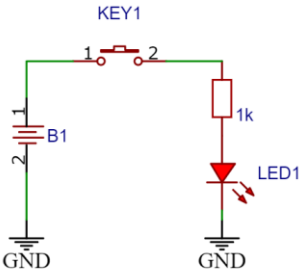
Hver gang en kontakt sluttes eller brydes, eller to ledninger forbindes eller afbrydes, vil der altid opstå kontakt-prel.

Det kan ikke lade sig gøre at to ledninger, - eller kontakter - rører hinanden uden at de først vil hoppe og danse lidt inden de lægger sig til hvile i sluttet tilstand.

Tilsvarende ved afbrydelse af forbindelsen.



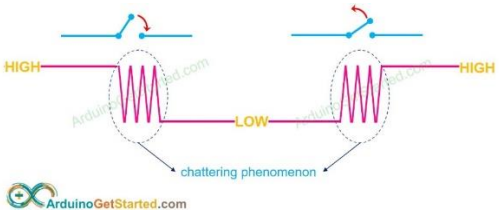
Nogle steder skrives om helt op til 30 millisek. før kontaktprel er døet ud!!



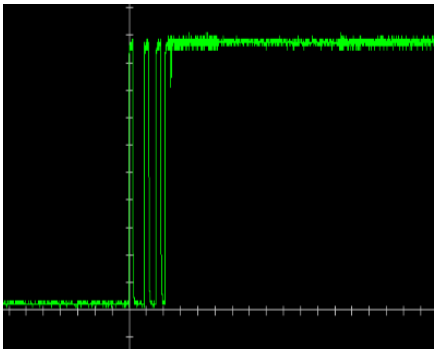
Drejer det sig blot om en simpel switch til at tænde en LED, som vist her, vil det jo ikke ha betydning, om der er lidt kontaktprel.

Men skal en kontakt bruges til at give pulser til en tæller, er det ret vigtigt, at der ikke er kontaktprel. Ellers tælles der mange pulser, hver gang kontakten aktiveres.

Der forekommer også kontaktprel ved afbrydning af en switch



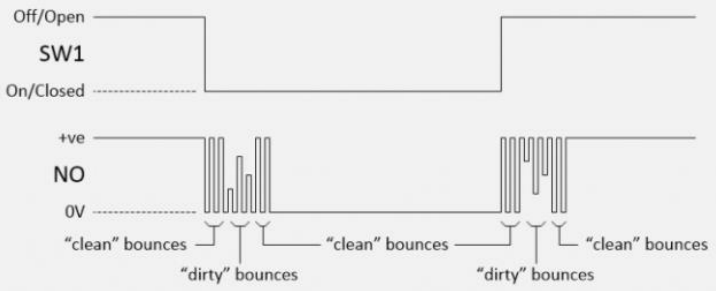
Der mangler en pull up modstand!!



Et scoop-billede af kontakt-prel.

Når en kontakt aktiveres, bringes 2 metal-stykker i kontakt med hinanden. De bevæger sig fysisk fra én position til en anden, og det giver acceleration, og deceleration.

De vil altid – mere eller mindre, - virke ”fjedrende”, dvs. der vil opstå kontakt, derefter afbrydes og gives igen kontakt flere gange, - før der er permanent kontakt.



Ikke alle bounches er nødvendigvis helt fra 0 til 5 Volt eller omvendt !

Derfor:

- Alle kontakter giver prel.
- Varigheden af kontakt-prel-perioden varierer, og varigheden for hver ”kontakt” i prel-perioden varierer også.
- Samme type kontakt giver forskelligt prel.
- Prel varierer afhængig af hvordan brugeren trykker på kontakten.

Se evt. video 1:42 her: der vises kontaktpel, og foreslås en RC-løsning.

Varighed

I forskellige kilder er der angivet, at bounce-tiden kan vare fra 10 til 30 mS. Det er måske afhængig af en kontakts kvalitet?

Og måske er der ikke samme prel-varighed ved slutning og åbningen af en switch??

Preltiden kan også kaldes *'settling time'*

Altså den tid der går, indtil kontakten er faldet til ro.

Hardware løsninger

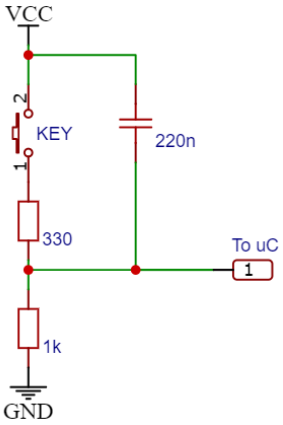
Ved hjælp af lidt ekstra elektronik kan det lade sig gøre at skabe en ”ren” puls.

Der findes forskellige løsninger:

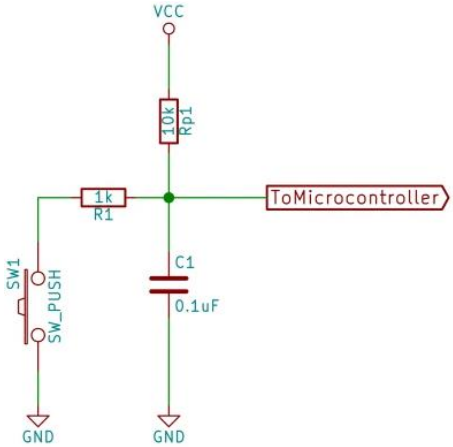
RC-løsninger, uden Smith-trigger

Her ses et eksempel på en – aktiv høj – switch.

Der er monteret en modstand – på 330 ohm – i serie med kontakten, så strømmen i switchen ikke i princippet bliver uendelig stor i det øjeblik, den skaber kontakt første gang. Det kan muligvis ”svejse” kontakten – måske ikke første gang, - men måske i længden!



Forklar kredsløbet !!



Her er switchen aktiv lav.

Forklar også dette kredsløb !!

Repeter lige $t_{1/2\ liv}$

Ovenstående kan udregnes på nogle sider på nettet, fx her:

Forklar !!

High logic level:	2.38 V	
Final voltage:	5.0 V	
Bounce time:	10 ms	<-- CALCULATE TIME
Cap value:	0.1 uF	<-- CALCULATE CAPACITOR
Resistor value:	154736 Ohm	<-- CALCULATE RESISTOR

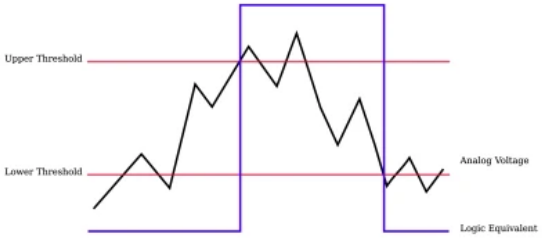
RC-løsning med aktive komponenter, Smith-trigger

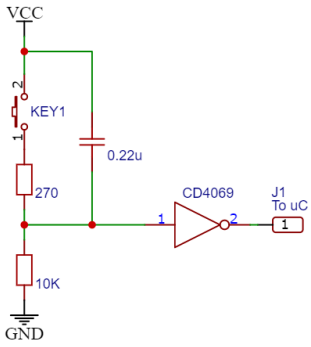
Løsninger med aktive komponenter, dvs. Gates, fx med Smith Trigger. (Hysterese)

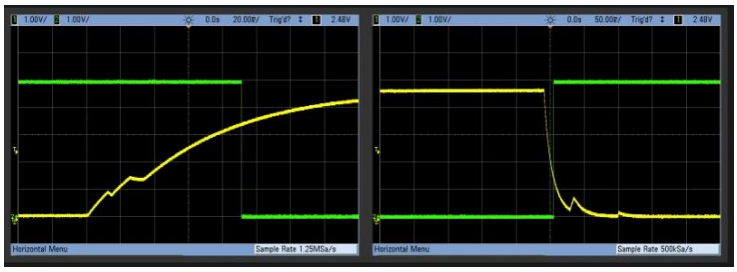
Begrebet Hysterese:

Der er to triggerlevels, Utl og Ltl.

De afgør, om udgangen ”dømmes” høj, eller lav.





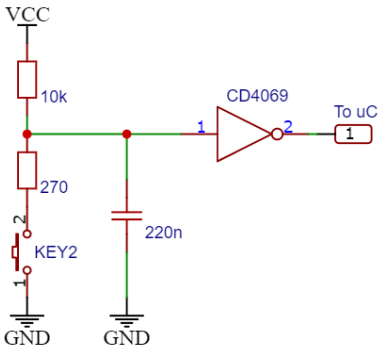


Når kontakten sluttes, kortsluttes kondensatoren. (dog via en lille modstand) Herved bliver spændingen på indgangen høj den første gang ”kontakten danser mod plus”.

Kontakten når at falde til ro i sluttet tilstand før C1 igen er blevet opladet.

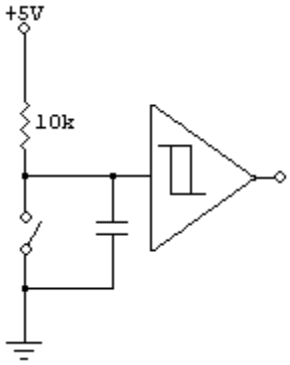
Her fås en positiv puls på udgangen, ellers som ovenfor!

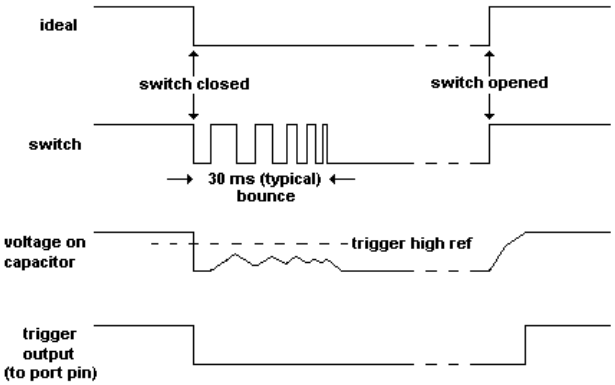
Her er det måske lettere at se, hvordan kondensatoren oplades igen efter kontakten er sluppet !!



Her er vist et eksempel, hvor kondensatorens spænding vises!

Bemærk, at den viste gate har hysterese, men er ikke en inverter!!!

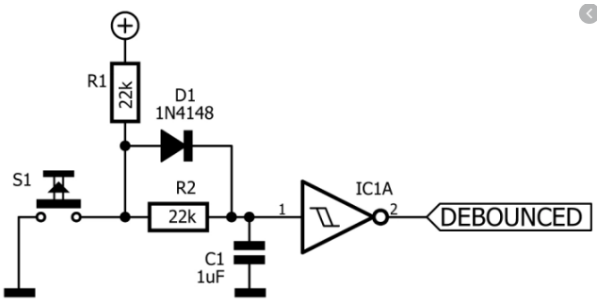




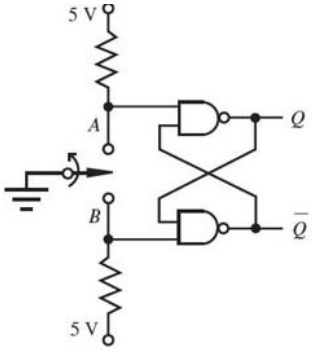
http://www.edsim51.com/8051Notes/interfacing.html#switches

Her et andet diagram:

Forklar kredsløbet !!



Flip-Flop-Løsning



En Flip Flop kan Sættes, og Resettes. Dette kan udnyttes som kontaktprelfjerner !!

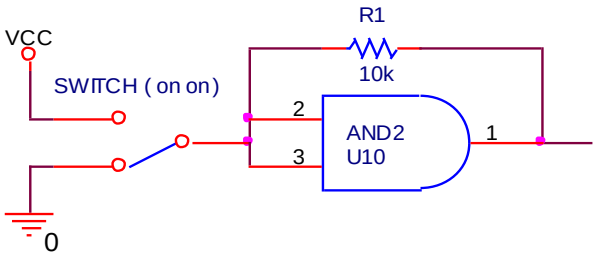
Første gang, skiftekontakten rører "A", bliver "Q" udgangen høj.

Og når kontakten rammer "B" bliver "/Q" høj, og dermed bliver "Q" lav !!.

Lav grafer for A, B og Q

Se også på en Nor-gate-FF.

AND-Gate løsning

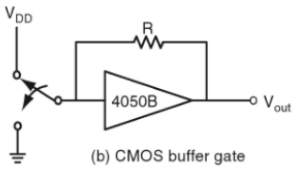


Det samme kan opnås med en ANDgate.

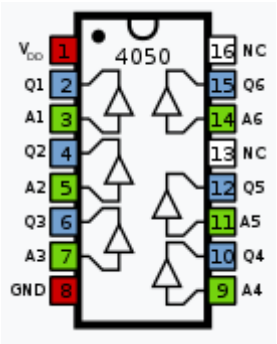
Dette er en ***virkelig smart*** løsning. Med én pakke AND-gate, - en 4081, - kan der laves 4 kontaktprelfjernere.

Men denne løsning kræver en skiftekontakt.:

Eller der kan bruges en 4050, der er vist her:



https://wikidevi.wi-cat.ru/File:4050_Pinout.svg



Pinouts af CD4050

Software-løsninger:

Mange af kontaktprel-problemerne kan også løses ved hjælp af passende kode.

Men kontakter bruges jo til mange formål, - og i mange situationer, og dermed vil koden også være afhængig af hvad programmet skal udføre. – Men også afhængig af om der er kontaktprel eller ej.

Derfor forskellige situationer. Først:

Uden kontaktprel, dvs. ideel puls:

En LED skal lyse så længe der er trykket:

> Læs knap i loop, hvis aktiv tænd LED, ellers sluk LED.

Stuf skal kun udføres 1 gang ved tryk. Kort eller langt tryk:

> Test om et knap-signal er aktiv, - udfør stuf, og husk, det er udført, så det ikke udføres igen før knap-signal er in-aktiv igen.

Simpel ON/Off

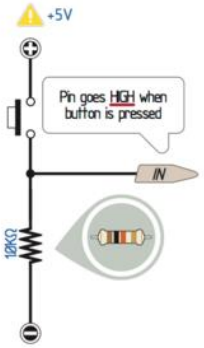
Hvis det "Bare" drejer sig om, at en udgang skal være høj **når knappen er trykket**, og lav når den ikke er det, så er det en simpel program-konstruktion. Her er det lyset, der skal være tændt så længe der er trykket, - og evt. kontaktprel når man ikke at se på en LED.

Pseudokode:

```
Læs kontakten
  hvis knap trykket så:
    udgang = high
 else
    udgang = low
 end if
```

Og med Arduino-kode-eksempel:

```
if (digitalRead(buttonPin)) {
   digitalWrite(ledPin, HIGH);
} else {
   digitalWrite(ledPin, LOW);
}

//  Eller blot skrevet med kun 1 linje:

digitalWrite(ledPin, digitalRead(buttonPin));
```



En action skal kun ske 1 gang ved et tastetryk:

Det er straks værre at håndtere kode, hvis noget **kun skal ske én gang** når en knap trykkes.

Her kan man opdele situationen efter om der er tale om en ideel puls, eller om der er kontaktprel. Først:

Ideel puls:

Når programmet konstaterer et tastetryk, udføres Stuf (på forkanten), og der sættes et flag, der indikerer, at Stuf er udført. Ellers vil det jo blive udført igen ved næste loop.

Når programmet læser at knappen er sluppet, resettes flaget igen, og så er det klar til at agere i en senere loop, hvis der tykkes igen .

Pseudokoden kunne se således ud:

Der bruges her et flag, - dvs. et bit, der bruges som ” huskebit ” til at fortælle, at det er udført

```
Done-flag = 0;


   Hvis ( knap er trykket & done-flag er 0)
       { Do stuf
       Sæt done-flag = 1 }

   Hvis knap ikke trykket & done-flag er 1
       { Sæt done-flag = 0 } // så kan knap trykkes igen
```

Eller som følgende: I hver loop testes knappen, og dens status gemmes:

```
   Hvis forrige knapstatus er lav og nuværende knapstatus er høj
       Do stuf
   End Hvis
   Gem nuværende knapstatus så den bliver den forrige i næste loop
```

Eksempel:

Her tælles en tæller 1 op og en LED skiftes hver gang der trykkes - med ideel puls:

```
const int btnPin = 7;      // pushbutton pin
const int ledPin = 8;      // LED pin

bool currentBtn;           // current state of the LED
bool previousBtn;          // state of the LED from the previous loop
uint8_t count;             // running sum of rising edges

void setup() {
  pinMode(btnPin, INPUT);
  pinMode(ledPin, OUTPUT);
  currentBtn = LOW;
  previousBtn = LOW;
  count = 0;
  Serial.begin(9600);
} // close setup
```

```
void loop() {
  currentBtn = digitalRead(btnPin);

  if ((previousBtn == LOW) && (currentBtn == HIGH)) { // check LOW to HIGH
    count++;
    Serial.println(count);
    // do other stuf
  } // Close if

  previousBtn = currentBtn;
  digitalWrite(ledPin, currentBtn);
} // close loop
```

Kilde: https://reference.digilentinc.com/learn/microprocessor/tutorials/debouncing-via-software/start?s%5b%5d=debouncing

Et eksempel mere på kode beregnet til ideelle pulser:

```
// Toggle LED ved at trykke og slippe trykknap.
// Der er ikke kontaktprel = Bounce

bool lastButtonState = LOW; // = 0
bool ledState = LOW;
bool buttonState = 0;

void setup() {
  pinMode(5, OUTPUT);
  pinMode(4, INPUT);
}

void loop() {
  buttonState = digitalRead(BUTTON_PIN);  // læs knap
  if (buttonState != lastButtonState) { // Kun hvis ændret gås ind i if
    lastButtonState = buttonState; // ændret! - så husk nu-state
    if (buttonState == LOW) {
      ledState = (ledState == HIGH) ? LOW : HIGH; // ( test) ? Hvis sand, brug
denne værdi : ellers denne værdi.
      digitalWrite(LED_PIN, ledState);
    }
  }
   // other stuf Here
}

//Fra <https://roboticsbackend.com/arduino-turn-led-on-and-off-with-button/>
```

Med kontaktprel:

Registrer, at der er trykket på knappen, vent fx 10 ms, og hvis der stadig er trykket så do stuf.
Men der er jo også prel når knappen slippes igen !!

Knap med kontaktprel.

Alle knapper giver kontaktprel. Derfor må koden tage hånd herom. Her er der mange muligheder:

Hvis der bruges en trykknap uden kontaktprel-fjerner, dvs. **med kontaktprel**, fx til at tælle noget, eller til fx at skifte status på en LED, - er det straks meget mere kompliceret at skrive kode.

Problemet med kontaktprel og software er jo, at programafviklingen er så hurtig, at processoren kommer rundt i sit loop rigtig mange gange i sekundet.

Derfor kan den jo i sit loop nå at læse knappen, udføre noget, fx tælle op, og læse knappen igen i næste loop inden knappen lægger sig til ro, eller bliver sluppet igen.

Eller programmet kan reagere på hver bounce !!

Test af kontaktprel i knap:

Her er vist et eksempel på, hvordan man kan teste om der er bounces:

Her er et test-program der bruger pinudløst interrupt til at tælle antal bounces:

```
// Switch Bounce Counter - via pininterrupt.
//
#define btnSTART 2 // Change these pins if you want to use different ones
#define swHIT 3 // External Interrupt 1-pin
#define ledTrigger 4

volatile int bounceCount = 0;

void setup() {
  // setup the switches and LED
  pinMode(btnSTART, INPUT_PULLUP); // aktiv lav, intern pull-up
  pinMode(swHIT, INPUT);  // switch under test
  pinMode(ledTrigger, OUTPUT);

  attachInterrupt(1, bounce, FALLING); // Pin 3
  digitalWrite(ledTrigger, HIGH);  // Turn off the LED
  Serial.begin(9600);   // Start the Serial port
  Serial.println("Arduino Switch Debounce");
  Serial.println();
}  // endsetup

void bounce() {
  bounceCount++;  // adder 1 til Count
}

void loop() {
  Serial.println("Press START button when ready");
  Serial.println("When the LED lights, the test is ready.");
```

```
  Serial.println();
  while (digitalRead(btnSTART)) {} // Wait for Start, until = 0.
  delay(10);
  while (!digitalRead(btnSTART)) {} // vent på, den er sluppet
  delay(1000);
  bounceCount = 0;  // Start the testing
  digitalWrite(ledTrigger, LOW);  // Sluk LED
  Serial.println("Ready for testing...");


  while (bounceCount == 0) {}  // Wait for the switch to close

  // If you are here, the switch was thrown
  // Wait a second to collect the bounces

  delay(1000);

  // Output the results

  digitalWrite(ledTrigger, HIGH);
  Serial.print("The switch bounced ");
  Serial.print(bounceCount);
  Serial.println(" times.");
  Serial.println();
}
```

Opgave:
Test en knap eller bare en ledning til gnd med ovenstående kode.
Der kan fx ændres så man kan nå at trykke på knappen 10 gange og så få vist antal udløste interrupts.

Ovenstående bruger delay-funktionen. Den næste bruger millis().

```
const byte lfPin = 2;  //switch under test conneted to pin 2

unsigned long timer;
volatile unsigned long trans;

void setup()
{
    Serial.begin(9600);
    pinMode(lfPin, INPUT_PULLUP);
    attachInterrupt(0, count, CHANGE);
    timer = millis();
}  // endsetup

void loop()  {
    if(millis() - timer > 1000) // 1 sec to test
    {
        Serial.print(trans);
        trans = 0;
        timer = millis();
```

```
    }
}  // endloop

void count() // Pin Interrupt Function
{
   trans++;
}
```

Med kontaktprel:

Oversigt over ting, der kan / bør overvejes ved kodeskrivning!!

Når man skriver kode, afhænger koden jo af hvad formålet er med programmet. Hvad er vigtigt, og hvor hurtigt skal programmet reagere på et tryk på en knap.

Derfor kan man fx overveje følgende:

- Der kan indføres en debouncetid – eller ”spærretid”, efter den første kontakt er registreret.
- Der kan indføres en debouncetid efter det sidste bounce, dvs. switchen skal være stabil i en tid, før den bliver godtaget.
- Skal der registreres, om knappen stadig er trykket efter spærretiden, - ellers kunne der jo blot være tale om støj, (transient).
- Skal tasten slippes igen før der kan registreres et nyt tastetryk
- Skal der også være debouncetid når knappen slippes.
- Skal det indrettes, så et vedblivende tryk vil gentage ”funktionen” med en given frekvens?
- Og delay() må ikke bruges, for så ”hænger” processoren jo !! Brug i stedet millis().

Så alt efter programmets funktion, må man vælge / skrive sit program til den givne situation.

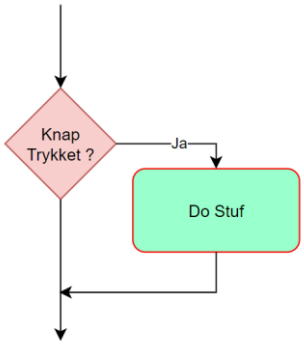
Her følger nogle overvejelser vist vha. flowcharts: Fra enkelt og videreudviklet til mere kompliceret:

Simpel test af knap.

Hvis knappen trykkes, udføres Stuff mange gange på 1 sekund, men afhængig af looptiden.

Også hvis der bare var tale om en spike, dvs. støj !

Altså bør man tage højde for både kontaktprel og evt. om knappen er sluppet igen, før programmet accepterer et nyt tastetryk!



Her udføres bruges et flag (boolean) til at huske, om ” Stuf ” skal udføres.

Hvis flaget er 0, tjekkes knappen. Er den trykket, udføres Stuf, og der hejses et ” Done-flag ”, og der laves et timestamp.

I næste loop er flaget jo ”hejst”, og der tjekkes om en ”debounce-tid” er gået – fx 20 mS.

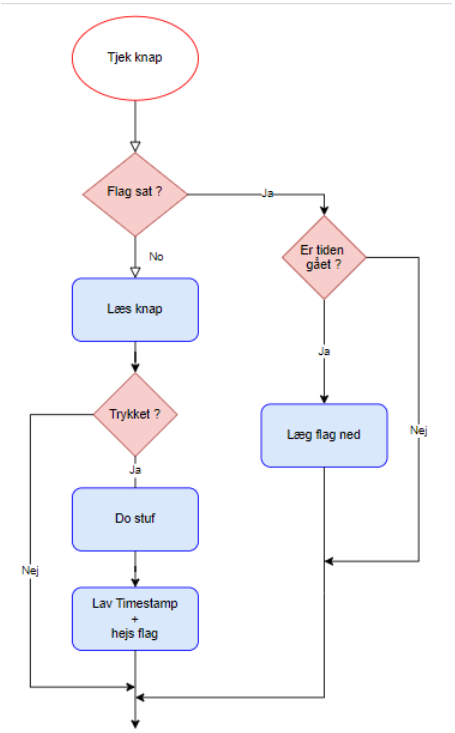
Er tiden gået, lægges flaget ned igen, og i næste loop vil knappen så igen blive tjekket.

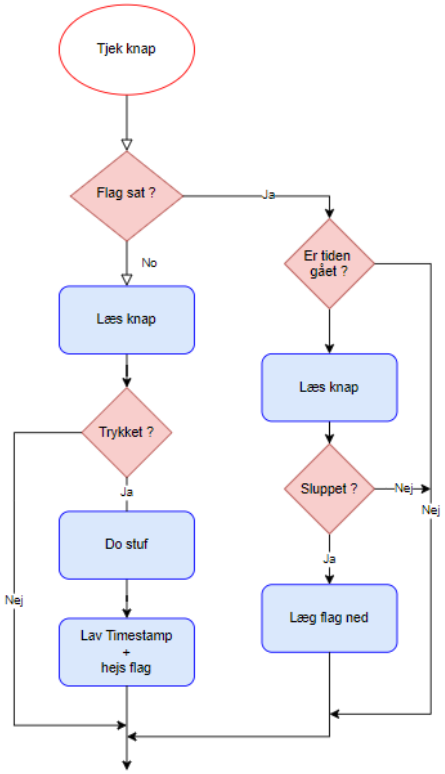
Der bruges millis() så processoren ikke hænger.

Obs:
Holdes knappen trykket, udføres Stuf igen efter debounce-tiden er gået.

Det kan være, det er meningen!! Men måske er det ikke !

Hvad med at tjekke om knappen er sluppet igen før Stuf kan udføres igen?





Her testes både om en tid er gået, og om knappen er sluppet.

Her er der dog stadig problemer med at der jo også er kontaktprel når knappen slippes.

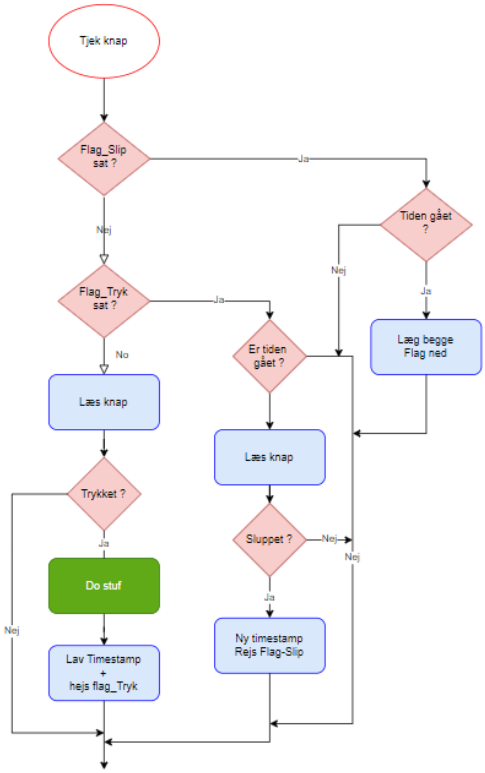
Så kan Stuf jo gentages!!

Så der bør også gå debounce-tid efter der er registreret at knappen er sluppet – før man kan starte forfra

Så kunne det måske se således ud?

Der er brugt 2 flag.

Opgave: Undersøg flowchartet her, og skriv Pseudokode for det!



Husk, at Debounce-tiden kan udmåles vha. følgende:

```
  if ((millis() - lastBounceTimeStamp) > DEBOUNCE_Delay) {   }
```

Debounce-tid resettes efter hver registreret niveau-skift af knap-signal.

Ved denne metode læses knappen i hvert loop. Og hvis knappen er læst til at være modsat det, den blev læst til i forrige loop, gemmes en ny timestamp, dvs. tidsudmålingen starter forfra.

Er der så gået en given tid efter en timestamp, **og** knappen er ON, udføres Stuf.

Dvs. der skal gå en debouncetid **efter sidst registrerede bounce**, - og knappen skal stadig være trykket, - før Stuf udføres.

Det kan fx gøres som følger:

```
  knapStatus = digitalRead(knap_pin);   // læs knap

  if( lastknapStatus != knapStatus ) {  // altså ændret

    Timestamp = millis();              // gem timestamp
    lastknapStatus = knapStatus        // gem nuværende som last i næste loop
```

osv.

Denne metode er brugt i de følgende eksempler:

Obs: Flere af eksemplerne er rimelige ens.

Eks.1

```
/*I dette eksempel tjekkes en inputpin efter en debounce-periode.
Hvis pin-state er ændret, gemmes en timestamp, og hvis pinstate er lav
Udføres Stuf
*/

#define LED_PIN 8
#define BUTTON_PIN 5
byte lastButtonState = LOW;
byte ledState = LOW;
unsigned long debounceDuration = 50; // millis
unsigned long lastTimeButtonStateChanged = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  digitalWrite(BUTTON_PIN, 1); // Intern PullUp
} // End-setup
```

```
void loop() {
  if (millis() - lastTimeButtonStateChanged > debounceDuration) {
    byte buttonState = digitalRead(BUTTON_PIN);
    if (buttonState != lastButtonState) {
      lastTimeButtonStateChanged = millis();
      lastButtonState = buttonState;
      if (buttonState == LOW) {
        ledState = !ledState;
        digitalWrite(LED_PIN, ledState);
      }
    }
  }
}  // End-loop

// Fra <https://roboticsbackend.com/arduino-turn-led-on-and-off-with-button/>
```

Lav flowchart for ovenstående kode!!

Eks.2

```
/* Debounce,      testet 25/3-2020

   Each time the push-button is pressed, the output pin is toggled.
There's a minimum delay between toggles to debounce.
*/

// constants won't change. They're used here to set pin numbers:
const int buttonPin = 8;    // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// Variables will change:
bool ledState = HIGH;         // the current state of the output pin
bool stableButtonState;       // the current stable reading from input pin
bool lastButtonState = LOW;   // the previous reading from the input pin
bool reading = 0;

// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an
int.
unsigned long debounceStartTime = 0;  // last time output pin was toggled
unsigned long debounceDelay = 50;    // debounce time; increase if the output
flickers

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);

  digitalWrite(ledPin, ledState);  // set initial LED state

}  // End-Setup

//----------------------
```

```
void loop() {

  reading = digitalRead(buttonPin); // read switch into a local variable:

  // check to see if you just pressed the button,
  // (i.e. the input went from LOW to HIGH), and you've waited long enough
  // since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:

  if (reading != lastButtonState) {
    debounceStartTime = millis();   // reset debouncing timer
  }
  if ((millis() - debounceStartTime) > debounceDelay) {
  // whatever the reading is at, it's been there for longer than the debounce
  // delay, so take it as the actual current state:

    if (reading != stableButtonState) {   // if the button state has changed:
      stableButtonState = reading;

      if (reading == HIGH) {  // only toggle LED if new buttonst. is HIGH
         // do stuf
        ledState = !ledState;
        digitalWrite(ledPin, ledState);  // set the LED:
      }
    }
  }
  // save the reading. Next time through the loop, it'll be lastButtonState:
  lastButtonState = reading;
}  // End-Loop
```

Eks.3

Kode, der skriver på debugvinduet, hver gang knappen trykkes og slippes.

```
/*
   Kilde:  https://arduinogetstarted.com/tutorials/arduino-button-debounce
   Aktiv lav input med intern pull up på inputpin !!
   Modificeret og testet - 3/10-2020
   / Valle
*/

uint8_t BUTTON_PIN = 7;         // pushbutton pin
uint8_t DEBOUNCE_Delay = 15;    // debounce time; increase if the output
flickers

bool lastStableState = HIGH;       // Sidste stabile tilstand for knappen
bool lastButtonState = HIGH;       // nuværende stabile tilstand
bool currentButtonState = HIGH;    // the current reading from the input pin
```

```
// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an
int.

unsigned long lastBounceTimeStamp = 0;  // timestamp for last time the
inputpin was toggled


void setup() {
  Serial.begin(9600);
  pinMode(BUTTON_PIN, INPUT_PULLUP);  // aktiver intern pull up, aktiv lav!
} // End-Setup

//**********************

void loop() {

// do other stuff in loop

// tjeck button

  currentButtonState = digitalRead(BUTTON_PIN); // read button state:

  if (currentButtonState != lastButtonState) {  // has input changed ?

    lastBounceTimeStamp = millis();             // yes, then save "now" time

    lastButtonState = currentButtonState;       // save the last flickerstate
  }

  if ((millis() - lastBounceTimeStamp) > DEBOUNCE_Delay) {
    // whatever the reading is at, it's been there for longer than the
    // debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (lastStableState == HIGH && currentButtonState == LOW) // Aktive Low !!
      Serial.println("The button is pressed");

    else if (lastStableState == LOW && currentButtonState == HIGH)
      Serial.println("The button is released");

    lastStableState = currentButtonState;    // save the last stable state
  }
  // do other stuff in loop
}
```

Kode kan også ses her: https://learn.adafruit.com/make-it-switch/debouncing

Eks.4

Dette program tæller en variabel op, og skriver den på debugvinduet. Her er læsningen af knappen lagt i en funktion!

```
int inputPin = 7;
```

```
int counter = 0;
int buttonState = 0;
int lastButtonState = 0;

int currentButtonState = 0;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void setup() {
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
}

void loop() {

  readbutton();
}


void readbutton() {
  currentButtonState = digitalRead(inputPin);

  if (currentButtonState != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (currentButtonState != buttonState) {
      buttonState = currentButtonState;
      if (buttonState == LOW) {
        counter++;
        Serial.println(counter);
      }
    }
  }
  lastButtonState = currentButtonState;
}
```

// Modificeret fra https://www.circuitbasics.com/how-to-use-switch-debouncing-on-the-arduino/

Eks.5

```
/*
  Debounce

  Each time the input pin goes from LOW to HIGH, the output pin
  is toggled from LOW to HIGH or HIGH to LOW.
*/

// constants won't change. They're used here to set pin numbers:
const int buttonPin = A1;   // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// Variables will change:
int ledState = HIGH;         // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = HIGH;  // the previous reading from the input pin
```

```
// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an
int.

unsigned long lastDebounceTime = 0;  // last time the output pin was toggled
unsigned long debounceDelay = 50;    // debounce time; increase if output
flickers

void setup() {
  pinMode(buttonPin, INPUT_PULLUP); // Active low
  pinMode(ledPin, OUTPUT);

  digitalWrite(ledPin, ledState); // set initial LED state
}

void loop() {

  int reading = digitalRead(buttonPin); // read switch state to variable:

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited long enough
  // since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    lastDebounceTime = millis();    // reset the debouncing timer

  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer than the
    // debounce delay, so take it as the actual current state:

    if (reading != buttonState) {  // if the button state has changed:
      buttonState = reading;      // save new reading

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  digitalWrite(ledPin, ledState);

  // save the reading. Next time through loop, it'll be lastButtonState:
  lastButtonState = reading;
}
```

// Fra: https://learn.adafruit.com/make-it-switch/debouncing

Eks.6

Program til at tælle op / ned:

Her er et eksempel på et program, der med to knapper tæller en variabel op eller ned.

```
const int inPinUp = 6;
const int inPinDown = 7;
int channel = 1;
int buttonUpState = 0;
int buttonDownState = 0;
int prevBtnUp = LOW;
int prevBtnDwn = LOW;
unsigned long lastBtnUp = 0;
unsigned long lastBtnDwn = 0;
int transInt = 50;

void setup()
{
  Serial.begin(9600);
  pinMode(inPinUp, INPUT);
  pinMode(inPinDown, INPUT);

}

void loop()
{

  buttonUpState = digitalRead(inPinUp);
  buttonDownState = digitalRead(inPinDown);

  if (buttonUpState == HIGH && prevBtnUp == LOW)
  {
    if (millis() - lastBtnUp > transInt)
    {
    channel++;
    if (channel > 9)
    {
      channel = 1;
    }
    lastBtnUp = millis();
    Serial.println(channel);
    }
  }
  prevBtnUp = buttonUpState;

  if (buttonDownState == HIGH && prevBtnDwn == LOW)
  {
    if(millis() - lastBtnDwn > transInt)
    {
    channel--;
    if (channel < 1)
    {
      channel = 9;
    }
    lastBtnDwn = millis();
    Serial.println(channel);

    }
  }
```

```
  prevBtnDwn = buttonDownState;
}
```

Fra: https://forum.arduino.cc/t/up-and-down-counter-with-debounce/26893/4

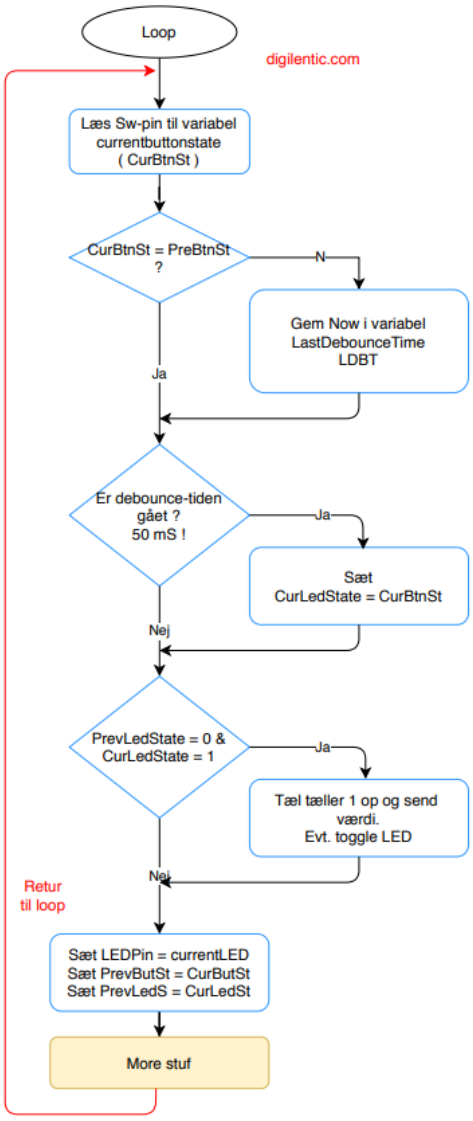
Eks.8

Her følger der et antal gaflede eksempler på debounce-kode:

// https://reference.digilentinc.com/learn/microprocessor/tutorials/debouncing-via-software/start?s[]=debouncing

```
/* Gaflet: På Kilde-siden er der også en demo af kode, der ikke er debounced!

   Testet, OK  d. 30/3-20 / Valle
*/
const int btnPin = 8;                       // Number of the pushbutton pin
const int ledPin =  13;                     // Number of the LED pin

bool currentLedState = 0;    // Current & previous states of output LED pin
bool previousLedState = 0;
bool currentBtnState = 0;    // Current & previous states of input Button pin
bool previousBtnState = 0;

unsigned int count = 0;                     // Rising edge count of LED state
unsigned long debounceStartTime = 0;
unsigned int debounceDelay = 50;                  // Delay time

void setup()
{
  pinMode(btnPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
} // Close Setup

void loop() {

  currentBtnState = digitalRead(btnPin);

  if (currentBtnState != previousBtnState)
  {
    debounceStartTime = millis();  // every time the button state changes,
                                   // get the time of that change
  } // Close If

  if ((millis() - debounceStartTime) > debounceDelay)
  {
    /* if the difference between the last time the button changed is
       Greater than the delay period, it is safe to say the button is in
       the final steady state, so set the LED state to button state. */

    currentLedState = currentBtnState;
  }  // Close If
```

```
  // start functional code, verification code, and a button press counter

  if ((previousLedState == LOW) && (currentLedState == HIGH)) {
    //count rising edges
    count++;
    Serial.println(count);
    digitalWrite(ledPin, currentLedState);
    // Do Stuf
  } // Close if

  // ******************** end functional code *********

  // set current states to previous states

  previousBtnState = currentBtnState;
  previousLedState = currentLedState;
}  // Close Loop

// Flowchart, se flg.:
```



Pseudokode: (reference.digilentinc)
(alle værdier starter lave)

Læs knapstatus og gem i variable CurrentBtnState

Hvis
den nye knapstatus er forskellig fra forrige knapstatus
så gem debounce-starttid.
End-Hvis

Hvis
debouncetiden er gået (now – starttid > deboundedelay)
så sæt nuværende Ledstatus = nuværende Knapstatus
end-Hvis

Hvis
forrige Ledstatus er lav og nuværende Ledstatus er høj
så tæl tæller op
print tæller
Måske mere stuf ??
endHvis

send nuværende Ledstatus til Ledpin.
sæt forrigeknapværdi = Nuværende-knapværdi
sæt forrige ledstatus = Nuværende Ledstatus

endLoop

Eks.9

Eksempel på debouncing af flere pins:

Kilde: https://forum.arduino.cc/t/debouncing-multiple-buttons-with-arrays-sample-for-review/499457

```
/*
  Debounce Multiple Buttons with Arrays
  Pushbuttons attached to pins 3,4, ... active Low
  Modificeret & testet af Valle, 25/02-2023
*/

byte buttons[] = { 3, 4, 5, 10 };  // pin numbers of the buttons that we'll use
#define NUMBUTTONS sizeof(buttons)
int buttonState[NUMBUTTONS];
int lastButtonState[NUMBUTTONS];
boolean buttonIsPressed[NUMBUTTONS];

long lastDebounceTime = 0;  // the last time output pin was toggled
long debounceDelay = 50;    // the debounce time; increase if output flickers

//*****

void setup() {
  Serial.begin(9600);
  // define pins:
  for (int i = 0; i < (NUMBUTTONS); i++) {
    pinMode(buttons[i], INPUT);
    digitalWrite(buttons[i], 1);
    lastButtonState[i] = LOW;
    buttonIsPressed[i] = false;
  }
}  // EndSetup

// *****

void loop() {
  check_buttons();
  action();
  // other stuff
}  // EndLoop
```

```
// *****

void check_buttons() {
  for (int x = 0; x < NUMBUTTONS; x++) {
    //
    int reading = digitalRead(buttons[x]);  // læs pin state
    if (reading != lastButtonState[x]) { lastDebounceTime = millis(); }
    // state ændret, så gem timestamp
    if ((millis() - lastDebounceTime) > debounceDelay) {  // if timed out
      if (reading != buttonState[x]) {                    // State er ændret
        buttonState[x] = reading;
        if (buttonState[x] == LOW) {  // knap trykket, aktiv lav !!!
          buttonIsPressed[x] = true;  // set flag to action function
        }
      }
    }
    lastButtonState[x] = reading;  // save reading.  Next time
                                   // through the loop, it'll be the lastButtonState:
  }
}  // EndLoop

//--------

void action() {
  for (int x = 0; x < NUMBUTTONS; x++) {
    if (buttonIsPressed[x]) {
      Serial.print("button ");
      Serial.print(buttons[x]);

      switch (x) {
        case 0:
          Serial.println(" Case 0 ");
          // other action
          break;

        case 1:
          Serial.println(" Case 1 ");
          break;

        case 2:
          Serial.println(" Case 2 ");
          break;

        case 3:
          Serial.println(" Case 3 ");
          break;
```

```
        case 4:
          Serial.println(" Case 4 ");
          break;

        case 5:
          Serial.println(" Case 5 ");
          break;
        // Add more !!
      }

      buttonIsPressed[x] = false;  //reset the button
    }
  }
}  // End-Action
```

Videoer: https://www.youtube.com/watch?v=DfKAwrBievM&ab_channel=RoboticsBack-End

https://www.youtube.com/watch?v=80Me9BqQuEs&ab_channel=HandsOnEngineering
(vist med gentagelse, hvis der fortsat trykkes !)

Kilder:

Et eksempel mere på flere buttons debounce, se:
http://ediy.com.my/tutorials/item/96-debouncing-multiple-switches

Debounce via bibliotek: https://www.pjrc.com/teensy/td_libs_Bounce.html

Se mere: https://www.arduino.cc/en/Tutorial/Debounce

Bibliotek: https://github.com/wkoch/Debounce

Kort / vs. Lang tryk: http://markus-wobisch.blogspot.com/2018/04/push-buttons-debounce-and-short-vs-long.html

Se: https://www.instructables.com/Arduino-Push-Switch-Debouncing-Interrupts/

Se https://forum.arduino.cc/index.php?topic=128056.0

Se fx også: https://www.instructables.com/Arduino-Software-debouncing-in-interrupt-function/

Se youtube med Jeremy Blum 19:53: https://www.jeremyblum.com/2011/03/07/arduino-tutorial-10-interrupts-and-hardware-debouncing/

Bonus Info:

Rotary-encoder: http://averageek.com/posts/rotary-encoder-gray-code-og-kontaktprel/