Geigerteller

Arduino-experiment	130410-EN	*Trefwoorden*	Interfacing met GM-sensor, hardwareteller, interrupts		
Versie	2018-07-10 / HS	*Niveau*	Gevorderd	*p.*	1/5

Wat je leert:

- Sluit een Frederiksen GM-sensor aan op Arduino
- Gebruik Arduino's ingebouwde 16-bit teller
- Gebruik de overflow interrupt van de teller

Naast de gebruikelijke Arduino accessoires heb je het volgende nodig:

- Een modulaire contactdoos voor breadboards (Frederiksen's 663010 bevat 2 van deze, plus geschikte kabels)
- Een GM-sensor met modulaire aansluiting (513575 – of 512515 met 512585). Verwijder de kabel met de klinkstekker (en stop hem weer terug als je klaar bent met de GM-sensor)
- Een gekruiste modulaire naar modulaire kabel (meegeleverd met bijv. 663010 – ook verkrijgbaar als 197571).
- Optioneel een 16x2 LCD-display

1 – Sluit de modulaire contactdoos aan op Arduino

Zoals hieronder getoond, worden pin 3, 4 en 5 op de modulaire contactdoos gebruikt.

Pin 3 sluit aan op 0 V (GND) op de Arduino, pin 5 sluit aan op 5 V op de Arduino. Pin 4 sluit aan op **Arduino pin 5**. Let op: in tegenstelling tot veel vorige experimenten zijn er **geen** alternatieven voor deze laatste verbinding.

Sluit de GM-sensor 513575 aan met een gekruiste modulaire naar modulaire kabel. Als je de combinatie van 512515 en 512585 gebruikt, steek dan ook de BNC-stekker van de GM-buis in het contact van de adapter.

513575 (of 512585) heeft een LED om aan te geven of de stroom is ingeschakeld – controleer of deze brandt.

Als je het geluid op 513875 (of 512585) inschakelt, hoor je af en toe een pieptoon van de achtergrondstraling. Dit betekent dat de geigerbuis werkt.

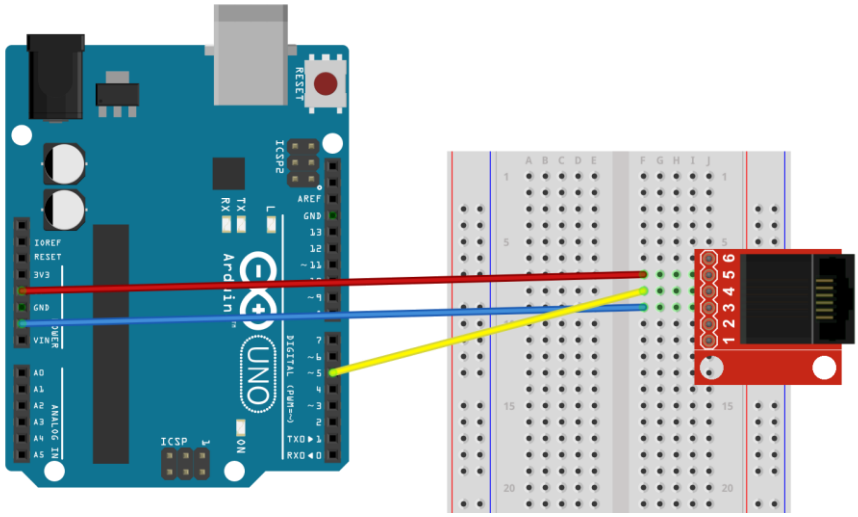
2 – Een eenvoudige controle

Voer het volgende in:

```
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);
  }
}
```



Het bovenstaande programma gebruikt de ingebouwde LED op pin 13 om het elke keer aan te geven als een puls wordt ontvangen op pin 5. Deze gaat hoog zodra pin 5 hoog is, en schakelt 50 ms nadat pin 5 laag gaat uit – tenzij er in de tussentijd een nieuwe puls is aangekomen.

Zet het geluid aan om de GM-apparatuur te laten piepen en houd de LED in de gaten om te controleren of deze bij elke pieptoon knippert. (Als je een radioactieve bron gebruikt, houd dan voldoende afstand om individuele pulsen individueel te laten opvallen met ongeveer een seconde ertussen – anders kun je niet zien wat er gebeurt)

OK – als dit werkt, is de hardware er klaar voor.

Het volgende is bedoeld als voorproefje op de volgende sectie en kan eventueel worden overgeslagen.

Uitdaging 1

Probeer het programma te herschrijven om een geigerteller te maken die het aantal pulsen in 10 seconden telt. Het resultaat wordt uitgevoerd op de PC in het monitorvenster. Na elke telperiode moet de teller opnieuw beginnen vanaf 0.

Tip:

Een puls mag pas worden geteld **nadat** pin 5 na de vorige puls laag is geweest. Gebruik bijvoorbeeld een Booleaanse variabele `ready` om dit bij te houden.

Vergeet niet de seriële poort te initialiseren in `setup()`.

Als je toegang hebt tot een radioactieve bron en een “echte” geigerteller die dezelfde GM-buis kan gebruiken die je hier gebruikt, kun je controleren of de gemiddelde tellingen in de twee opstellingen relatief dicht bij elkaar liggen. Radioactief verval is een willekeurige gebeurtenis, dus individuele metingen zullen altijd variëren. Maar als je de bron en de GM buis stabiel plaatst (met behulp van geschikt opstelmateriaal) heb je alles gedaan om de twee situaties vergelijkbaar te maken.

3 – Hardwaretellers

De pulsen van de Geiger sensor hebben een duur die in “heel wat microseconden” wordt gemeten – en dat is eigenlijk nodig als de Arduino ze allemaal zou moeten kunnen tellen. Als het programma complexer wordt – of misschien gebruikt het ingebouwde functies die lang duren om uit te voeren – lopen we het risico dat we een aantal van de tellingen missen. Hetzelfde gebeurt als we pulsen met een nog kortere duur willen meten. Hoe gaan we hier mee om?

Door gebruik te maken van een van de ingebouwde *hardware tellers* van de Arduino kunnen we het probleem waarbij het programma bezig is met andere zaken dan het tellen, volledig omzeilen.

Arduino heeft drie tellers (ook wel timers genoemd). De eerste – Timer 0 – wordt intern in het systeem gebruikt en kan niet universeel worden gebruikt. Timer 0 en Timer 2 zijn beide 8-bit tellers, dus ze kunnen slechts tot 255 tellen, maar Timer 1 is een 16-bit teller en kan dus tot 65535 tellen. Die laatste is dus geschikt voor ons doel. Timer 1 kan worden geprogrammeerd om de Arduino pin 5 te accepteren als teller-ingang – vandaar deze keuze van de input pin.

De Arduino timers kunnen worden geprogrammeerd voor ongelooflijk veel taken door een aantal *registers* specifieke waarden te geven. In deze context kunnen *registers* worden beschouwd als variabelen met speciale namen waarbij elke individuele bit een deel van het gedrag van de timer bestuurt.

Door de volgende twee lijnen in `setup()` te plaatsen, zorgen we dat Timer 1 telkens met 1 omhoog gaat elke keer dat pin 5 van laag naar hoog gaat:

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

Zoals gebruikelijk helpen de opmerkingen om het programma (iets beter) leesbaar te maken – zelfs na een paar weken.

De waarde van de teller wordt bijgehouden in het register `TCNT1` dat zowel lezen als schrijven toelaat. Dit zal de teller op nul zetten:

```
TCNT1 = 0;
```

… en de volgende instructie zet het werkelijke aantal tellingen om naar een variabele (bv. van het type `word`):

```
counts = TCNT1;
```

De meeste details over de Arduino tellers kunnen op 90 pagina's in de handleiding van de microcontroller worden uitgelegd – maar op het internet zijn ook veel goede en illustratieve voorbeelden te vinden als je een specifieke behoefte hebt. *Op dit moment* heb je geen andere informatie nodig dan wat hier in hoofdstuk 3 en 4 staat.

Uitdaging 2

Schrijf een nieuw programma met de in `setup()` genoemde regels, en schrijf daarnaast twee functies `start()` en `stop()`, die zich als volgt gedragen:

`start()` moet de starttijd in een globale variabele opslaan en het telregister op nul zetten.
`stop()` moet het telregister opslaan in een globale variabele.
Je hebt ook een Booleaanse variabele nodig om bij te houden of er geteld wordt of niet.

Onderaan in de `loop()` functie moet je controleren of het tellen bezig is:

– als de teller **niet** loopt, roep dan de `start()` functie aan

– als de timer **loopt**, controleer dan de gebruikte tijd. Zodra er 10 seconden zijn verstreken, roep je de `stop()` functie aan en onmiddellijk daarna moet het resultaat worden weergegeven.

De loop () logica zorgt ervoor dat de teller een nieuwe meting start zodra de vorige is afgelopen. (Maar dit is gemakkelijk uit te breiden met wat nieuwe functionaliteit.)

We gaan nu de opstelling wat verbeteren. Sluit een schakelaar aan die we als startknop kunnen gebruiken. Kies een geschikte pin op de Arduino.

Als er ook maar enige twijfel bestaat kan het een goed idee zijn om terug te keren naar module 130115 met behulp van een schakelaar.

Uitdaging 3

Bewerk het programma om de schakelaar te kunnen lezen. Vergeet niet om de input pull-up te laten gebruiken.

In de `loop()` functie willen we niet meer meteen beginnen. Wacht in plaats daarvan tot de startknop wordt ingedrukt. Tel dan 10 seconden en geef het resultaat weer zoals voorheen.

Uitdaging 4

Bewerk het programma opnieuw; nu moet de schakelaar zowel als start- als stopknop functioneren, de meetperiode staat dus niet meer vast. Dit betekent ook dat je switch debouncing moet uitvoeren.

Hier is een geweldig punt wat betreft de hardwaretellers: het maakt niet uit of je een primitieve debouncing maakt door `delay()` aan te roepen, de teller blijft werken terwijl de hoofdprocessor “slaapt”! Vergeet niet om `start()` of `stop()` aan te roepen, **voordat** je de debouncing start.

Wanneer er op stop wordt gedrukt, worden zowel de meetperiode (in seconden met decimalen) als de telling weergegeven.

Laat het programma elke seconde tijdens het tellen “lopende” tellingen uitvoeren.

4 – Onderbrekingen

Wat gebeurt er als het programma heel lang blijft tellen? De teller kan slechts nummers tot 65535 opslaan! Als je een voldoende sterke bron en genoeg tijd hebt om te besteden, kun je het proberen – hier heb je de “lopende” tellingen nodig om de voortgang te kunnen volgen. Probeer maar!

Ongeacht de uitkomst van bovenstaande test kan het resultaat niet meer geldig zijn zodra puls nr. 65536 is geteld. We hebben daarom een indicatie nodig wanneer zo'n overflow zich voordoet.

We zullen nu heel even kijken naar een zeer sterk instrument dat hier gemakkelijk voor zorgt. We gaan de teller programmeren om een zogenaamde *interrupt* te activeren, wanneer de teller overloopt.

Een onderbreking betekent: 1) de normale programmastroom wordt onderbroken – 2) een speciale *onderbrekingsroutine* loopt – 3) tenslotte gaat de uitvoering weer door precies waar deze werd onderbroken.

De onderbrekingsroutine moet een Booleaanse variabele instellen, die de "overflow"-toestand markeert – en het hoofdprogramma kan dan deze variabele controleren en daar naar handelen.

Voer eerst dit stukje code in (bijv. voor de `setup()` functie):

```
volatile bool Overflow=false;  // Initially, no overflow has happened

ISR(TIMER1_OVF_vect) {
  Overflow=true;               // Overflow: Set a Booolean marker
}
```

Eerst wordt de Booleaanse variabele gecreëerd. De speciale term `volatile` is een signaal naar de compiler om niet te proberen te raden hoe en wanneer deze variabele verandert. (Noodzakelijk als de variabele in de onderbrekingsroutine wordt benaderd.)

Wat volgt lijkt op een normale functie – maar schijn bedriegt:
ISR betekent *Interrupt Service Routine* – wat betekent dat dit een functie is die wordt aangeroepen wanneer een specifieke onderbreking optreedt.
TIMER1_OVF_vect geeft precies aan welk soort onderbreking – in dit geval wordt deze veroorzaakt door TIMER 1 overflow.

Wat er tussen de **{** accolades **}** gebeurt is aan ons – zolang het maar niet te lang duurt om het te doen.

Om de teller te programmeren om deze interrupt te activeren, moet een ander register worden geïnitialiseerd in de `setup()`:

```
TIMSK1 = 1;       // Interrupt when Timer 1 overflows
```

Wat overblijft is het toevoegen van een controle op de waarde van `Overflow` binnen de `loop()` functie. Vergeet niet deze variabele `false` opnieuw in te stellen bij het starten van een nieuwe meetperiode – doe dit in de `start()` functie. In het geval dat de variabele waar is, moet dit in de output worden weerspiegeld.

Uitdaging 5

Doe dit. En test het indien je een toereikende bron (of geduld) hebt.

Uitdaging 6 (vereist een 16x2 display)

Verplaats de output naar een 16x2 LCD-display.

Nu werkt de Geigerteller onafhankelijk van de PC. Je hoeft alleen maar een batterij aan te sluiten op de Arduino en je bent mobiel. Zolang de opstelling zelf maar stabiel genoeg is om te verplaatsen zonder uit elkaar te vallen.

Je hebt nu verschillende mogelijkheden geprobeerd – en het is niet vanzelfsprekend dat je de voorkeur geeft aan de versie met de start/stop-knop. Je kunt het programma opnieuw bewerken om de automatisch herhaalde metingen terug te krijgen.

Tip: er is een accuclip (voor 9 V accu's) te krijgen, aangesloten op een DC-stekker die past op de voedingsingang van de Arduino Uno. – Handig voor gebruik weg van de PC. Frederiksen artikelnummer 663810 bevat een handvol (5 stuks) van deze clips.

Uitdaging 7

In plaats van simpelweg een Booleaanse variabele in te stellen, kan de onderbrekingsroutine een integervariabele verhogen – en zo het bereik van de teller vergroten.

De variabele voor deze overflowteller moet globaal zijn en `volatile unsigned long` verklaard worden.

Het type voor de totaaltelling moet worden gewijzigd van `word` naar `unsigned long`.

Breng deze wijzigingen aan en voeg het volgende toe:

In `start()` moet de overloopteller op nul worden gezet, samen met het telregister.
In `stop()` moet het resultaat zo worden gecombineerd: 65536 * (overflowteller-variabele) + (telregister)