Geigertæller

Arduino-eksperiment	130410	*Stikord*	Interfacing til GM sensor, hardware tæller, interrupts	
Version	2018-07-10 / HS	*Niveau*	Videregående	*p.* 1/4

Det lærer du:

- Forbind en Frederiksen GM-sensor til Arduino
- Brug Arduinos indbyggede 16-bit tæller
- Brug tællerens overflow-interrupt

Du skal bruge følgende ud over det mest almindelige Arduino-tilbehør:

- En modularbøsning til breadboards (Frederiksens 663010 rummer 2 af disse samt tilhørende kabler)
- En GM-sensor med modularbøsning (513575 – eller 512515 sammen med 512585). Kablet med Jack-stikket afmonteres (og sættes på igen, når du er færdig med at bruge GM-sensoren).
- Et krydset modular-modular-kabel (medfølger bl.a. 663010 – og kan desuden fås som 197571).
- Evt. et 16x2 LCD-display

1 – Forbind modularbøsningen til Arduino

Som vist nedenfor anvendes ben 3, 4 og 5 på modularbøsningen.

Ben 3 forbindes til 0 V (GND) på Arduinoen, ben 5 forbindes til 5 V på Arduinoen. Ben 4 forbindes til **Arduino ben 5**. NB: I modsætning til mange tidligere eksperimenter, er der **ikke** frit valg for denne forbindelse.

Fortsæt med at tilslutte GM-sensor 513575 med et krydset modular-modular kabel. Anvendes i stedet kombinationen 512515 og 512585, sættes BNC-stikket fra GM-røret ligeledes i bøsningen på adapteren.

513575 (eller 512585) har en lysdiode, som markerer, at der er spænding til stede – check, at den lyser.

Hvis lyden er slået til på 513875 (eller 512585), vil du kunne høre et bip i ny og næ fra baggrundsstrålingen. Det viser, at Geigerrøret fungerer.

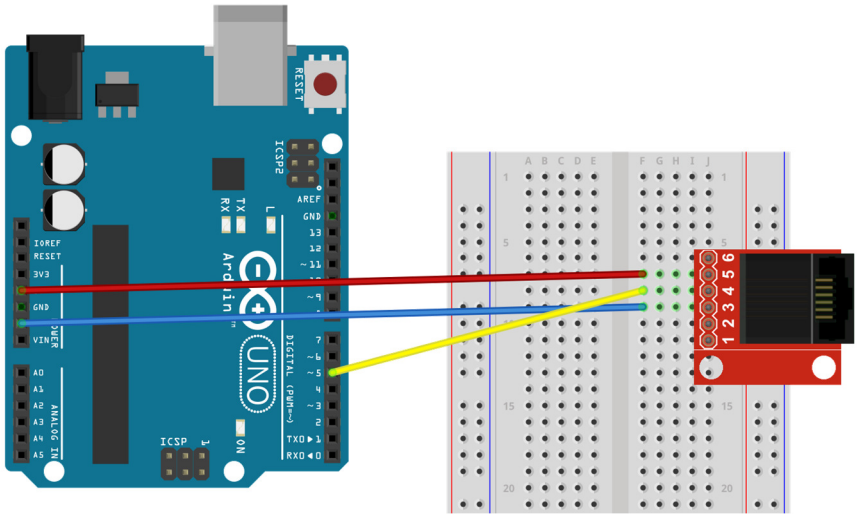
2 – Simpelt check

Indtast følgende:

```
void setup() {
  pinMode(5,INPUT);
  pinMode(13,OUTPUT);
}

unsigned long last;

void loop() {
  if (digitalRead(5)) {
    last=millis();
    digitalWrite(13,HIGH);
  } else {
    if (millis()-last>50) digitalWrite(13,LOW);
  }
}
```



Ovenstående program anvender den indbyggede lysdiode på ben 13 til at blinke, hver gang der modtages en impuls på ben 5. Der tændes, så snart ben 5 er høj, og der slukkes 50 ms efter ben 5 går lav – med mindre der er ankommet en ny impuls i mellemtiden.

Sæt lyden til, så GM-udstyret bipper, og hold øje med, at lysdioden giver et blink for hvert bip. (Hvis du har en radioaktiv kilde til rådighed, skal den være så langt fra Geiger-Müller-røret, at de enkelte impulser registreres enkeltvis med passende mellemrum, f.eks. ca. et sekund – ellers kan du ikke se, hvad der foregår.)

OK – hvis det fungerer, er hardwaren i orden.

Følgende udfordring er mest ment som optakt til næste afsnit og kan evt. overspringes.

Udfordring 1

Prøv, om du kan lave programmet om til en Geigertæller, som tæller antallet af impulser i 10 sekunder. Resultatet skal udlæses på PC'en i monitor-vinduet. Efter hver måleperiode skal tælleren automatisk starte forfra fra 0.

Tips:

Du må kun tælle en ny impuls, når indgang 5 **har** været lav siden sidst. Brug f.eks. en logisk variabel `ready` til at holde styr på det.

Husk at starte den serielle port et sted i `setup()`.

Har du adgang til en radioaktiv kilde og en "rigtig" Geigertæller, som kan tilsluttes det GM-rør, du bruger her, kan du kontrollere, at tælletallene for de to opstillinger i gennemsnit ligger nogenlunde tæt på hinanden. Radioaktive henfald sker tilfældigt, så de enkelte målinger vil altid svinge, men placerer du kilde og GM-rør stabilt i forhold til hinanden, har du gjort, hvad du kan, for at gøre de to situationer sammenlignelige.

3 – Hardware-tællere

Impulserne fra Geiger-sensoren har en varighed, som skal måles i "ganske mange mikrosekunder" – og det er faktisk nødvendigt, for at Arduinoen kan nå at tælle dem alle. Hvis programmet bliver større – og måske ligefrem kalder nogle indbyggede funktioner, som tager lang tid inden de returnerer – så risikerer vi, at tællinger går tabt. Det samme vil være tilfældet, hvis man vil tælle impulser, som har en meget kort varighed. Hvad gør man så?

Ved at bruge en af Arduinoens indbyggede *tællere* kommer vi helt udenom problemet med at programmet kan have travlt med noget andet end at tælle.

Arduino råder over tre tællere (også kaldet timere). Den første – Timer 0 – bruges internt i systemet og kan ikke bruges universelt. Timer 0 og Timer 2 er begge 8-bit tællere, så de kan kun tælle til 255, men Timer 1 er en 16 bit tæller og kan derfor tælle op til 65535. Den vil vi bruge til formålet. Timer 1 kan programmeres til at få sine tælleimpulser fra ben 5 på Arduinoen – derfor dette valg af indgangsben.

Timerne i Arduinoen kan programmeres til utroligt mange opgaver ved at tildele nogle *registre* specielle talværdier. *Registre* kan i denne sammenhæng opfattes som variabler med særlige navne, hvor hver enkelt bit styrer en del af timerens opførsel.

Ved at skrive følgende to linjer i `setup()` får vi Timer 1 til at tælle én op, hver gang indgang 5 går fra lav til høj:

```
TCCR1A = 0;         // Timer1, normal mode
TCCR1B = 7;         // External clock on T1  (= ben 5), rising edge
```

Som sædvanlig er kommentarerne med til at gøre programmet (lidt mere) læseligt – også efter et par uger.

Tællerens værdi fremgår af registeret `TCNT1`, som kan både skrives og læses. Følgende linje nulstiller tælleren:

```
TCNT1 = 0;
```

… og følgende instruktion overfører tællerens øjeblikkelige værdi til en variabel (f.eks. af typen `word`):

```
counts = TCNT1;
```

De fleste af detaljer om tællerne i Arduinoen kan klemmes ned på 90 sider i manualen for microcontrolleren – men man kan finde en del gode og mere overskuelige eksempler på nettet, hvis man har en konkret opgave. *Lige nu* har du ikke brug for flere oplysninger, end der står her i afsnit 3 og 4.

Udfordring 2

Skriv et nyt program med de nævnte linjer i `setup()`, og skriv desuden to funktioner `start()` og `stop()`, som gør følgende:

`start()` skal gemme starttiden i en global variabel og nulstille tælle-registeret.
`stop()` skal gemme tælle-registerets værdi i en global variabel.
Du vil også få brug for en logisk variabel til at holde styr på, om en tælling er i gang eller ej.

Nede i `loop()`-funktionen undersøges, om tællingen er i gang:

– hvis den ***ikke er*** i gang, kaldes `start()`-funktionen

– hvis den ***er*** i gang, kontrolleres tidsforløbet. Så snart der er gået 10 sekunder, kaldes `stop()`-funktionen og umiddelbart derefter udskrives måleresultatet.

Logikken i loop() vil få tælleren til et starte en ny måling, så snart den er færdig med den foregående. (Men det er nemt at udvide med ny funktionalitet om lidt.)

Vi kan nu udvide opstillingen lidt. Forbind en kontakt, som skal bruges til start-knap. Vælg selv et passende ben på Arduinoen.

Det er nok en ide at vende tilbage til modul 130115 Tænd med en kontakt, hvis du er det mindste i tvivl om noget.

Udfordring 3

Ret programmet fra før, så kontakten kan aflæses. Husk at sætte indgangen til at bruge pull-up.

I `loop()`-funktionen skal der ikke længere startes med det samme. I stedet ventes, til der trykkes på knappen. Derefter skal tællingen ske i 10 sekunder og resultatet printes i monitorvinduet.

Udfordring 4

Ret programmet fra før; kontakten skal nu både fungere som start- og stopknap, dvs. der måles ikke på nogen fast tid. Det vil også sige, at du får brug for at lave debouncing på kontakten.

Her er det fantastiske ved hardware-tællere: Uanset om du laver primitiv debouncing med kald af `delay()`, så arbejder tælleren videre, mens processoren i øvrigt "sover"! Husk blot at kalde `start()` hhv. `stop()`, **inden** du begynder på debouncing.

Efter tryk på stop udlæses både måletid i sekunder med decimaler og tælletallet.

Lad programmet skrive "mellemresultater" ud for hvert sekund, mens der tælles.

4 – Interrupts

Hvad sker der, hvis programmet bliver ved med at tælle i lang tid? Tælleren kan jo ikke rumme tal, der er større end 65535! Har du en kraftig kilde og tid, kan du jo prøve – det er rart at få "mellemresultater" skrevet ud undervejs, ellers aner du ikke, hvor du er henne. Prøv!

Uanset hvad du observerede ovenfor, vil resultatet ikke længere være korrekt, når impuls nr. 65536 skal tælles. Vi har brug for at få denne overflow-situation markeret.

Vi vil her lige akkurat "snuse" til et meget stærkt værktøj, som i dette tilfælde gør løsningen ultra-simpel. Vi kan programmere tælleren til at udløse et såkaldt *interrupt*, når tælleren ryger ovenud af skalaen.

Et interrupt betyder: 1) den normale programafvikling afbrydes – 2) en speciel *interrupt-rutine* køres – 3) til sidst hopper vi ind i programmet igen på præcis samme sted, som vi forlod det.

Interrupt-rutinen skal sætte en logisk variabel, som markerer tilstanden ”overflow”, hovedprogrammet kan derefter tjekke denne variabel og reagere passende.

Skriv først denne programstump (f.eks. inden linjerne med `setup()`):

```
volatile bool Overflow=false;   // Der er ikke overflow i starten

ISR(TIMER1_OVF_vect) {
  Overflow=true;                // Der er sket overflow: Sæt en markør
}
```

Først erklæres den logiske variabel. Den særlige betegnelse `volatile` er et signal til compileren om at undlade at gætte på, hvordan og hvornår variablen ændres. (Nødvendigt, når variablen bruges i en interrupt-rutine.)

Derefter kommer noget, der ligner en almindelig funktion – men skinnet bedrager:
ISR betyder *Interrupt Service Routine* – det vil sige, at dette er en funktion, som kaldes, når der sker et bestemt interrupt.
TIMER1_OVF_vect udpeger den præcise type interrupt – i dette tilfælde dén, som skyldes, at TIMER 1 har laver overflow.

Hvad der står mellem de **{** krøllede parenteser **}**, bestemmer vi selv – det må bare ikke være noget, der tager ret lang tid at udføre.

For at programmere tælleren til at udløse dette interrupt ved overflow skal endnu et register klargøres i `setup()`:

```
TIMSK1 = 1;       // Interrupt ved Timer 1 overflow
```

Så er der bare tilbage at tilføje et check på værdien af `Overflow` nede i `loop()`-funktionen. Og i øvrigt huske at sætte variablen falsk igen ved start på en ny måleperiode – i funktionen `start()`. Hvis variablen er sand, skal dette naturligvis give en reaktion i udskriften.

Udfordring 5

Gør det. Og prøv det evt. af, hvis du har en kraftig kilde (eller god tålmodighed).

Udfordring 6

Flyt udskrivningen til at foregå på et 16x2 LCD-display.

Så kan Geigertælleren også fungere uafhængigt af PC’en. Du skal blot have et batteri sluttet til Arduinoen, så er du mobil. (For så vidt du kan flytte opstillingen, uden den skvatter fra hinanden.)

Du har nu afprøvet forskellige muligheder – og det er jo ikke sikkert, at den med start/stop-knappen er din foretrukne version. Ret evt. programmet igen, hvis du hellere vil lave automatisk gentagne målinger.

Tip: Du kan få en batteriklips (til 9 V batteri), forbundet til et DC stik, som passer i Power-stikket på en Arduinoen Uno. – Smart til opgaver væk fra PC’en. Frederiksens varenummer 663810 er en håndfuld (5 stk.) af disse.

Udfordring 7

I stedet for blot at sætte en logisk variabel, kunne interrupt-rutinen tælle en heltalsvariabel op – og dermed udvide tællerens talområde.

Variablen til denne overflowtæller skal være global og erklæres `volatile unsigned long`.

Typen for det samlede tælletal skal ændres fra at være `word` til at være `unsigned long`.

Lav disse ændringer og tilføj følgende:

I `start()` skal overflowtælleren nulstilles sammen med tælleregisteret.

I `stop()` skal resultatet nu sammensættes som 65536 * overflowtælleren + tælleregisteret.