Verder met ARDUINO

Frans Killian

Frans Killian V 1.0 nov 2019

Voorwoord

Dit is het vervolg op ‘Microcontrollers programmeren met ARDUINO’.
Zelf heb ik ‘Microcontrollers programmeren….’ veel gebruikt als naslagwerk. Bij de vele instrumenten en andere projecten die ik heb gemaakt heb ik het vaak opengeslagen om te zien hoe het ook alweer moest.
Dit boek kan nadat je het hebt doorgewerkt ook als naslagwerk worden gebruikt. Allerlei problemen die ik tijdens het programmeren ben tegengekomen heb ik hierin opgenomen, zodat ik het later nog eens eenvoudig kan nazoeken: Hoe kon je ook al weer nauwkeuriger meten met een analoge ingang? Welke transistor is het meest geschikt voor een taak? Hoe sla ik een variabele op in het EEPROM. Hoe kan je meerdere LEDs elk met een andere frequentie laten knipperen? Hoe maak je de Arduino energiezuiniger?....
Als je wat dieper ingaat op de werking en het programmeren van microcontrollers is het nodig om ook kennis te nemen van de achterliggende wiskunde: Wat is booleaanse algebra? Hoe werken de hexadecimale en binaire getallenstelsels? Wat hebben bitmanipulaties met logische poorten te maken?
Als je een goede ARDUINO-programmeur wilt worden kan je veel kennis, tips en voorbeelden in dit boek vinden. Geen compleet uitgewerkte projecten, maar wel veel voorbeeldsketches waarmee je programmeervaardigheden kan opdoen die je zeker van pas zullen komen bij je een eigen projecten.
Het is niet noodzakelijk om dit boek van voor tot achter helemaal door te werken. In de tekst wordt vaak verwezen naar een ander hoofstuk waar zaken worden uitgelegd die je éérst moet weten.

Toen ik begon met schrijven aan dit boek was ik van plan om in het hoofdstuk communicatie te schrijven over 2,4 GHz transceivers, bluetooth, WiFi, ethernet, internet en meer. Omdat de ontwikkelingen hierin heel snel gaan komen er veel nieuwe boards uit die in deze taken zijn gespecialiseerd en vaak goedkoper zijn dan een arduino shield die dezelfde funcionaliteit geeft aan de arduino. Veel bedrijven maken microcontrollerboards voor allerlei toepassingen waarbij ze meeliften op het succes van de gratis Arduino IDE. Deze paragrafen zouden heel snel verouderd zijn, dus heb ik besloten om ze weg te laten.

Veel plezier met Arduino!

Frans Killian

Inhoud

1 Schakelaars en toetsen

1.1 Toggle

Met een gewoon schakelaartje kan je een lampje aan of uit zetten. Daar is geen microcontroller voor nodig. Met een druktoets die alléén contact maakt als je hem ingedrukt houd kan dit niet. Nou ja, het kan wel, maar dan moet je de toets ingedrukt blijven houden zolang als je het lampje aan wilt laten.
Met een eenvoudige sketch kan je van een druktoets een aan/uit schakelaar maken. Eén keer drukken is aan, nog een keer drukken is uit, enzovoort. We noemen zo'n toets een toggle-toets.

Vragen en opdrachten

1.1 Maak een tekening van een schakeling met een Arduino en een drukschakelaartje tussen pin 10 en GND en een LED met een weerstand van 220Ω in serie tussen pin 11 en GND.

1.2 Bouw de schakeling op een breadboard en upload de volgende sketch:

```
1  // Sketch 1.1: Toggle 1  ////////////////////////////////////////////
2
3  int toets=10;                        // druktoets op pin 10
4  int led=11;                          // led met weerstand op pin 11
5
6  void setup(){
7    pinMode(toets,INPUT_PULLUP);       // druktoets tussen pin 10 en GND
8    pinMode(led,OUTPUT);               // led op pin 11
9  }
10
11 void loop(){
12   if(digitalRead(toets)==LOW){           // Als toets wordt ingedrukt
13     digitalWrite(led, !digitalRead(led)); // Aan/uit omkeren
14     while(digitalRead(toets)==LOW){}      // Wacht op loslaten toets
15   }
16 }
```

1.3 Controleer of de schakeling werkt.

1.2 Contactdender

Heb je bovenstaande opdrachten goed uitgevoerd, dan heb je misschien gemerkt dat de toets toch niet altijd goed werkt. Dat ligt niet aan jou, maar aan een mechanisch probleem van schakelaars. Als een schakelaar gesloten wordt bewegen de contacten snel naar elkaar toe. Als de contacten elkaar raken stuiteren ze soms even tegen elkaar, of glijden ze een beetje tegen elkaar, voordat ze stevig tegen elkaar gedrukt stil staan. Door dit stuiteren en schuiven maken de contacten heel even meerdere keren wel en geen contact. Dit noemen we contactdender. Contactdender duurt vaak maar een paar milliseconden, maar omdat de microcontroller erg snel is kan de programma-lus gedurende deze tijd meerdere keren de schakelaar uitlezen, en daarbij dus meerdere keren de led aan en uit zetten. De oplossing is een korte delay tussen regel 13 en 14 om de contactdender af te wachten voordat de sketch verder gaat.

```
13     digitalWrite(led, !digitalRead(led)); // Aan/uit omkeren
14     delay(50);                            // Contactdender afwachten
15     while(digitalRead(toets)==LOW){}      // Wacht op loslaten toets
```

Hoe lang de delay moet zijn is afhankelijk van de schakelaar. Grote zware schakelaars denderen meestal langer dan kleine lichte druktoetsjes. Meestal is een delay van 50 ms voldoende.
Bij het openen (loslaten) van een schakelaar kan ook contactdender optreden. In dat geval kun je na het loslaten van de toets ook een `delay(50)` opnemen.

1.3 Alternatieve toggle

We kunnen de toggle toets op verschillende manieren programmeren. Zo kan het ook:

```
// Sketch 1.2: Toggle 2  ////////////////////////////////////////////

int toets=10;                          // druktoets op pin 10
int led=11;                            // led met weerstand op pin 11
int A=0;

void setup(){
  pinMode(toets,INPUT_PULLUP);         // druktoets tussen pin 10 en GND
  pinMode(led,OUTPUT);                 // led op pin 11
}

void loop(){
  if(digitalRead(toets)==LOW){         // Als toets wordt ingedrukt
    A++;                               // A wordt A + 1
    if(A>=2)A=0;                       // A wordt maximaal 1
    if(A==0)digitalWrite(led, LOW);    // led uitzetten
    if(A==1)digitalWrite(led, HIGH);   // led aanzetten
    delay(50);                         // Contactdender afwachten
    while(digitalRead(toets)==LOW){}   // Wacht op loslaten toets
    delay(50);                         // Contactdender afwachten
  }
}
```

Het voordeel van deze manier is dat we nu ook andere functies kunnen laten uitvoeren als aan één van de **`if(A==...)`** voorwaarden wordt voldaan. Het is een wat universelere manier. Bovendien is deze sketch eenvoudig uit te breiden naar een menu-selectie toets met meerdere menu-opties en kan eenvoudig het aantal menu-items worden gewijzigd.

1.4 Menu selectie toets

Met de schakeling van de vorige paragraaf kunnen we heel eenvoudig een dimbaar schemerlampje maken. Het lampje kan in 5 stappen worden gedimd:

```
// Sketch 1.3: Eenvoudige dimmer  ///////////////////////////////////////

int toets=10;                          // druktoets tussen pin 10 en GND
int led=11;                            // led met weerstand op pin 11
int A=0;

void setup(){
  pinMode(toets,INPUT_PULLUP);         // Druktoets tussen pin 10 en GND
  pinMode(led,OUTPUT);                 // led op pin 11
}

void loop(){
  if(digitalRead(toets)==LOW){         // Als toets wordt ingedrukt
    A=A+51;                            // A wordt A+51
    if(A>=256)A=0;                     // A wordt maximaal 255
    analogWrite(led, A);               // PWM dimwaarde = A
    delay(50);                         // Contactdender afwachten
    while(digitalRead(toets)==LOW){}   // Wacht op loslaten toets
    delay(50);                         // Contactdender afwachten
  }
}
```

Ook hier kan de sketch wat universeler worden gemaakt met verschillende if-vergelijkingen. De volgende sketch doet hetzelfde als de dimmer op de vorige bladzijde, maar je kunt hem gemakkelijk aanpassen zodat je andere functies krijgt.

```
// Sketch 1.4: Menuselectietoets  ////////////////////////////////////
int toets=10;                          // druktoets tussen pin 10 en GND
int led=11;                            // led met weerstand op pin 11
int A=0;
void setup(){
  pinMode(toets,INPUT_PULLUP);         // druktoets tussen pin 10 en GND
  pinMode(led,OUTPUT);                 // led met weerstand op pin 11
}
void loop(){
  if(digitalRead(toets)==LOW){         // Als toets wordt ingedrukt
    A=A+1;                             // A wordt A+1
    if(A>5)A=0;                        // A wordt maximaal 5
    if (A==0)analogWrite(led,0);
    if (A==1)analogWrite(led,51);
    if (A==2)analogWrite(led,102);
    if (A==3)analogWrite(led,153);
    if (A==4)analogWrite(led,204);
    if (A==5)analogWrite(led,255);
    delay(50);                         // Contactdender afwachten
    while(digitalRead(toets)==LOW){}   // Wacht op loslaten toets
    delay(50);                         // Contactdender afwachten
  }
}
```

Als je vind dat de laagste dimmerstand **`(A==1)`** toch nog te fel is kan je de PWM waarde iets lager zetten, bijvoorbeeld: **`if(A==1)analogWrite(led,20)`**.
Je kunt natuurlijk van alles programmeren na de if-vergelijkingen. In plaats van een hele rij if-vergelijkingen kan je ook de switchcase-opdracht gebruiken. Je kunt deze sketch gebruiken als sjabloon voor een menu-selectie toets.

1.5 Extra opstart-functie in setup

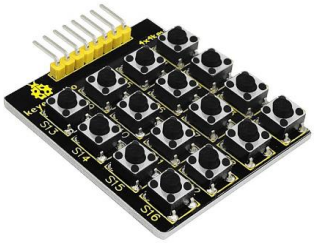
De **`void setup()`** functie wordt maar één keer uitgevoerd, direct na het opstarten van de Arduino. Als je tijdens de setup een druktoets uitleest kun je tijdens de setup een bepaalde opstartfunctie uitvoeren, bijvoorbeeld een voorkeursinstelling maken die je maar heel zelden hoeft te veranderen. De setup functie kan er dan zo uitzien:

```
void setup(){
  pinMode(toets, INPUT_PULLUP);  // Druktoets tussen pin toets en GND.
  if(digitalRead(toets)==LOW){   // Toets tijdens opstarten ingedrukt?
    setupfunctie();              // Dan setupfunctie uitvoeren.
  }
}
```

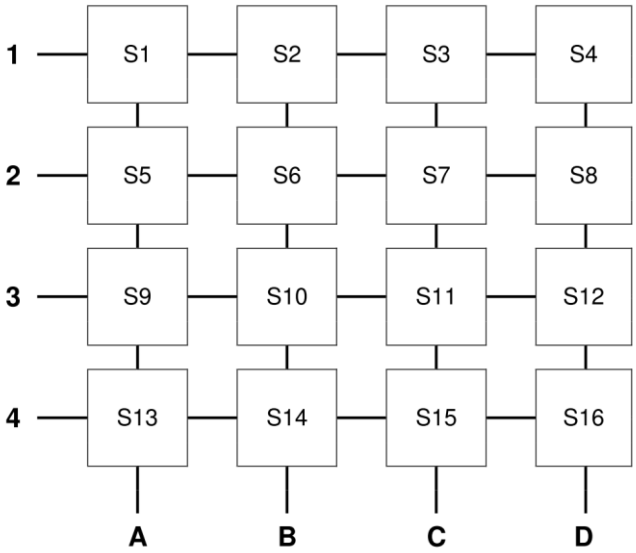
Als je de toets ingedrukt houd tijdens het opstarten, dan zal eerst de setupfunctie worden uitgevoerd. Het uitlezen van de toets in de setup moet natuurlijk wel ná de **`pinMode`** opdracht staan.

1.6 Matrix keypad

Als je veel druktoetsen in je project wilt gebruiken kom je al snel digitale ingangen tekort. Een manier om het aantal benodigde digitale ingangen te beperken is door gebruik te maken van een toetsenmatrix (key matrix). Bij een matrix toetsenbord worden de schakelaars in rijen en kolommen geschakeld. Elke rij en elke kolom heeft één aansluiting. Een matrix van 12 toetsen (3 rijen van 4) heeft dus maar 3+4=7 aansluitingen. Een matrix van 16 toetsen in 4 rijen van 4 heeft 4+4=8 aansluitingen enzovoort.

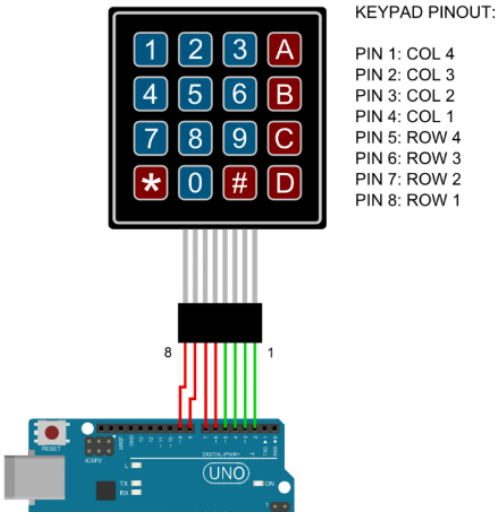


In onderstaand schema zie je een toetsenmatrix van 16 toetsen, S1 t/m S16. De matrix heeft 8 aansluitingen, 1, 2, 3, 4, A, B, C en D. Als je toets S7 indrukt, worden de aansluitingen 2 en C met elkaar verbonden. Als je toets S2 indrukt worden de aansluitingen 1 en B met elkaar verbonden enzovoort. Welke toets is ingedrukt als rij 3 verbonden is met kolom B?



Het uitlezen van de toetsen gaat als volgt: De rijen 1 t/m 4 worden aangesloten op 4 digitale uitgangen van de Arduino, De kolommen A t/m D worden aangesloten op 4 digitale ingangen. Om de beurt worden de rijen 1 t/m 4 hoog gemaakt en worden de kolommen uitgelezen. Op het moment dat rij 3 hoog is gemaakt met `digitalWrite(rij3, HIGH)` en ingang C is ook hoog `if(digitalRead(kolomC)==HIGH)` dan is toets S11 ingedrukt.
Je kan zelf een toetsmatrix maken met losse schakelaartjes, maar je kan ook een complete toetsmatrix kopen.
Als je niet weet welke aansluitingen voor welke rij of kolom zijn, dan kun je dat als volgt meten: Sluit een weerstandmeter (liefst één die gaat piepen bij een weerstand van 0Ω) aan op twee willekeurige aansluitingen van de matrix. Druk de toetsen één voor één in. Als geen van de toetsen 0Ω aangeeft, dan heb je twee rijen of twee kolommen aangesloten. Als de weerstandmeter 0Ω aangeeft (piept), dan weet je dat de aansluitingen horen bij de rij en de kolom van die schakelaar.

Als je een matrixtoetsenbord wilt gebruiken dan kan je nog beter en gemakkelijker de Keypad library gebruiken.
Het uitlezen van het volgende zelfklevend folietoetsenbord kan met de volgende voorbeeldsketch.




```
// Sketch 1.5: Matrix toetsenbord ////////////////////////////////////////////////

#include <Keypad.h>              // de Keypad library is nodig
const byte rij = 4;              // De toetsmatrix heeft 4 rijen
const byte kol = 4;              // de toetsmatrix heeft 4 kolommen

char toetsen[rij][kol] = {       // definieer toetskarakters in
  {'1','2','3','A'},            // 2 dimensionale array 'toetsen'
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rijPinnen[rij] = {9, 8, 7, 6};  // rij 1 op pin 9, rij 2 op pin 8 enz.
byte kolPinnen[kol] = {5, 4, 3, 2};  // kol 1 op pin 5, kol 2 op pin 4 enz.
Keypad matrix = Keypad(makeKeymap(toetsen), rijPinnen, kolPinnen, rij, kol);
                                 // Initialiseer nieuwe keypad genaamd matrix
void setup(){
  Serial.begin(9600);            // seriele communicatie starten
}

void loop(){
  char toets = matrix.getKey(); // definieer het karakter toets
  if (toets){                    // als toets wordt ingedrukt
    Serial.println(toets);       // ingedrukte toets in monitorvenster tonen
  }
}
```

In 2016 heb ik een controller gebouwd voor de theaterverlichting in de aula van mijn school. De controller heeft 30 druktoetsen die in een matrix van 5 x 6 geschakeld zijn. Je ziet dat ze op het apparaat niet per sé in 5 rijen van 6 gerangschikt hoeven te zijn. Als je ze maar met de juiste digitale in- en uitgangen van de Arduino verbind.

LIchtcontroller met 5 x6 toetsenmatrix

Vragen en opdrachten

1.4 Waarom heeft een toetsmatrix van vier toetsen geen zin?

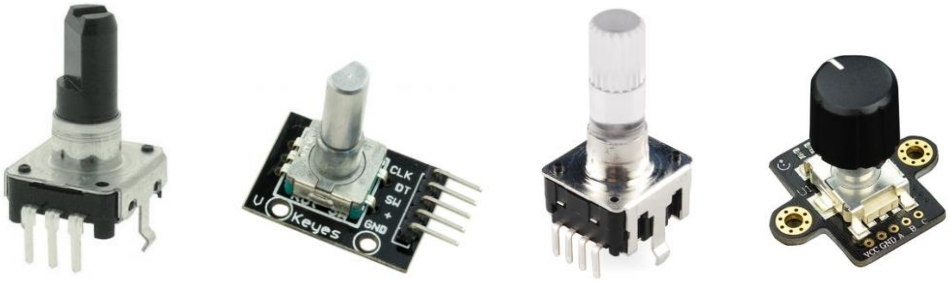
1.5 Je wilt iets bouwen met 27 druktoetsjes in een matrix. Leg uit hoeveel digitale in- en uitgangen je hiervoor minimaal nodig hebt.

1.6 De Aruino Mega heeft 54 digitale in/uitgangen. Hoeveel schakelaartjes kan je hier maximaal aansluiten met een matrix?

1.7 Sluit een matrix toetsenbord en een led aan op de Arduino en maak een sketch waarmee je de led verschillende dim-standen kan geven met de verschillende toetsen.

1.7 Rotary encoder

Een rotary encoder ziet er vanaf de buitenkant van een apparaat hetzelfde uit als een potmeter. Gewoon een draaiknop. Het is echter geen weerstand, maar het zijn twee schakelaars. De rotary encoder kan eindeloos draaien, zowel rechtsom als linksom. Bij de meeste encoders voel je tijdens het draaien klikjes.



De twee schakelaars A en B in de rotary encoder kunnen samen 4 mogelijke combinaties van aan (1) en uit (0) hebben. Draait de encoder rechtsom, dan worden de stappen in de volgorde ...1-2-3-4-1... doorlopen. Linksom draaien geeft de stappenvolgorde ...4-3-2-1-4... In een waarheidstabel ziet het er zo uit:

Stap	A	B		
1	0	0	> > > >	< < < <
2	1	0		
3	1	1		
4	0	1		

rechtsom | linksom

Het aan en uit schakelen van de twee schakelaartjes gebeurt volgens een Gray-code. Dat betekent dat er per stap steeds maar één schakelaar van positie verandert. Uit het schakelpatroon kan de Arduino afleiden of er gedraaid wordt, en welke kant er op gedraaid wordt. Een tellerstand houd de positie van de rotary encoder bij.
Eerst wordt gekeken of de positie van schakelaar A verandert (regel 22 in onderstaand voorbeeld). A verandert tussen stap 1 en 2, en tussen stap 3 en 4. Het doorlopen van de waarheidstabel van 1 naar 4 (of van 4 naar 1) geeft dus maar twee veranderingen in de positie van schakelaar A en dus 2 tellen.
Als A verandert is, en B is daarna gelijk aan A, dan draaien we linksom (**`teller--`**), van 4 naar 3 of van 2 naar 1. Als A veranderd is en B is daarna ongelijk aan A, dan draaien we rechtsom (**`teller ++`**), van 1 naar 2 of van 3 naar 4. In de sketch ziet het er zo uit:

```
20    A=digitalRead(sA);          // A = stand schakelaar A
21    B=digitalRead(sB);          // B = stand schakelaar B
22    if(A != Aoud){              // A is verandert, er is gedraaid
23      Aoud = A;                 // Nieuwe waarde voor Aoud
24      if(A == B)teller --;      // Als A is gelijk aan B linksom
25      else teller ++;           // Anders rechtsom
```

We kunnen hetzelfde doen voor schakelaar B. Dan worden ook de stappen 2>3 en 3>2 geteld. Elke stap in de tabel resulteert dan in een **`teller ++`** of een **`teller --`**. Het doorlopen van de 4 stappen resulteert dan in 4 tellen. Een hele omwenteling van de rotary geeft dan 2 keer zoveel tellen als wanneer we alleen de verandering in schakelaar A verwerken. De complete sketch met verwerken van veranderingen in A en B ziet er dan zo uit:

```
1  // Sketch 1.6: Rotary encoder ////////////////////////////////
2
3  byte sA = 2;                    // Rotary schakelaar A op pin 2
4  byte sB = 3;                    // Rotary schakelaar B op pin 3
5  int teller = 0;                 // Stand van de Rotary
6  bool A;                         // Stand van schakelaar A
7  bool B;                         // Stand van schakelaar B
8  bool Aoud;                      // Vorige stand van schakelaar A
9  bool Boud;                      // Vorige stand van schakelaar B
10
11 void setup() {
12    pinMode (sA,INPUT_PULLUP);
13    pinMode (sB,INPUT_PULLUP);
14    Aoud = digitalRead(sA);      // Startpositie schakelaar A
15    Boud = digitalRead(sB);      // Startpositie schakelaar B
16    Serial.begin (9600);         // Seriële monitor aan
17  }
18
19 void loop() {
20    A=digitalRead(sA);          // A = stand schakelaar A
21    B=digitalRead(sB);          // B = stand schakelaar B
22    if(A != Aoud){              // A is verandert, er is gedraaid
23      Aoud = A;                 // Nieuwe waarde voor Aoud
24      if(A == B)teller --;      // Als A is gelijk aan B linksom
25      else teller ++;           // Anders rechtsom
26      Serial.println(teller);   // Print tellerstand naar monitor
27    }
28    if(B != Boud){              // B is verandert, er is gedraaid
29      Boud = B;                 // Nieuwe waarde voor Boud
30      if(B == A)teller ++;      // Als B is gelijk aan A rechtsom
31      else teller --;           // Anders linksom
32      Serial.println(teller);   // Print tellerstand naar monitor
33    }
34 }
```

Is het voldoende om alleen de verandering in schakelaar A te gebruiken, dan kun je de regels 28 t/m 33 weglaten.
De rotary encoder heeft meestal 3 aansluitingen. De beide schakelaartjes hebben één gemeenschappelijke aansluiting. Dat is meestal de middelste pin. Gebruik je bovenstaande sketch met **`INPUT_PULLUP`**, dan moet de middelste pin van de rotary worden aangesloten op GND.
Vaak heeft de encoder nog een derde schakelaar die contact maakt als je de knop indrukt.
Er zijn ook encoders met in de knop een driekleurenled ingebouwd.
De rotary encoder werkt betrouwbaarder als je de schakelaartjes A en B aansluit via een externe interrupt (zie hoofdstuk 2.5, blz 19). Ook voor de rotary encoder zijn er vele libraries die het leven makkelijker maken. Deze libraries maken ook meestal gebruik van interrupts.

Vragen en opdrachten

1.8 Zoek op wat een Gray-code is.
1.9 Waarom zijn de schakelaartjes van de rotary encoder volgens een Gray-code gecodeerd?

2 Verder met programmeren

2.1 Functies met parameters

De functies **`void setup()`**, **`void loop()`** en **`millis()`** hebben geen parameters want er staat niets tussen de haakjes. Tusen de haakjes van een functie staan de parameters. De meeste functies gebruiken één of meer parameters. Voorbeelden van functies met parameters zijn **`delay(x)`**, **`random(x,y)`**, **`pinMode(x,y)`**, **`analogRead(x)`** en **`digitalWrite(x,y)`**. Een parameter is eigenlijk gewoon een getal. Dat kan dus ook een variabele zijn, zelfs als die variabele wordt verkregen uit een andere functie zoals bij **`delay(random(x,y))`**.
Ook als je zelf een functie definiëert kan je één of meer parameters met die functie meegeven. In het volgende voorbeeld wordt de functie **`flits(x)`** met één parameter gedefinieerd. Met deze functie laten we een led x milliseconden lang branden:

```
1  // Sketch 2.1: Functie met 1 parameter ////////////////////////
2
3  int led=13;
4
5  void setup(){
6    pinMode(led, OUTPUT);
7  }
8  void loop(){
9    flits(100);            // De functie flits met parameter 100
10   delay(5000);           // Wacht 5 seconden
11 }
12
13 void flits(int wacht){          // wacht krijgt waarde 100
14     digitalWrite(led, HIGH);    // led aan
15     delay(wacht);               // wacht 100 ms
16     digitalWrite(led, LOW);     // led uit
17 }
```

In het volgende voorbeeld heeft de functie **`flits(x, y, z)`** drie parameters waarmee de led x keer knippert en daarbij steeds y ms aan en z ms uit is.

```
1  // Sketch 2.2: Functie met 3 parameters ////////////////////////
2
3  int led=13;
4
5  void setup(){
6    pinMode(led, OUTPUT);
7  }
8  void loop(){
9    flits(5, 100, 200);              // Functie met 3 parameters
10   delay(5000);                     // Wacht 5 seconden
11 }
12 void flits(int aantal, int aanTijd, int uitTijd){
13   for(int n=0; n<aantal; n++){     // aantal = 5 keer flitsen
14     digitalWrite(led, HIGH);       // led aan
15     delay(aanTijd);                // 100 ms aan
16     digitalWrite(led, LOW);        // led uit
17     delay(uitTijd);                // 200 ms uit
18   }
19 }
```

In de volgende waarheidstabel zie je de cyclus van de verkeerslichten bij een voetgangers-oversteekplaats.

waarheidstabel verkeerslichten

autolichten			voetgangerslichten		tijdsduur
rood	oranje	groen	rood	groen	(s)
0	0	1	1	0	30
0	1	0	1	0	5
1	0	0	1	0	5
1	0	0	0	1	8
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	0	0	0,5
1	0	0	0	1	0,5
1	0	0	1	0	5

Door gebruik te maken van een functie met 6 parameters kunnen de 6 kolommen van de waarheidstabel bijna letterlijk worden overgenomen in de sketch:

```
// Sketch 2.3: Voetgangersoversteekplaats ////////////////////////////////

int ledAR=13; int ledAO=12; int ledAG=11; int ledVR=10; int ledVG=9;

void setup(){
  pinMode(ledAR, OUTPUT); pinMode(ledAO, OUTPUT); pinMode(ledAG, OUTPUT);
  pinMode(ledVR, OUTPUT); pinMode(ledVG, OUTPUT);
}

void loop(){
  setLicht(0,0,1,1,0,30000);  // Autolicht 30 seconden op groen
  setLicht(0,1,0,1,0,5000);   // Autolicht 5 seconden op oranje
  setLicht(1,0,0,1,0,5000);   // Autolicht 5 seconden op rood
  setLicht(1,0,0,0,1,8000);   // Voetgangerslicht 8 seconden op groen
  for(int n=0; n<8; n++){     // Voetgangerslicht 8 seconden knipperen
    setLicht(1,0,0,0,0,500);
    setLicht(1,0,0,0,1,500);
  }
  setLicht(1,0,0,1,0,5000);   // Voetgangerslicht 5 seconden op rood
}

void setLicht(bool autoR, bool autoO, bool autoG,
              bool voetR, bool voetG, int wacht){
  digitalWrite(ledAR, autoR);
  digitalWrite(ledAO, autoO);
  digitalWrite(ledAG, autoG);
  digitalWrite(ledVR, voetR);
  digitalWrite(ledVG, voetG);
  delay(wacht);
}
```

2.2 Functies met een return-waarde

Void betekent leeg of geen. In **void setup()** betekent **void** dat de functie geen datatype heeft. Een functie heeft een datatype nodig als de functie een waarde terugkoppelt via return.
In het volgende voorbeeld zie je in regel 5 de functie **schuineZijde(a, b)** waarmee de schuine zijde van een rechthoekige driehoek met rechthoekszijden **a** en **b** in regel 10 word uitgerekend. De uitkomst (variabele **c**) wordt in regel 11 met de opdracht **return** teruggekoppelt naar de functie in regel 5. De functie krijgt nu de waarde van **c** terug en deze wordt in regel 5 toegewezen aan de variabele **hypotenusa**. Omdat de functie nu een waarde vertegenwoordigt en dus niet leeg is krijgt hij in regel 9 een datatype in plaats van **void**. In dit geval **float**.

```
1  // Sketch 2.4: Pythagoras ///////////////
2
3  void setup(){
4    Serial.begin(9600);
5    float hypotenusa = schuineZijde(3, 4);
6    Serial.print(hypotenusa);
7  }
8
9  float schuineZijde(float a, float b){
10   float c = sqrt(sq(a) + sq(b));
11   return c;
12 }
13
14 void loop(){}
```

Dit lijkt erg ingewikkeld, en bovendien overbodig omdat hetzelfde is te bereiken met globale variabelen. De voordelen van de return-waarde worden duidelijk bij een grote sketch, met veel variabelen. Omdat alle variabelen in de functies lokaal zijn (zelfs de variabelen **c** en **hypotenusa** die dezelfde waarde hebben) hoef je je niet druk te maken over het definieren van globale variabelen. Bovendien hoef je geen rekening te houden met globale variabelen als je een functie aan je sketch toevoegt of weghaalt.

2.3 No delay!

Vrijwel elk boek over de Arduino begint met de sketch blink! Waarschijnlijk heb jij ook bij de eerste kennismaking met de Arduino een knipperlicht geprogrammeerd met de opdracht **delay(1000)**.
Het grote nadeel van delay is dat de Arduino tijdens de delay niets anders meer kan doen dan wachten. Een schakelaartje, potmeter of sensor wordt tijdens de delay niet uitgelezen.
Twee knipperende LEDs, met verschillende delaytijden van bijvoorbeeld 500 milliseconden en 550 milliseconden is ook erg lastig om te programmeren met delays. Eigenlijk kun je dus beter geen delays gebruiken. Zó kun je een knipperende led programmeren zonder **delay()**:

```
1  // Sketch 2.5: Knipper zonder delay ////////////////////////////////////
2
3  int led=13;
4  unsigned long start=0;
5
6  void setup(){
7    pinMode(led, OUTPUT);
8  }
9
10 void loop(){
11   if((millis()-start)>=1000){       // Als 'delay' van 1000 ms is bereikt
12     start=millis();                 // Delayklok resetten
13     digitalWrite(led,!digitalRead(led));  // Aan wordt uit, uit wordt aan
14   }
15 }
16
```

De lichtcontroller op bladzijde 11 heeft naast 30 druktoetsen ook 11 potmeters om de kleuren van 32 aangesloten lampen te kunnen regelen en om de lampen te draaien en te dimmen.
De lampen kunnen ook met verschillende frequenties knipperen. De sketch voor deze controller is meer dan 2200 programmaregels lang en bevat geen enkele `delay()` opdracht!

Vragen en opdrachten

2.1 Bestudeer sketch 2.5 op de vorige bladzijde tot je snapt hoe het werkt.
2.2 Als je in deze sketch een schakelaar zou moeten programmeren, waar zou je de digitalRead(schakelaar) opdracht dan zetten?
2.3 Verander de sketch zodat de led steeds 50 ms aan, en 950 ms uit is, zonder delay().
2.4 Je gaat een schakeling bouwen met drie potmeters op de analoge ingangen 1, 2 en 3 en drie led's met weerstanden van 220Ω op digitale uitgangen 13, 12 en 11.
Teken het schema van deze schakeling.
2.5 Bouw de schakeling en upload de volgende sketch (2.6).
2.6 De drie potmeters regelen de knipperfrequenties van de drie led's. Controleer of de schakeling werkt.
2.7 Breid de schakeling uit met een drukschakelaar (eerst het schema tekenen).
Zolang de schakelaar ingedrukt is moeten de drie led's synchroon knipperen met een 'delay' van 200 milliseconden. Je mag geen `delay()` opdracht gebruiken!

```
// Sketch 2.6: Drie potmeters regelen frequenties van drie led's //////////

int potA=1;               // Potmeter A op analoge ingang 1
unsigned long startA=0;   // timer van led A
int ledA=13;              // led A op pin 13
int potB=2;               // Potmeter B op analoge ingang 2
unsigned long startB=0;   // timer van led B
int ledB=12;              // led B op pin 12
int potC=3;               // Potmeter C op analoge ingang 3
unsigned long startC=0;   // timer van led C
int ledC=10;              // led C op pin 10

void setup(){
  pinMode(ledA, OUTPUT);
  pinMode(ledB, OUTPUT);
  pinMode(ledC, OUTPUT);
}
void loop(){
  if((millis()-startA)>=analogRead(potA)){  // Als eindtijd bereikt is
    startA=millis();                        // DelayklokA resetten
    digitalWrite(ledA,!digitalRead(ledA));  // Aan wordt uit, uit wordt aan
  }
  if((millis()-startB)>=analogRead(potB)){
    startB=millis();
    digitalWrite(ledB,!digitalRead(ledB));
  }
  if((millis()-startC)>=analogRead(potC)){
    startC=millis();
    digitalWrite(ledC,!digitalRead(ledC));
  }
}
```

2.4 Interrupts

De microcontroller moet veel verschillende taken uitvoeren, zoals het bijhouden van timers, seriële communicatie, rekenen, naar het geheugen schrijven, data uit het geheugen ophalen enz.
Sommige van deze taken zijn belangrijker dan andere en die taken kunnen voorrang krijgen. De microcontroller onderbreekt dan wat hij aan het doen is om eerst een belangrijkere taak uit te voeren. We noemen zo'n onderbreking een interrupt.

Een interrupt kan invloed hebben op de timers dus als je kritische tijdmetingen wilt doen voor bijvoorbeeld een ultrasone afstandsensor, dan kan je de interrupts beter even uitzetten met `noInterrupts()`. Sommige functies zoals seriële communicatie werken niet meer als de interrupts uit staan. Het is daarom belangrijk om de interrupts zo snel mogelijk ook weer aan te zetten met `interrupts()`.

2.5 Externe interrupt

Met een externe interrupt kan je de processor onderbreken met een signaal van buitenaf. Dit is nodig bij snelle digitale sensoren zoals een Geiger-Müller telbuis. Een Geiger Müller telbuis is een sensor die reageert op ioniserende straling (radioactieve straling). De buis produceert aan de uitgang zeer korte pulsjes. Hoe hoger de dosis straling, des te meer pulsjes per seconde. Omdat de pulsjes zeer kort zijn kan de Arduino via `digitalRead` in een lus gemakkelijk een pulsje missen als de processor toevallig ergens anders in de lus bezig is. Rotary encoders en digitale lichtsensoren kunnen ook zeer veel korte pulsjes geven. Daarom worden dit soort sensoren vaak aangesloten via een externe interrupt. Zodra er een puls van de sensor binnenkomt wordt het met voorrang verwerkt. Dan hoef je geen enkel pulsje te missen.
De Arduino UNO heeft twee externe interrupts, genummerd 0 en 1. Interrupt 0 is altijd verbonden met digitale pin 2, interrupt 1 is altijd verbonden met digitale pin 3.
Zo gebruik je een externe interrupt:

```
attachInterrupt(interruptnummer, ISR, mode);
```

ISR (Interrupt Service Routine) is de functie die moet worden uitgevoerd bij een interrupt en mode bepaalt of de interrupt wordt getriggerd door een verandering op de pin van low naar high (`RISING`), high naar low (`FALLING`) of beiden (`CHANGE`).
De volgende sketch meet hoeveel keer een schakelaar dendert (zie ook hoofdstuk 1.2, blz 7). Omdat het denderen met erg korte pulsjes gaat is de schakelaar aangesloten via een externe interrupt.

```
// Sketch 2.7: Contactdender meten ////////////////////////////////////////

byte schakelaar=2;       // Denderende schakelaar op pin 2 (interruptpin)
volatile byte teller=0;  // Variabele die het aantal denders bijhoudt

void setup(){
  pinMode(schakelaar, INPUT_PULLUP);   // schakelaar tussen pin 2 en GND
  attachInterrupt(0, Tellen, FALLING); // definieer interrupt
  Serial.begin(9600);                  // seriele monitor aanzetten
}

void loop() {
  if(teller!=0){                       // Zodra er geschakeld wordt
    Serial.print("Aantal denders: ");
    delay(100);                        // Contactdender afwachten;
    Serial.println(teller);            // Resultaat naar monitor
    teller=0;                // Teller resetten voor de volgende meting
  }
}

void Tellen(){               // De ISR (Interrupt Service Routine)
  teller++;                  // Tel aantal neergaande flanken
}
```

Vragen en opdrachten

2.8 Meet met behulp van bovenstaande sketch hoe vaak verschillende schakelaars denderen.
2.9 Is er verschil in contactdender bij openen en sluiten van de schakelaar?
2.10 Verander de sketch zodat je kan meten hoe lang de contactdender duurt. Gebruik hiervoor micros() want delay() en millis() zijn niet betrouwbaar tijdens een interrupt.

Verschillende Arduino boards hebben de externe interrupts op verschillende pinnen:

Arduino board	UNO	MEGA2560	DUE	Leonardo	Pro Micro
Interrupt 0	pin D2	pin D2	pin D0	pin D3	pin D3
Interrupt 1	Pin D3	pin D3	pin D1	pin D2	pin D2
Interrupt 2	-	pin D21	pin D2	pin D0	pin D0
Interrupt 3	-	pin D20	pin D3	pin D1	pin D1
Interrupt 4	-	pin D19	pin D4		
Interrupt 5	-	pin D18	pin D5		

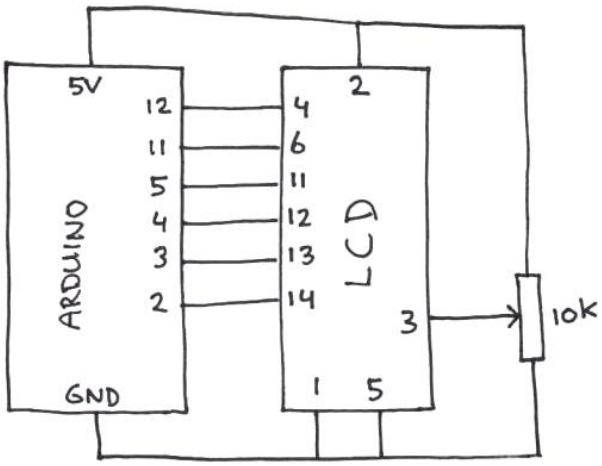
Waar je verder op moet letten bij het gebruik van externe interrupts:

- Hou de ISR kort en snel.
- Gebruik geen verdere functieaanroepen binnen de ISR.
- De ISR kan geen parameters meenemen, en geeft ook geen waarde return (zie 2.1 en 2.2.)
- Gebruik alleen volatile variabelen in de ISR. Als je geen volatile gebruikt worden de variabelen tijdelijk in een register geschreven om de processor sneller te maken. Als tijdens het schrijven een ISR wordt getriggerd, dan wordt het schrijven onderbroken en kan de variabele fout worden opgeslagen.
- Gebruik geen delay(), maar delayMicroseconds().
- Gebruik geen millis(), maar micros().
- Seriële communiatie werkt niet binnen de ISR.

2.6 Hoe snel is de Arduino?

De processor in de Arduino UNO werkt met een klokfrequentie van 16MHz. Een klokcyclus duurt dus éénzestiende microseconde. Dat betekent dat hij 16 miljoen handelingen per seconde uitvoert. Een eenvoudige opdracht bestaat al uit meerdere handelingen en dus meerdere klokcycli. Het vertalen van de sketch naar de instructies in de microcontroller-chip duurt ook vaak meerdere cycli. Hoe lang de Arduino over een bepaalde opdracht doet is moeilijk te voorspellen, maar het is te meten door vóór de opdracht een pin laag te maken met `digitalWrite(8, LOW)` en direct na de opdracht de pin weer hoog te maken met `digitalWrite(8, HIGH)`. Met een geheugenoscilloscoop meet je vervolgens hoe lang de uitgang (pin 8) laag is geweest. Heb je geen oscilloscoop? De meeste Arduino gebruikers hebben geen oscilloscoop dus daarom doen we het zonder. We maken een nauwkeurige stopwatch met een tweede Arduino en daarmee meten we hoeveel microseconden (1 microseconde = 10^{-6} s) de pin laag is met de opdracht `pulseIn(pin, LOW)`. Bij het uitvoeren van de pulseIn opdracht wacht de processor tot het signaal op de pin laag wordt en start dan de meting. Zodra het signaal op de pin weer hoog wordt stopt de meting.

Het is niet zo handig om de Arduino stopwatch via de seriële monitor uit te lezen want we willen ook een Arduino kunnen programmeren met de verschillende programmaregels waarvan we de uitvoertijden willen meten. Twee Arduino's op één computer kan wat verwarrend zijn. Daarom maken we een standalone stopwatch die de gemeten waarde op een LCD display laat zien.
Sluit het 16*2 LCD display aan en upload sketch 2.8 van de stopwatch:




```
// Sketch 2.8: Microseconden stopwatch ////////////////////////////////////////
#include <LiquidCrystal.h>              //  LCD Library
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // LCD (RS, E, D4, D5, D6, D7)
                                       // Op Arduino pin (12, 11, 5.....)
void setup(){
  lcd.begin(16,2);                     // Display 16 karakters en twee regels
  pinMode(8, INPUT);                   // Ingang van de stopwatch
}

void loop(){
  duur = pulseIn(8, LOW)-5;            // Meet hoe lang pin 8 laag wordt
  lcd.clear();                         // Display leeg maken
  lcd.setCursor(0, 0);                 // Cursor naar kolom 0, rij 0
  lcd.print(duur);                     // Tijdsduur op display tonen
}
```

Upload onderstaande sketch in de andere Arduino:

```
// Sketch 2.9: Snelheid meten ////////////////////////
void setup(){
  pinMode (8, OUTPUT);
  pinMode (9, OUTPUT);
  digitalWrite (8, LOW);
  Serial.begin(9600);
}

void loop(){
  digitalWrite (8, LOW);       // Start meting
//////////////////////////////////////////////////////
// Zet hier de code waarvan je de snelheid wilt meten
//////////////////////////////////////////////////////
  digitalWrite (8, HIGH);      // Stop meting
  delay(500);
}
```

Verbind de GND en eventueel de +5V van beide Arduino's met elkaar.
Verbind ook pin 8 van de stopwatch-Arduino met pin 8 van de andere Arduino.

Als er géén code op regel 13 van bovenstaande sketch staat, dan wordt pin 8 laag en vervolgens zo snel mogelijk weer hoog. Dit duurt ongeveer 5 µs (microseconden). Dit willen we niet mee meten en daarom wordt in de stopwatch in regel 12 hiervoor gecorrigeerd met -5.
Met deze opstelling heb ik de volgende resultaten gemeten:

```
digitalRead(9);                    // 5   us
digitalWrite(9, LOW)               // 7   us
////////////////////////////////////////////////////////////////////////////
  digitalWrite(9,HIGH);
  digitalWrite(9,LOW);
  //enz ... Totaal 100x aan en uit   1251 us
////////////////////////////////////////////////////////////////////////////
  for (int n=0; n<100; n++){       // Totaal 100x aan en uit
  digitalWrite(9,HIGH);
  digitalWrite(9,LOW);}            // 1276 us
////////////////////////////////////////////////////////////////////////////
analogWrite(0,127);                // 13  us
analogRead(0);                     // 110 us
byte A=random(10);                 // 95  us
int A=random(10);                  // 95  us
int A=random(1000000);             // 95  us
Serial.begin(9600);                // 45  us
Serial.print(1);                   // 61  us
int A=123; Serial.print(A);        // 150 us
const int A=123; Serial.print(A); // 150 us
Serial.print(A);                   // 60  us
Serial.print("A");                 // 14  us
Serial.print("Arduino");           // 65  us
Serial.print("Arduino speed");     // 117 us
////////////////////////////////////////////////////////////////////////////
long x=0; long y=0;
for(x=0; x<100; x++){
  y = x + sq(x+1);
  if (y>10) y=0;}                  // 682 us
//////////////////////
  digitalWrite (8, HIGH);          // Stop meting
  Serial.print(y);                 // Nodig om optimalisatie te voorkomen
  delay(500);
////////////////////////////////////////////////////////////////////////////
long x=0; float y=0;
for(x=0; x<100; x++){
  y = x + sq(x+1);
  if (y>10) y=0;}                  // 1292 us
//////////////////////
  digitalWrite (8, HIGH);          // Stop meting
  Serial.print(y);                 // Nodig om optimalisatie te voorkomen
  delay(500);
////////////////////////////////////////////////////////////////////////////
```

digitalRead duurt ongeveer 5 µs (microseconden). Dat lijkt erg kort, maar in de wereld van de digitale elektronica is 5 µs erg lang. Bij een systeemklokfrequentie van 16 MHz duurt dit 16*5 = 80 klokcycli. digitalWrite duurt ietsje langer, ongeveer 7 µs.
100 keer digitalWrite (9, HIGH) gevolgd door digitalWrite (9, LOW) duurt 1250 µs.
Als je die 100 keer in een lus programmeert duurt het maar ietsje langer, 1276 µs.
analogWrite en analogRead duren nog langer, respectievelijk 13 en 110 µs.
Het definiëren van een random duurt 95 µs, ongeacht de omvang van het random getal.
Een variabele versturen met Serial.print duurt ongeveer 60 µs. Karakters versturen via Serial.print duurt langer naarmate er meer karakters worden verstuurd, tussen de 8 en 9 µs per karakter.

Bij het meten van berekeningen met long y is het nodig om na het stoppen van de puls op pin 8 de waarde van y een keer te gebruiken, bijvoorbeeld met Serial.print(y). Doe je dit niet, dan weet de microcontroller dat hij het getal y verder niet nodig heeft, en gaat hij het ook niet uitrekenen. Dit noemen we compileroptimalisatie. De stopwatch geeft dan 0 µs aan.

De voorbeeldberekening met long y duurt 682 µs. Als je long y verandert in float y, dan duurt de berekening bijna twee keer zo lang, 1292 µs.

2.7 Arduino sneller maken

Systeemklok opvoeren

Als we de systeemklok van de UNO laten werken op 32MHz in plaats van 16MHz, dan zal alles ook twee keer zo snel gaan. Maar helaas, als het zo simpel was zou de fabrikant de klokfrequentie wel hoger hebben gemaakt. Elke handeling van de microcontroller kost een beetje energie, en resulteert in een beetje warmteontwikkeling. Als de microcontroller twee keer zo veel handelingen verricht, dan zal er ook twee keer zo veel warmte ontstaan. De microcontroller wordt dan te heet en kan gemakkelijk doorbranden.
De Arduino DUE heeft een andere microcontroller (de ATSAM3X8E) die werkt met een klokfrequentie van 84 MHz. Deze Arduino is dus ongeveer 5 keer zo snel als de UNO.

Sketch stroomlijnen

Heb je de sketch niet onnodig lang gemaakt? Zitten er delays in die de sketch vertragen en die korter kunnen worden gemaakt of helemaal weg kunnen?
Overigens maakt het niet uit hoe lang de namen van variabelen zijn. De naam van een variabele heeft geen enkele invloed op de snelheid.

Gebruik het snelste datatype

Uit de metingen blijkt dat het werken met een float iets meer tijd kost dan werken met een long. Ofschoon beide datatypen gebruik maken van vier bytes is het rekenen met floats iets complexer en daarom duurt het langer.
Kijk of je in je sketch wel echt een float nodig hebt. Vaak voldoet een long ook.
Nog sneller is een int (2 bytes) of een byte (1 byte).

Directe poortaansturing

Als het echt veel sneller moet kun je gebruik maken van directe poortaansturing.
100 Keer digitale uitgang 9 hoog en laag maken in een lus duurde 1276 µs.

```
////////////////////////////////////////////////////////////////
  for (int n=0; n<100; n++){      // Totaal 100x aan en uit
  digitalWrite(9,HIGH);
  digitalWrite(9,LOW);}           // 1276 us
////////////////////////////////////////////////////////////////
```

100 Keer digitale uitgang 9 hoog en laag maken in een lus via directe aansturing van de ATmega328 poort duurt maar 38 µs:

```
////////////////////////////////////////////////////////////////
  for (int n=0; n<100; n++){      // Totaal 100x aan en uit
  PORTB = B00000010;
  PORTB = B00000000;}             // 38 us
////////////////////////////////////////////////////////////////
```

Dat is meer dan 30 keer zo snel als digitalWrite!
38 µs x 16 MHz = 608 klokcycli. Één keer hoog en laag duurt dus nog maar 608 / 100 = 6 klokcycli.
Hoe werkt dat?
Om te begrijpen hoe dit werkt moet je eerst weten hoe het binaire stelsel werkt en wat bits en bytes precies zijn. Weet je dat niet, dan moet je eerst hoofdstuk 3 bestuderen.

Alle in- en uitgangen van de Arduino, analoog en digitaal, worden in de microcontroller gestuurd via de bits van drie poorten, poort D, poort B en poort C. Elke poort is één byte (8 bits) groot:

Poort D stuurt digitale pinnen D0 t/m D7, poort B stuurt digitale pinnen D8 t/m D13 en poort C stuurt de pinnen A0 t/m A5 (die niet alleen analoog, maar ook digitale in- of uitgangen kunnen zijn).

Poort	PORT D							
Bit	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Arduino I/O	D7	D6	D5	D4	D3	D2	D1	D0

Poort	PORT B							
Bit	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Arduino I/O			D13	D12	D11	D10	D9	D8

Poort	PORT C							
Bit	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Arduino I/O			A5	A4	A3	A2	A1	A0

Met de opdracht **`PORTB = B00000010`** uit het voorbeeld maken we bit 1 van poort B hoog en de rest van de bits van poort B laag. Deze opdracht doet dus in één keer hetzelfde als:

```
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
digitalWrite(10, LOW);
digitalWrite(11, LOW);
digitalWrite(12, LOW);
digitalWrite(13, LOW);
```

Maar dan vele malen sneller.
Met de opdracht **`DDRB = B00001010`** maken we pin D9 en D11 uitgangen en de rest van de pinnen van poort B ingangen. DDRB staat voor Data Direction Register B.
De opdracht **`DDRB = B00001010`** doet dus hetzelfde als:

```
pinMode(8, INPUT);
pinMode(9, OUTPUT);
pinMode(10, INPUT);
pinMode(11, OUTPUT);
pinMode(12, INPUT);
pinMode(13, INPUT);
```

Maar dan vele malen sneller.
Met de variabele **`PINB`** kunnen we poort B uitlezen, net als met **`digitalRead()`**, maar dan alle in- en uitgangen van poort B tegelijk en - je raadt het al - vele malen sneller.
Met de opdracht **`Serial.print(PINB, BIN)`** wordt de waarde van poort B op de seriële monitor weergegeven als een binaire byte, bijvoorbeeld 00001010 waarbij 0 staat voor LOW, en 1 staat voor HIGH.

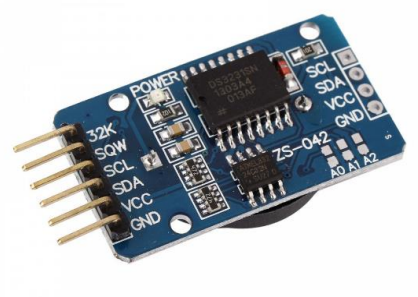
Met behulp van bitmanipulaties kun je afzonderlijke bits veranderen en uitlezen. Dit is precies wat de Arduino functies **`pinMode`**, **`digitalWrite`** en **`digitalRead`** ook doen. Hoe je zelf één enkele digitale in/uitgang van de Arduino met **`bitWrite`** en **`bitRead`** kan bewerken staat in hoofdstuk 3.9, blz. 42.

Je ziet dat het werken met de Arduino opdrachten veel simpeler is dan werken met bitmanipulaties. De Arduino opdrachten doen het moeilijke werk voor je. Dat was ook precies de bedoeling van de makers van de Arduino: een systeem wat zo eenvoudig is dat iedereen het kan gebruiken. Kunstenaars, knutselaars, studenten, scholieren en hobbyisten. Arduinomakers dankjewel! Gelukkig zijn de Arduino opdrachten in de meeste toepassingen snel genoeg.

2.8 Realtime clock

Zodra je de Arduino aan zet begint het tellertje millis() te lopen. Elke milliseconde wordt de teller met 1 opgehoogd. Je kan hier een eenvoudig 24 uurs klokje mee maken maar erg nauwkeurig is dit klokje niet. Het kan makkelijk een minuut per week afwijken. Die afwijking is ook nog eens temperatuur-afhankelijk. Bovendien zal het tellertje millis() na iets minder dan 50 dagen weer bij nul beginnen omdat zijn geheugenplaats dan vol is. Met millis() of delay() kan je een eenvoudig kookwekkertje of een escape-room timer programmeren maar voor een betrouwbare klok is hij niet geschikt.

Als je een nauwkeurige klok of stopwatch wilt maken heb je een RTC-module nodig. Een RTC (RealTime Clock) is een elektronische schakeling die nauwkeurig de tijd bijhoudt. Een ingebouwde temperatuursensor zorgt ervoor dat de RTC kan corrigeren voor de temperatuur. De meeste RTC-modules hebben ook een batterijtje aan boord zodat de klok blijft lopen als de voedingsspanning van de Arduino wegvalt.



Er zijn veel libraries voor de verschillende klokmodules. Een library voor de RTC DS3231 die erg eenvoudig in het gebruik is, is die van Henning Karlsen. Deze kan je downloaden van: http://www.rinkydinkelectronics.com/library.php?id=73
De volgende sketch maakt gebruik van deze library.

```
1  // Sketch 2.10: RTC DS3231 ////////////////////////////////////////////////
2                       // (C)2015 Rinky-Dink Electronics, Henning Karlsen
3  #include <DS3231.h>   // web: http://www.RinkyDinkElectronics.com/
4  DS3231  rtc(SDA, SCL);// Initialiseer de DS3231 met de seriele interface
5
6  void setup(){
7  Serial.begin(9600);     // Start seriele interface
8  //while (!Serial) {}    // Deze regel aan zetten bij gebruik van Leonardo.
9  rtc.begin();            // Start RTC communicatie
10 ///// Klok instellen: Volgende drie regels aanzettn en sketch uploaden ////
11 //rtc.setDOW(MONDAY);       // Dag instellen (Engels, HOOFDLETTERS)
12 //rtc.setTime(14, 07, 00);  // Tijd instellen als 14:07:00 (24hr format)
13 //rtc.setDate(1, 7, 2019);  // Datum instellen als maand, dag, jaar
14 ///// Klok ingesteld? Dan regels weer uitzetten en opnieuw uploaden ///////
15 }
16
17 void loop(){
18   Serial.print(rtc.getDOWStr());     // Dag v/d week ophalen en printen
19   Serial.print(" ");                 // Spaties printen
20   Serial.print(rtc.getDateStr());    // Datum ophalen en printen
21   Serial.print(" ");                 // Spaties printen
22   Serial.println(rtc.getTimeStr()); // Tijd ophalen en printen
23   delay (1000);                      // Seconde wachten
24 }
```

Op de seriële monitor verschijnt de tijd als: **`Monday 01.07.2019 14:07:20`**
Je kan de klok gelijk zetten door de dubbele slash voor de regels 11, 12 en 13 weg te halen, de instellingen in die regels te maken en de sketch uploaden.
Vergeet niet om direct na het uploaden de regels weer uit te zetten en de sketch nog een keer te uploaden. Doe je dat niet, dan zal de tijd steeds opnieuw worden ingesteld op de laatst ingestelde datum en tijd, elke keer als je de Arduino aan zet.
Wil je de uren, minuten en seconden apart kunnen ophalen, probeer dan de voorbeeldsketch DS3231_Serial_Hard. Deze vind je in de libraries folder onder DS3231/examples/arduino of probeer een geheel andere library!

2.9 Energie besparen

Een elektronische schakeling heeft energie nodig om te kunnen werken. Als je de stroom via een adapter uit het stopcontact haalt hoef je je niet zo druk te maken om een paar mA meer of minder, maar als een apparaat op batterijen moet werken, dan wil je dat de batterijen zo lang mogelijk mee gaan. Je wilt de schakeling zo energiezuinig maken als mogelijk is.
Er zijn verschillende maatregelen die je kunt nemen om het energieverbruik van je Arduino te beperken, zoals het verwijderen van overbodige hardware, de voedingsspanning verlagen, de kloksnelheid verlagen of speciale softwarematige aanpassingen. Hoe meer van deze aanpassingen je maakt, des te minder energie je verbruikt, en des te langer je batterijen mee gaan. Het is mogelijk om het stroomverbruik van enkele tientallen mA van een standaard Arduino UNO terug te brengen tot enkele µA!

Overbodige hardware verwijderen

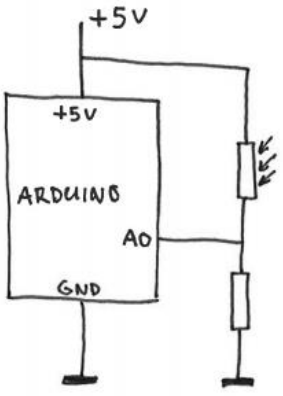
Op het Arduino UNO board zit behalve de ATmega328 chip nog een aantal andere chips die allemaal stroom gebruiken. Sommige hiervan heb je eigenlijk niet nodig en kun je weglaten.
De 9 Volt op de voedingsingang van de Arduino wordt door een spanningsregelaar-chip teruggebracht naar 5 V voor de ATmega328 en de andere chips. Het probleem van de spanningsregelaar is dat hij wel de spanning, maar niet de stroom regelt. Als de uitgang van de spanningsregelaar 100 mA moet leveren, dan zal de 9 V ingang ook 100 mA stroom uit de adapter of batterij trekken. Het 5 V gedeelte van het board gebruikt dan een vermogen van 5 V * 100 mA = 0,5 Watt. Het totale gebruik is 9 V * 100 mA = 0,9 W. Van de 0,9 W gaat 0,4 W verloren in de spanningsregelaar. Dat is 44%! In plaats van een 9V batterij op de 9V ingang kan je dus beter 5 V aan batterijen aansluiten op de 5 V aansluiting van de Arduino.
De led's op het board gebruiken ook relatief veel energie. Tot slot heeft de UNO een aparte chip voor de USB communicatie. Die chip verbruikt ook stroom. De Arduino Pro Mini gebruikt minder stroom dan de UNO, onder andere omdat er geen aparte USB-chip op zit.
Gebruik je in plaats van het UNO board een ATmega328 barebone, dan heb je in één keer alle overbodige hardware verwijderd.
Om het stroomverbruik te meten heb ik in een standaard UNO een minimale sketch ge-upload:

```
1  void setup(){}
2  void loop(){}
```

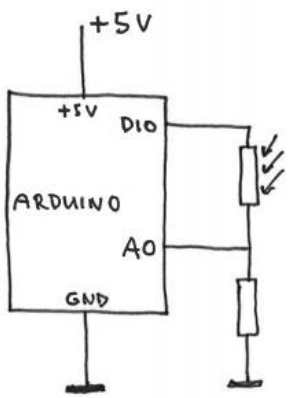
Aangesloten op 5 V gebruikt de UNO met deze sketch 47 mA. De barebone met dezelfde sketch, aangesloten op 5 V gebruikt nog maar 17 mA.

Sensoren uit zetten

Sensoren hoeven meestal maar af en toe te worden uitgelezen.Vaak is één keer per seconde genoeg. Een lichtsensor die ervoor zorgt dat je fietslicht automatisch aan gaat als het donker wordt hoeft maar eens in de minuut te meten. Een sensor die meet of je planten water moeten hebben hoeft maar één of twee keer per dag te meten of de plant een scheut water nodig heeft. Een sensor wordt meestal direct op de 5 V voedingsspanning aangesloten en verbruikt dan continu stroom. Sluit je de voeding van de sensor aan op een digitale uitgang in plaats van op de 5 V, dan kan je in de sketch de sensor kortstondig aan zetten met een digitalWrite, alleen als het nodig is.



LDR aangesloten op 5V



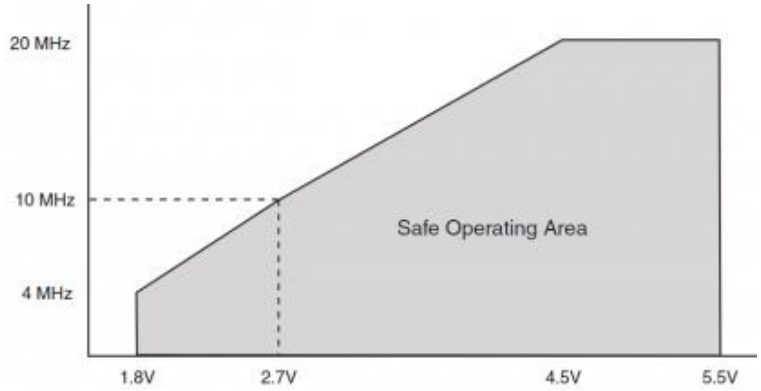
LDR aangesloten op digitale uitgang D10

```
pinMode(10, OUTPUT);

digitalWrite(10, HIGH); // Sensorvoeding aan.
delay(50);              // Trage sensor.
sens = analogRead(0);   // Sensor uitlezen.
digitalWrite(10, LOW);  // Sensorvoeding weer uit.
```

Een fotodiode-sensor is erg snel en kan direct na het aanzetten worden uitgelezen. Een LDR is wat trager en kan pas worden uitgelezen als de voedingsspanning 50 ms aan staat. Je moet dan een delay(50) programmeren als in bovenstaand voorbeeld. Een gassensor moet meestal een halve minuut opwarmen voordat je hem kan uitlezen.

Voedingsspanning verlagen

De stroom die gaat lopen is afhankelijk van de spanning. Hoe lager de spanning, des te lager ook de stroom. Verlaag je de spanning van 5 V naar 3,3 V, dan zal de stroom zakken van 17 mA naar 6,5 mA.
Een 5 V Arduino werkt meestal ook wel op 3,3 V, maar je hebt kans dat het niet goed gaat omdat de kloksnelheid te hoog is. Verlagen van de kloksnelheid zorgt er voor dat het wel werkt. Vandaar dat een 3,3 V Arduino Pro Micro werkt met een kloksnelheid van 8 MHz in plaats van 16 MHz.



Maximale klokfrequentie als functie van de voedingsspanning

System clock prescaler

De klokfrequentie van de ATmega328 chip (Arduino UNO) kan worden verlaagd met de system clock prescaler. Hiermee kan de 16 MHz klokfrequentie worden gedeeld door 2, 4, 8, 16, 32, 64, 128 of 256. Zo ontstaan er klokfrequenties van respectievelijk 8 MHz, 4 MHz, 2 MHz, 1 MHz, 500 kHz, 250 kHz, 125 kHz, en 62,5 kHz.
Het delen van de klokfrequentie gaat met het CLKPR-register.

Om te voorkomen dat het register per ongeluk wordt gezet moet eerst bit 7 van het CLKPR-register op 1 gezet worden, terwijl de andere bits nul zijn. Dan moet binnen 4 klokcycli de prescaler worden ingesteld met bits 0 t/m 3 terwijl bit 7 weer op nul staat.

Het CLKPR-register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Deler	Frequentie
1	0	0	0	0	0	0	0	Enable	
0	0	0	0	0	0	0	1	2	8 MHz
0	0	0	0	0	0	1	0	4	4 MHz
0	0	0	0	0	0	1	1	8	2 MHz
0	0	0	0	0	1	0	0	16	1 MHz
0	0	0	0	0	1	0	1	32	500 kHz
0	0	0	0	0	1	1	0	64	250 kHz
0	0	0	0	0	0	1	1	128	125 kHz
0	0	0	0	1	0	0	0	256	62,5 kHz

Andere getallen dan in bovenstaande tabel staan zijn gereserveerd voor andere functies en mogen niet worden gebruikt! Om er zeker van te zijn dat de procedure goed en snel verloopt mag ze niet worden onderbroken en dus moeten de interrupts tijdelijk worden uitgezet (zie ook 2.4, blz 18).

Het instellen van een clock prescaler van 16 voor een klokfrequentie van 1 MHz ziet er dan zo uit:

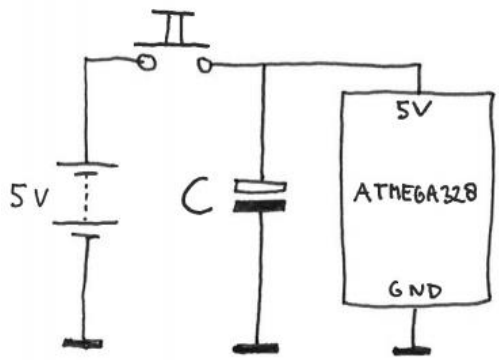
```
void setup(){
  noInterrupts();      // Interrupts uitzetten.
  CLKPR = B10000000;   // Clock prescaler aan zetten.
  CLKPR = B00000100;   // Direct na aanzetten prescaler op 16 (1MHz).
  interrupts();        // Interrupts weer aan zetten.
}
void loop(){}
```

De B in de getallen geeft aan dat het binaire getallen zijn (zie hoofdstuk 3.3, blz. 35).
Met deze sketch in de ATmega328 barebone, aangesloten op een voedingsspanning van 3,3 V loopt er nog maar 1 mA. De voedingsspanning kan nu zefs worden verlaagd tot 2,5 V zodat de chip kan werken op een lithiumbatterij van 3 V.
Als de systeemklokfrequentie wordt gedeeld zullen alle processen langzamer gaan. `delay(1000)` duurt met prescaler 16 dan ook 16 keer zo lang, `millis()` zal elke 16 ms worden opgehoogd. Zelfs de PWM frequentie wordt 16 keer zo klein en servo's werken ook niet meer goed. Hier moet je dus wel rekening mee houden. Een sketch om een led (op pin 13) te laten knipperen met een frequentie van 1 Hz ziet er met een prescaler van 16 zo uit:

```
// Sketch 2.11: Clock Prescaler ////////////////////////////////////
void setup(){
  noInterrupts();      // Interrupts uitzetten.
  CLKPR = B10000000;   // Clock prescaler aan zetten.
  CLKPR = B00000100;   // Direct na aanzetten prescaler op 16 (1MHz).
  interrupts();        // Interrupts weer aan zetten.
  pinMode(13, OUTPUT);
}
void loop(){
  digitalWrite (13, HIGH);
  delay(31);                // Geeft een delay van 31 * 16 = 496 ms.
  digitalWrite (13, LOW);
  delay(31);
}
```

Een elco als oplaadbare batterij

Een electrolytische condensator (elco) is vergelijkbaar met een oplaadbare batterij die supersnel oplaadt en maar een kleine capaciteit heeft. Heb je een toepassing waarbij je heel af en toe door middel van een drukknop een Arduino een paar seconden lang wilt laten werken, dan is onderstaande schakeling misschien geschikt.



Als je één keer kort op de drukschakelaar drukt wordt elco C opgeladen en deze kan vervolgens de Arduino een paar seconden voeden. Als de schakelaar niet wordt ingedrukt verbruikt de schakeling helemaal geen stroom. Hoe groot de elco moet zijn is afhankelijk van het stroomverbruik van de Arduino en hoe lang hij aan moet blijven staan. Hoe langer de Arduino moet werken en hoe hoger

het stroomverbruik, des te groter de capaciteit van de elco moet zijn. Met een elco van 1000 µF en 5 V batterijen kan een barebone die 0,3 mA gebruikt ongeveer 10 seconden werken. Tijdens die tien seconden zakt de voedingsspanning van 5 V naar 3 V en dus wordt het uitlezen van een analoge sensor erg onbetrouwbaar. De capaciteit kan worden verhoogd door meerdere condensatoren parallel te schakelen (niet in serie zoals batterijen!). Let op: een elco heeft een plus- en een min-kant.

Sleep modes

Met sleep modes kunnen delen van de processor zoals de AD-converter en de verschillende timers die je niet gebruikt tijdelijk worden uitgezet. Er zijn 6 verschillende modes die elk verschillende onderdelen uit zetten en elk een andere besparing opleveren. De power-down mode zet bijna alle onderdelen uit en geeft dan ook de grootste besparing. De sleep modes kunnen worden ingesteld en geactiveerd via directe sturing van de betreffende poorten in de ATmega328 maar er zijn ook libraries voor. In het volgende voorbeeld gebruiken we de avr/sleep library. Deze library zit standaard in de Arduino IDE.
Onderstaande sketch is ge-upload naar de Arduino UNO en daarna is de geprogrammeerde ATmega328 chip als barebone op een breadboard opgebouwd met een schakelaar tussen pin 2 en GND en een LED met een weerstand van 220Ω op pin 13.

```
1  // Sketch 2.12: Sleep ////////////////////////////////////////////////////////
2  #include <avr/sleep.h>                    // De avr/sleep library is nodig.
3
4  void setup(){
5    noInterrupts();       // Interrupts uitzetten.
6    CLKPR = B10000000;    // Clock prescaler aan zetten.
7    CLKPR = B00000100;    // Direct na aanzetten prescaler op 16 (1MHz).
8    interrupts();         // Interrupts weer aan zetten.
9    pinMode(2, INPUT_PULLUP);              // pin 2 is om chip te wekken.
10   pinMode(13, OUTPUT);                   // LED op pin 13
11   attachInterrupt(0, wakeUp, LOW);       // interrupt als pin 2 laag wordt.
12   set_sleep_mode(SLEEP_MODE_PWR_DOWN);   // sleep mode: POWER DOWN.
13 }
14
15 void loop(){
16   digitalWrite(13, HIGH); // LED aan.
17   delay(312);             // Wacht 5 tellen om de stroom te meten: 13,8 mA.
18   digitalWrite(13, LOW);  // LED uit.
19   delay(312);             // Wacht 5 tellen om de stroom te meten: 9,3 mA.
20   sleep_enable();          // Is nodig om de sleep_mode te kunnen activeren.
21   sleep_mode();            // Activeert sleep_mode; chip gaat in slaap!!
22                            // Meet de stroom: 0,15 mA!
23   sleep_disable();         // Voorkomt dat de sleep_mode geactiveerd wordt.
24 }
25
26 void wakeUp(){}            // Interrupt routine is nodig (mag leeg zijn).
```

De de chip kan uit de power-down modus worden gehaald door een externe interrupt met het schakelaartje op pin 2.
Met de opdracht **`set_sleep_mode(SLEEP_MODE_PWR_DOWN)`** in regel 12 wordt de power-down mode ingesteld.
De sketch zet eerst de LED 5 seconden aan zodat de stroom kan worden gemeten. Deze blijkt 13,8 mA te zijn. Als de LED vervolgens 5 seconden uit is meten we 9,3 mA. Blijkbaar verbruikt de LED 4,5 mA.
Om de chip in slaap te krijgen moet eerst een **`sleep_enable()`** opdracht worden gegeven. Daarna kan de chip in de PWR_DOWN slaapstand worden gezet met **`sleep_mode()`**. In de slaapstand loopt er nog maar 0,15 mA.
De chip blijft in slaap tot de schakelaar wordt ingedrukt. Dan worden de opdrachten in de ISR **`void wakeUp()`** uitgevoerd en daarna gaat het programma verder vanaf regel 22.

Met een prescaler van 16 (1 MHz systeemklok) en een voedingsspanning van 3 V loopt er tijdens de sleep mode nog maar 90 µA.

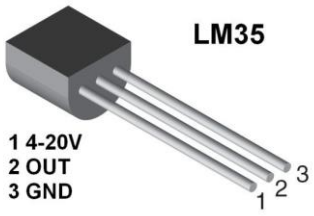
Een low power library

Er zijn libraries die het programmeren van sleep modes makkelijk maken. De LowPower library geeft de mogelijkheid om de microprocessor uit de slaapstand te halen met een timer. Hiermee kan je een low-power delay programmeren.
https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption

2.10 De LM35 temperatuursensor

De LM35 is een lineaire temperatuursensor met een gevoeligheid van 10 mV/°C. Dat betekent dat de uitgangsspanning 10 mV (millivolt) hoger wordt bij elke °C temperatuurstijging. Bij 0°C is de uitgangsspanning 0 V.
In een formule:

Uitgangsspanning (mV) = 10 x Temperatuur (°C)



Omdat de sensor lineair is, is hij eenvoudig te ijken met de Arduino `map` functie.
De analoog-digitaal converter (ADC) in de Arduino UNO verdeelt de ingangsspanning van 5 V in 1023 gelijke stapjes. Bij een spanning van 5 V is de ADC waarde 1023. Bij deze spanning is de temperatuur volgens de formule 5000(mV) / 10 = 500(°C).
De ijking van een LM35 op analoge ingang A0 wordt dan:

0 tot 1023 van de ADC wordt geschaald naar 0 tot 500°C.

```
graadCelsius = map(analogRead(0), 0, 1023, 0, 500);
```

De hele sketch van een LM35 thermometer die wordt uitgelezen met de seriële monitor wordt dan:

```
1  // Sketch 2.13: LM35 Thermometer ////////////////////////
2
3  void setup() {
4    Serial.begin(9600);       // Gebruik de seriële monitor.
5  }
6
7  void loop() {
8    int graadCelsius = map(analogRead(0), 0, 1023, 0, 500);
9    Serial.println(graadCelsius);
10   delay(1000);
11 }
```

Vragen en opdrachten

2.11 Sluit een LM35 sensor aan met de buitenste pinnen op 5V en GND volgens bovenstaande tekening en de middelste pin op analoge ingang A0 van de Arduino.
2.12 Upload sketch 2.13 en controleer met de seriële monitor of je thermometer werkt.

2.11 Nauwkeuriger meten met Aref

De LM35 thermometer is geijkt als de integer graadCelsius. Dat betekent dat het een geheel getal is en dus kan de temperatuur alléén worden gemeten in hele graden. Zelfs als de temperatuur wordt gedefiniëerd als een float, dan nog is de resolutie beperkt omdat het bereik van 500°C wordt verdeeld in 1023 stappen. De kleinste stap is dan 500°C / 1023 = 0,489°C.
Het bereik van de LM35 is 0-140°C en we willen de thermometer gebruiken om temperaturen tot maximaal 100°C te meten. We zouden de resolutie kunnen vergroten als we het bereik van 100°C kunnen verdelen in 1023 stappen van 100 / 1023 = 0,098°C. Dan kunnen we een thermometer maken die de temperatuur met één cijfer achter de komma weergeeft.
We kunnen dit doen met de analoge referentiespanning (Aref). De analoge referentiespanning is normaal 5 volt. Met de opdracht `analogReference(INTERNAL)` kunnen we een

referentiespanning van 1,1 V instellen. Een spanning van 1,1 V op een analoge ingang geeft dan een ADC waarde van 1023. Analoge spanningen van meer dan 1,1 V kunnen dan uiteraard niet meer worden gemeten. De opdracht `analogReference(INTERNAL)` moet natuurlijk vóór de `analogRead` opdracht staan, bijvoorbeeld in de setup().
Een spanning van 1,1 V komt met de LM35 overeen met een temperatuur van 110°C dus de thermometer kan dan tot maximaal 110,0°C meten.
Omdat de `analogRead` en `map` functies werken met integers moeten we eerst van de `analogRead` waarde een `float` maken, en dan een float ijkfunctie maken zonder de `map` opdracht:

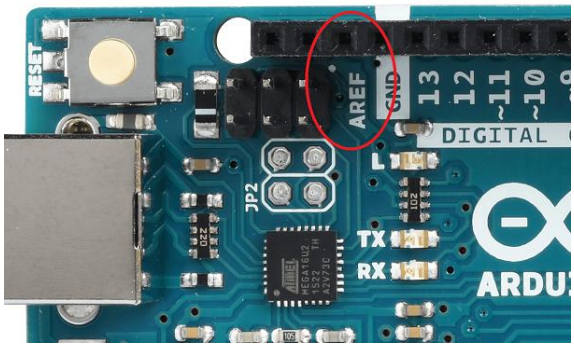
```
float LM35 = analogRead(0);
float graadCelsius = (LM35*110)/1023;
```

```
1  // Sketch 2.14: LM35 Thermometer met Aref ////////////////////
2
3  void setup() {
4    Serial.begin(9600);          // Gebruik de seriële monitor.
5    analogReference(INTERNAL);   // Aref = 1,1 volt.
6  }
7
8  void loop() {
9    float LM35 = analogRead(0);              // LM35 op pin A0.
10   float graadCelsius = (LM35*110)/1023;    // IJken.
11   Serial.println(graadCelsius, 1);         // met 1 decimaal.
12   delay(1000);
13 }
```

Zie blz. 38 voor meer informatie over float. De interne analoge referentiespanning is niet bij alle Arduino boards hetzelfde. Gebruik je een ander board dan de UNO, check dit dan voordat je de functie gebruikt:
https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference/

Externe Aref

In plaats van de interne analoge referentiespanning kan je ook een externe referentiespanning instellen met de opdracht `analogReference(EXTERNAL)`. Deze externe spanning kan je maken met een spanningsdeler van twee weerstanden of een instelpot, en aanbieden op de Aref-pin van de Arduino. De externe referentiespanning moet minimaal 0,7 V zijn, en maximaal de boardspanning van 5V.



Let op! Geef de opdracht `analogReference(EXTERNAL)` vóórdat je de `analogRead` opdracht geeft, anders sluit je de interne referentiespanning kort met de externe referentiespanning en daardoor kan de microcontroller stuk gaan.

Vragen en opdrachten

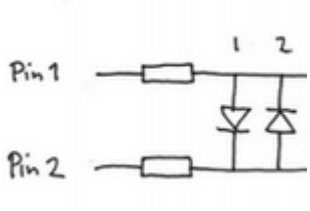
2.13 Bouw een thermometer met een nauwkeurigheid van 0,1 °C.
De temperatuur moet worden uitgelezen op een 16x2 LCD display.
Met een drukschakelaartje moet je de eenheid kunnen instellen op °C, °F of K.

2.12 Charlieplexing: 20 LEDs aansturen met 5 pinnen

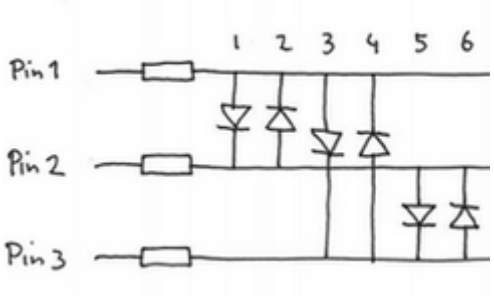
Normaal wordt een LED samen met een serieweerstand tussen een digitale uitgang van de Arduino en GND geschakeld. Op die manier kan je met 10 digitale uitgangen ook 10 LEDs aansturen. Met een techniek die Charlieplexing wordt genoemd kan je met 10 digitale pinnen van de Arduino zonder extra hardware 90 LEDs aansturen! Handig voor bijvoorbeeld een digitale windroos, roulette, rad van avontuur, pim-pam-pet of een superlang looplicht. Nadeeltje: er kan maar één LED tegelijk branden.
Het aantal LEDs dat je kunt gebruiken en het aantal pinnen wat daarvoor nodig is wordt gegeven door de formule: $L = P^2 - P$ waarin L = aantal LEDs en P = aantal pinnen.

Pins	P	2	3	4	5	6	7	8	9	10	14
LEDs	$P^2 - P$	2	6	12	20	30	42	56	72	90	182

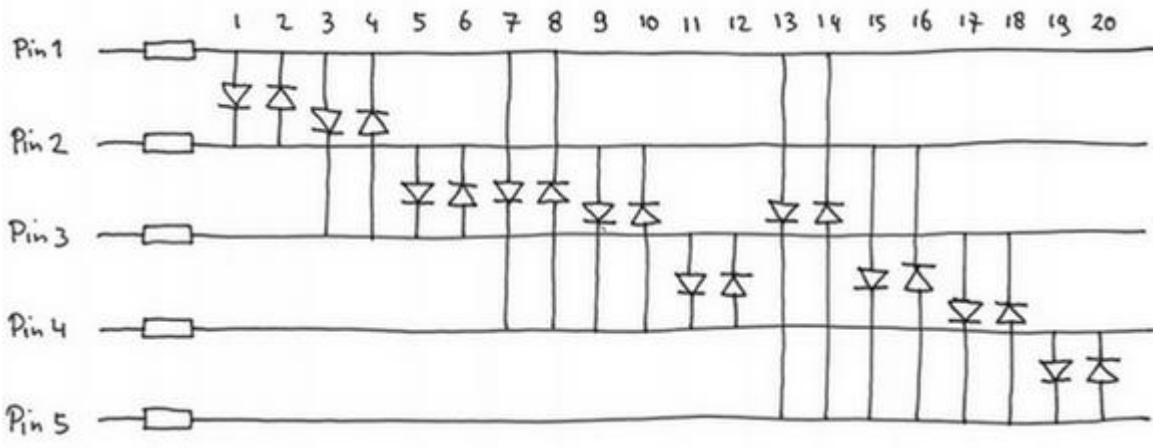
Met charlieplexing schakel je de LEDs niet tussen tussen de digitale pinnen en GND, maar tussen de digitale pinnen onderling. Door één pin hoog en één pin laag te maken kan je de gewenste LED aansturen.



Als in bovenstaand voorbeeld pin 1 hoog is en pin 2 laag, dan brandt LED 1. Als pin 1 laag is en pin 2 hoog, dan brandt LED 2. Met 2 pinnen heeft charlieplexing geen zin omdat je er maar twee LEDs mee kan aansturen. Het wordt pas interessant met meerdere pinnen.
Met 3 pinnen kan je 6 LEDs aansturen:



Als pin 1 hoog is, en pin 3 laag, dan brand LED 3. Pin 2 mag nu niet meedoen want als pin 2 hoog is, dan brandt ook LED 5 of als pin 2 laag is brandt ook LED 1. Pin 2 moet worden uitgeschakeld door er een input van te maken. In de sketch wordt dus steeds één pin hoog gemaakt, één pin laag en de overige pinnen moeten digitale input worden. Met 20 LEDs op 5 pinnen ziet de schakeling er zo uit:



Een LED heeft hier steeds twee serieweerstanden dus als de LED normaal brand met één serieweerstand van 200Ω, dan moeten de weerstanden in de charlieplex-schakeling 100Ω zijn.

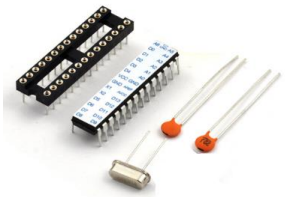
In de sketch ziet het er zo uit:

```
// Sketch 2.15: Charlieplexing ////////////////////////////////////////
byte pin1 = 8;        // Charlieplex pin 1 op Arduino pin 8
byte pin2 = 9;        // Charlieplex pin 2 op Arduino pin 9
byte pin3 = 10;       // Charlieplex pin 3 op Arduino pin 10
byte pin4 = 11;       // Charlieplex pin 4 op Arduino pin 11
byte pin5 = 12;       // Charlieplex pin 5 op Arduino pin 12
byte LED = 0;         // Led die moet worden aangezet

void setup(){}

void loop(){
    LED++;                          // Volgende LED
    if(LED>20)LED=1;                // Maximaal 20 LEDs
    setCharliePins(LED);            // Pinnen aan en uit zetten
    delay(250);                     // Langzaam alle 20 doorlopen
}

void setCharliePins(byte LED){
  pinMode (pin1, INPUT);            // Eerst alle pins INPUT maken
  pinMode (pin2, INPUT);
  pinMode (pin3, INPUT);
  pinMode (pin4, INPUT);
  pinMode (pin5, INPUT);
  switch (LED){
    case 1:                                                 // LED 1
      pinMode (pin1, OUTPUT); digitalWrite (pin1, HIGH); // pin1 HIGH
      pinMode (pin2, OUTPUT); digitalWrite (pin2, LOW);  // pin2 LOW
    break;
    case 2:                                                 // LED 2
      pinMode (pin1, OUTPUT); digitalWrite (pin1, LOW);  // pin1 LOW
      pinMode (pin2, OUTPUT); digitalWrite (pin2, HIGH); // pin2 HIGH
    break;
    case 3:                                                 // LED 3
      pinMode (pin1, OUTPUT); digitalWrite (pin1, HIGH); // pin1 HIGH
      pinMode (pin3, OUTPUT); digitalWrite (pin3, LOW);  // pin3 LOW
    break;
    // Enzovoort, tot en met LED 20
    case 20:                                                // LED 20
      pinMode (pin4, OUTPUT); digitalWrite (pin4, LOW);  // pin4 LOW
      pinMode (pin5, OUTPUT); digitalWrite (pin5, HIGH); // pin5 HIGH
    break;
  }
}
```

2.13 De ATmega328 barebone achterhaald?

Mijn eerste Arduino UNO was voorzien van een ZIF voetje voor de ATmega328 chip. Een ZIF-voetje (Zero Insertion Force socket) heeft een hendeltje waarmee je de pootjes van de chip kunt vastklemmen en weer los maken. Met dit voetje kan de ATmega328 snel en gemakkelijk op het Arduino board worden gezet en weer los gehaald zonder de pootjes krom te buigen. Dat was erg handig omdat ik bij mijn projecten meestal gebruik maakte van een barebone.

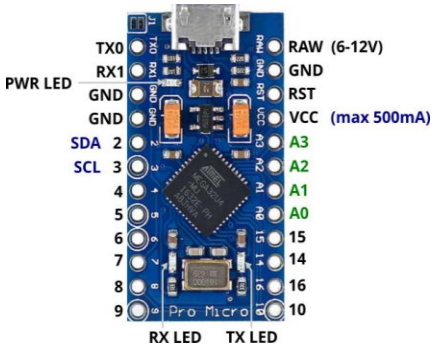
Arduino UNO met ZIF-socket



ATmega328 barebone

Arduino Pro Micro

Tegenwoordig maak ik nog maar zelden iets met een barebone. Nu (2019) gebruik ik vaak een Arduino Pro Micro. Deze is met gesoldeerde pinnen ongeveer even groot als de 328 barebone. De Pro Micro met pinnen weegt 4,0 gram, de 328 barebone weegt 4,6 gram. De Pro Micro heeft minder in- en uitgangen, maar meestal heb je er niet zo veel nodig. Als je je prototype wilt omzetten naar een permanent project, kijk dan eerst even of je de Micro Pro kunt gebruiken. Met aangesoldeerde pinnen kun je hem ook in je breadboard prikken om een prototype te maken. De Pro Micro is o.a. zo klein omdat hij gebruik maakt van de ATmega32U4 microcontroller. Deze chip heeft een ingebouwde USB interface. Bij de Arduino UNO zit de USB interface in een aparte chip op het board. Als je een sketch maakt voor de Pro Micro, vergeet dan niet in de IDE bij hulpmiddelen/board het Leonardo board te kiezen. De Pro Micro staat niet in de lijst maar hij heeft dezelfde 32U4 chip als de Leonardo.



3 Wiskunde

3.1 Getallenstelsels

Als kleuter leerde je al tot tien tellen. We gebruiken de hele dag door getallen, van schoenmaat tot televisiekanaal en van afstanden in kilometers tot prijzen in euro's. We maken daarbij gebruik van het tientallig (decimaal) stelsel.
Computers en andere digitale elektronica zoals de Arduino werken met het tweetallig (binair) stelsel. Zij maken diep van binnen gebruik van slechts twee cijfers, 0 en 1, omdat de kleinste eenheid van een computerchip, de transistor, maar twee dingen kan: wél of géén stroom doorlaten.

3.2 Decimaal stelsel

Het decimale stelsel heeft 10 cijfers: 0, 1, 2, 3, 4, 5, 6, 7, 8 en 9. Een decimaal getal is opgebouwd uit deze cijfers. Omdat het tien cijfers zijn, is het decimale stelsel opgebouwd met machten van tien.

$10^0=1$ $10^1=10$ $10^2=100$ $10^3=1000$ $10^4=10000$ $10^5=100000$ $10^6=1000000$ enz.

Het getal **214** is opgebouwd uit **2** x 10^2 + **1** x 10^1 + **4** x 10^0 = **2**00 + **1**0 + **4**

3.3 Binair stelsel

In het binaire (tweetallig) stelsel bestaan maar twee cijfers: **0** en **1**. Een binair cijfer wordt een bit genoemd (uit het engels binairy digit, wat binair cijfer betekent). Het binaire stelsel is opgebouwd met machten van twee.

$2^0=1$ $2^1=2$ $2^2=4$ $2^3=8$ $2^4=16$ $2^5=32$ $2^6=64$ $2^7=128$ $2^8=256$ enz.

Een binair getal omrekenen naar een decimaal getal gaat als volgt:
Als voorbeeld het binaire getal **10110110**. Dit is opgebouwd uit

1 x 2^7 + **0** x 2^6 + **1** x 2^5 + **1** x 2^4 + **0** x 2^3 + **1** x 2^2 + **1** x 2^1 + **0** x 2^0 =

128 + 0 + 32 + 16 + 0 + 4 + 2 + 0 = 182 (decimaal)

Omrekenen van decimaal naar binair kan ook. Bijvoorbeeld het decimale getal **214**

Zoek eerst de grootste macht van 2 die in 214 past. Dat is 2^7 (128). Het binaire getal wordt dan 8 bits lang want 2^7 is het 8ste bit van rechts.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1							

214 - 128 = 86 De rest is 86. De grootste macht van 2 die daar in past is 64, dus het volgende bit is ook 1.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**1**						

214 -128 - 64 = 22 De rest is nu 22. De grootste macht van 2 die daar in past is 16:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**1**		**1**				

214 -128 - 64 - 16 = 6 Hier past 4 als grootste in:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**1**		**1**		**1**		

214 - 128 - 64 - 16 - 4 = 2 Hier past 2 precies in. We weten nu welke bits 1 zijn. De overige zijn 0.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**1**	**0**	**1**	**0**	**1**	**1**	**0**

214 decimaal = **11010110** binair.

De Atmega328 microcontroller (Arduino UNO) heeft een 8 bits processor. Dat betekent dat hij met getallen van 8 bits werkt. Een geheugenplaats (geheugencel) in deze chip beslaat ook 8 bits.
Een getal van 8 bits noemen we een byte. Een byte kan 256 verschillende waarden hebben, van 0 t/m 255.

00000000 - **11111111** binair = **000** - **255** decimaal

Omdat de microcontroller met 8 bits werkt kom je het getal 255 dus ook vaak tegen.
Een variabele van het type `byte` kan een waarde hebben van 0 tot 255.
De PWM-waarde in `analogWrite` kan een waarde van 0 tot 255 hebben.
De A/D omzetter die de analoge waarde van `analogRead` omzet in een getal werkt met 10 bits. Daardoor is het bereik verdeeld in 2^{10} = 1024 stapjes, van 0 tot 1023.
De variabele van het type `int` gebruikt 2 bytes en heeft daardoor een bereik van 256 x 256, dus 65536 waarden. Omdat `int` zowel positief als negatief kan zijn loopt het bereik van - 32768 tot +32767. Als je alleen positieve getallen nodig hebt kun je beter een `unsigned int` gebruiken (unsigned betekent zonder min-teken). Deze heeft een bereik van 0 - 65535.
De variabele van het type `long` is 4 bytes (32 bits) groot en heeft daardoor een bereik van -2147483648 tot 2147483647.
Een `unsigned long` heeft een bereik van 0 - 4294967295 (2^{32} - 1).

Je kunt binaire getallen ook in je sketch gebruiken. Je moet dan een B voor het binaire getal zetten. Bijvoorbeeld **B11010110**. Het binaire getal mag maximaal 8 bits (1 byte) lang zijn.
Met onderstaande sketch kan je binaire getallen omrekenen naar decimaal, en met de seriële monitor het resultaat bekijken:

```
1  // Binair omrekenen //
2  void setup(){
3    Serial.begin(9600);
4  }
5  void loop(){
6    Serial.println(B1);
7    Serial.println(B10);
8    Serial.println(B00000001);
9    Serial.println(B11010110);
10   Serial.println(B11111111);
11   delay(10110);
12 }
```

COM8 (Arduino

```
1
2
1
214
255
1
2
```

Je kan ook een decimaal getal omrekenen naar een binair getal:

```
Serial.print(214, BIN); // Print het getal als 11010110
```

Er zijn 10 soorten mensen, mensen die wél en mensen die niet binair kunnen tellen.

Vragen en opdrachten

3.1 Schrijf de binaire getallen van 0000 tot 1111 achter elkaar op.
3.2 Probeer uit je hoofd hardop binair te tellen van 0000 tot 1111.
3.3 Bereken je binaire leeftijd.

3.4 De schrijver van dit boek is geboren in het binaire jaar 11110100111.
Hoe oud wordt hij (decimaal) in het jaar 2059?

3.4 Hexadecimaal stelsel

Het hexadecimaal stelsel werkt met 16 cijfers en is dus opgebouwd met machten van 16.

$16^0=1$ $16^1=16$ $16^2=256$ $16^3=4096$ $16^4=65536$ $16^5=1048576$

Omdat we maar 9 cijfers hebben worden voor de cijfers 10 t/m 15 de letters A t/m F gebruikt. Een hexadecimaal getal bestaat dus uit de cijfers **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **A**, **B**, **C**, **D**, **E**, **F**.
Voor de cijfers A t/m F mag je ook de kleine letters gebruiken.
Het hexadecimale getal **3FB1** is opgebouwd uit:

3 x 16^3 + **F** x 16^2 + **B** x 16^1 + **1** x 16^0

12288 + 3840 + 176 + 1 = 16305 (decimaal)

Omdat 16 ook een macht van 2 is, is het heel eenvoudig om een hexadecimaal getal om te rekenen naar een binair getal en omgekeerd. Elk hexadecimaal cijfer is een 4 bits binair getal. Je kan elk hexadecimaal cijfer van een hexadecimaal getal apart omrekenen en de bits achter elkaar zetten.

hexadecimaal	**3**				**F**				**B**				**1**			
decimaal	**3**				**15**				**11**				**1**			
binair	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**

dus **3FB1** hexadecimaal = **0011111110110001** binair.
Als je een binair getal wilt omrekenen naar een hexadecimaal getal, dan moet je er eerst voor zorgen dat het aantal cijfers een veelvoud is van 4 door er zo nodig nullen voor te zetten.
11111110110001 heeft maar 14 bits, dus daar moet je eerst 16 bits van maken door er twee nullen voor te zetten, dus **0011111110110001** en daarna in groepjes van 4 bits omrekenen naar **3FB1**.
Programmeurs die zich op een laag niveau met de microcontrollerchip bezighouden programmeren meestal in hexadecimale code, omdat dit overzichtelijker is en minder fouten geeft dan de lange binaire getallen.
Je kunt hexadecimale getallen ook in je sketch gebruiken. Je moet dan 0x (nul x) voor het getal zetten. Bijvoorbeeld **0x3FB1**. Het hexadecimale getal mag maximaal 8 cijfers (4 bytes) lang zijn.
Met onderstaande sketch kan je hexadecimale getallen omrekenen naar decimaal, en met de seriële monitor het resultaat uitlezen:

```
1  // Hexadecimaal omrekenen //
2  void setup(){
3    Serial.begin(9600);
4  }
5  void loop(){
6    Serial.println(0x1);
7    Serial.println(0x10);
8    Serial.println(0xF);
9    Serial.println(0xFF);
10   Serial.println(0x3FB1);
11   Serial.println(0xFFFFFFFF);
12   delay(0x3E8);
13 }
```

```
COM8 (Arduino
4294967295
1
16
15
255
16305
4294967295
1
16
```

Je kan ook een decimaal getal omrekenen naar een hexadecimaal getal:

```
Serial.print(16305, HEX); // Print het getal als 3FB1
```

Vragen en opdrachten

3.5 De delay in bovenstaande sketch bedraagt 0x3E8. Hoeveel milliseconden is dat decimaal?
3.6 Een binair getal heeft 24 bits. Hoeveel cijfers heeft dit getal in hexadecimaal.
3.7 Een hexadecimaal getal begint met een F en heeft 8 cijfers. Hoeveel bits heeft dit getal in binair?
3.8 Een hexadecimaal getal begint met een 2 en heeft 8 cijfers. Hoeveel bits heeft dit getal in binair?

3.5 Octaal stelsel

Het octaal stelsel werkt met 8 cijfers (0 t/m 7) en is dus opgebouwd met machten van 8.
Hoe het octaal stelsel werkt en hoe je octaal telt kan je nu zelf bedenken. Het octaal stelsel wordt niet vaak meer gebruikt. Wil je een octaal getal in je sketch gebruiken, dan moet je er een nul voor zetten. Bijvoorbeeld **0362** = octaal **362** = decimaal 242.

Wist je dat?

10^3 = 1000 decimaal
2^3 = 1000 binair
16^3 = 1000 hexadecimaal
8^3 = 1000 octaal
x^3 = 1000 in het x-tallig stelsel

3.6 Data types

int

De integer is waarschijnlijk het meest gebruikte datatype in Arduino sketches. Een integer is een geheel getal. Het bereik loopt van -32768 tot 32767.
Als je buiten dit bereik komt, bijvoorbeeld als gevolg van een berekening, dan krijg je een effect dat 'rolling over' wordt genoemd. Kom je boven de maximum waarde, dan rolt de int over naar de miniumum waarde en omgekeerd. Dus: 32767 + 1 = -32768 en -32768 - 1 = 32767.
Optellen, aftrekken en vermenigvuldigen met integers gaat goed, maar delen kan problemen met afrondingen geven omdat alles achter de komma wegvalt:

```
int x = 10;
int y = x / 3;  // y krijgt de waarde 3 i.p.v. 3,33
```

Als je cijfers achter de komma nodig hebt moet je het datatype float gebruiken.
Een int wordt opgeslagen als twee bytes (16 bits). Hiervoor zijn twee geheugenplaatsen in de Arduino nodig.

unsigned int

Als je alléén positieve getallen gebruikt kan je ook unsigned int gebruiken. Unsigned betekent dat er geen min-teken gebruikt kan worden. Het bereik van een unsigned int is van 0 – 65535 (2^{16}-1).
Een unsigned int gebruikt net als een int twee bytes (16 bits) geheugenruimte

byte

Een byte is een geheel positief getal met een bereik van 0-255 (2^8-1). Een byte wordt opgeslagen als 8 bits getal. Hiervoor is één geheugenplaats in de Arduino nodig. Berekeningen met bytes resulteren net als bij een int in gehele getallen. Een byte heeft ook het 'rolling over' effect.

long

Heb je grotere gehele getallen nodig, gebruik dan een long. Een long is 4 bytes groot en heeft daardoor een bereik van -2147483648 tot 2147483647. Verder werkt het net als een int.

unsigned long

Een unsigned long is een geheel getal en heeft een bereik van 0 – 429976295 (2^{32}-1).

float

Een float (van het Engels floating point) is een getal met cijfers achter de komma. In plaats van een komma gebruiken we een punt (engelse notatie). Het bereik loopt van -3.4028235 x 10^{38} tot

3.4028235 x 10^{38}. Een float gebruikt 4 bytes (32 bits) en heeft een nauwkeurigheid van ongeveer 6 significante cijfers.

```
Serial.begin(9600);
float x = 10;
float y = x / 3;
Serial.print(y);  // Resultaat: 1.33
```

Je kan het aantal cijfers achter de komma aangeven met (y, decimalen).

```
Serial.begin(9600);
float x = 10;
float y = x / 6;
Serial.print(y, 4);  // Resultaat: 1.6667

float z = 3.76;
Serial.print(z, 0);  // Resultaat: 4
```

Als je het aantal decimalen niet opgeeft wordt als default 2 decimalen gegeven.

double

Een double gebruikt normaal 8 bytes (64 bits) en is daarmee twee keer (double) zo nauwkeurig (tot 15 significante cijfers) als een float. Maar dit geldt (voorlopig, 2019) alleen voor de Arduino DUE. Op alle andere Arduino boards is een double hetzelfde als een float, met 32 bits precisie.

bool

Een bool kan slechts twee waarden hebben, TRUE of FALSE. In plaats van TRUE en FALSE gebruiken we meestal 1 en 0. In plaats van TRUE of 1 mogen we zelfs elke getal behalve nul gebruiken.

```
bool x = TRUE;
bool y = FALSE;
bool a = 1;       // Het zelfde als TRUE
bool b = 0;       // Het zelfde als FALSE
bool c = -273     // Het zelfde als TRUE of 1

Serial.begin(9600);
bool x = -487510;
Serial.print(x);     // Resultaat: 1
```

Een bool heeft eigenlijk een grootte van één bit. Toch wordt een boole opgeslagen als een byte. Een byte (8 bits) is de kleinste eenheid van het geheugen.
Bij vergelijkingen met een boole mogen we de opdracht inkorten.

```
bool a;
if(a == 1)    // in plaats van deze vergelijking
if(a)         // mogen we hem ook inkorten
```

boolean

Een boolean is hetzelfde als een bool. Het is beter om bool te gebruiken.

short

Een short is hetzelfde als een int. Het is beter om int te gebruiken.

word

Een word is hetzelfde als een unsigned int. Het is beter om unsigned int te gebruiken.

char

Een char is een karakter uit de ASCII-tabel (zie tabel 7.1, blz. 61). Een variabele van het type char wordt opgeslagen als een 8 bit getal. De letter A wordt opgeslagen als 65.
`char karakter = 'A'` is hetzelfde als `char karakter = 65`.

String
Een string is een reeks karakters. Strings zijn handig wanneer je complete zinnen of woorden wilt gebruiken. Je kunt ook van een variabele een string maken, bijvoorbeeld om naar een display te printen. De vele manieren waarop je een String kan gebruiken zijn te vinden op de Arduino reference pagina's:
https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/

```
String stringOne = "Hello String";                   // using a constant String
String stringOne = String('a');                      // converting a constant char into a String
String stringTwo = String("This is a string");       // converting a constant string into a String object
String stringOne = String(stringTwo + " with more"); // concatenating two strings
String stringOne = String(13);                       // using a constant integer
String stringOne = String(analogRead(0), DEC);       // using an int and a base
String stringOne = String(45, HEX);                  // using an int and a base (hexadecimal)
String stringOne = String(255, BIN);                 // using an int and a base (binary)
String stringOne = String(millis(), DEC);            // using a long and a base
String stringOne = String(5.698, 3);                 // using a float and the decimal places
```

const
Met de toevoeging const (van constant) wordt aangegeven dat het getal niet kan worden veranderd. Dit in tegenstelling tot een variabele. Const kan voor elk datatype worden gebruikt, zoals const int, const byte, const long enzovoort. Omdat het gebruik van de Arduino pinnen in de hardware meestal niet verandert kunnen ze als `const int` of `const byte` worden gedefiniëerd. Het voordeel hiervan is dat je ze ook niet per ongeluk kan veranderen.

define
Het definiëren van constanten kan ook met de opdracht #define vóór het datatype.
`#define int LED 7` doet hetzelfde als `const int LED = 7;`.
Achter de #define opdracht komt geen `;` . `#define int LED = 7` (met een = teken) mag ook niet. Nogal verwarrend. Je kan `#define` beter niet gebruiken. Gebruik liever `const`. Een sketch met #define is waarschijnlijk gemaakt door een programmeur die gewend is om in de programmeertaal C te programmeren.

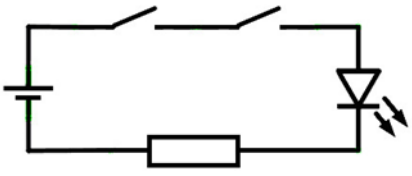
uint8_t
Unsigned integer met een lengte van 8 bits. Eigenlijk dus hetzelfde als byte. Als je programmeert voor verschillende platforms gebruik je uint8_t omdat dit in de meeste dialecten van de taal C werkt. Byte werkt alléén in de Arduino sketch.

uint16_t
Unsigned integer met een lengte van 16 bits. Hetzelfde als een unsigned int.

volatile
De toevoeging volatile zorgt er voor dat de variabele niet in een tijdelijk register wordt opgeslagen, maar in het RAM. Dit is nodig als je interrupts gebruikt (zie hoofdstuk 2.4 blz. 18).

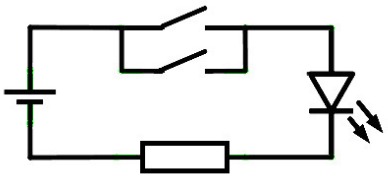
3.7 Booleaanse algebra
De Engelse wiskundige George Boole leefde van 1815 tot 1864. Hij bedacht een algebra waarin slechts twee waarden worden gebruikt, TRUE en FALSE (WAAR en NIET WAAR). Een algebra waarin slechts twee waarden worden gebruikt leent zich uitstekend om te worden toegepast in digitale elektronica. Omdat digitale elektronica binair werkt kennen we daar ook slechts twee waarden, 1 en 0. In plaats van TRUE en FALSE gebruiken we meestal 1 en 0. Computers, microcontrollers en andere digitale elektronica rekenen diep van binnen met booleaanse algebra. De booleaanse algebra kent 3 basisbewerkingen: AND, OR en NOT.

Met twee schakelaars S1 en S2 in serie en een led kunnen we een AND schakeling bouwen. Als schakelaars S1 AND S2 beide gesloten zijn, dan brandt de led.



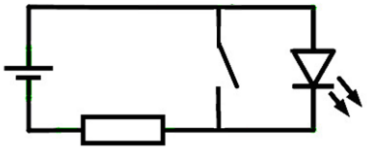
S1	S2	LED
0	0	0
0	1	0
1	0	0
1	1	1

Een OR schakeling kan je maken met twee schakelaars parallel.
Als schakelaar S1 OR S2 gesloten is, dan brandt de led.



S1	S2	LED
0	0	0
0	1	1
1	0	1
1	1	1

Een NOT schakeling heeft één ingang, en keert deze om. Als de schakelaar gesloten is is de LED uit en andersom. Een NOT schakeling wordt ook wel inverter genoemd:



S	LED
0	1
1	0

Met transistors kunnen we ook AND, OR en NOT schakelingen bouwen. Deze schakelingen worden poortschakelingen of poorten genoemd. Met combinaties van AND-, OR- en NOT-poorten kunnen we de afgeleide NAND-, NOR- en XOR-poorten maken.
Een NAND (Not AND) poort is een AND poort gevolgd door een NOT poort.
Een NOR (Not OR) poort is een OR poort gevolgd door een NOT poort.
Een XOR (eXclusive OR) krijg je met een NAND en een OR, gevolgd door een AND.
In een waarheidstabel ziet het er zo uit:

INGANGEN		UITGANG				
a	b	AND	NAND	OR	NOR	XOR
0	0	0	1	0	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	0	0

Door meerdere poortschakelingen te combineren kunnen we schakelingen met meer ingangen maken, schakelingen die bits kunnen optellen, bits kunnen schuiven en schakelingen die bits kunnen onthouden (geheugenschakelingen).
Een microcontrollerchip bestaat uit vele duizenden transistors die samen vele poortschakelingen vormen waarmee de microcontroller werkt.

3.8 Logische bewerkingen

Ook in een sketch kunnen we gebruik maken van booleaanse algebra.
Het datatype `bool` (of `boolean`) kan TRUE of FALSE zijn. HIGH en LOW of 1 en 0 mag ook.
In plaats van 1 mogen we zelfs elk getal gebruiken, positief of negatief, als het maar niet 0 is.
De verschillende bewerkingen hebben de symbolen: `&` (AND), `|` (OR), `!` (NOT) en `^` (XOR).

a	b	AND a&b	NAND !(a&b)	OR a|b	NOR !(a|b)	XOR a^b	NOT !a	NOT !b
0	0	0	1	0	1	0	1	1
0	1	0	1	1	0	1	1	0
1	0	0	1	1	0	1	0	1
1	1	1	0	1	0	0	0	0

3.9 Bitmanipulaties

De logische bewerkingen &, | en ^ kunnen ook worden toegepast op de getaltypen **byte**, **int** en **long**. Alle afzonderlijke bits in de getallen van de vergelijking worden dan logisch bewerkt.

```
Voorbeeld: 77 & 150 == 4        77 == 0 1 0 0 1 1 0 1
                               150 == 1 0 0 1 0 1 1 0
                         77 & 150 == 0 0 0 0 0 1 0 0 == 4


Voorbeeld: 77 | 150 == 223      77 == 0 1 0 0 1 1 0 1
                               150 == 1 0 0 1 0 1 1 0
                         77 | 150 == 1 1 0 1 1 1 1 1 == 223


Voorbeeld: 77 ^ 150 == 219      77 == 0 1 0 0 1 1 0 1
                               150 == 1 0 0 1 0 1 1 0
                         77 ^ 150 == 1 1 0 1 1 0 1 1 == 219
```

Maar wat heb je daar dan aan?
Op deze manier kunnen we de afzonderlijke bits in een byte maskeren, inverteren en schuiven.
Met de & (AND) functie kunnen we bits maskeren, bijvoorbeeld als je alléén de tweede byte uit een integer wilt hebben, dan kan je met & 0xFF (= binair 11111111) de eerste byte maskeren:

```
                            int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
                             0xFF == 0 0 0 0 0 0 0 0  1 1 1 1 1 1 1 1
                          a & 0xFF == 0 0 0 0 0 0 0 0  1 1 0 1 0 1 1 0
```

Met de ^ (XOR) functie kunnen we bits inverteren:

```
                    int a == 0xFF00 == 1 1 1 1 1 1 1 1  0 0 0 0 0 0 0 0
                             0x0808 == 0 0 0 0 1 0 0 0  0 0 0 0 1 0 0 0
                          a ^ 0x0808 == 1 1 1 1 0 1 1 1  0 0 0 0 1 0 0 0
```

Met << kunnen we bits naar links schuiven, met >> kunnen we bits naar rechts schuiven.

```
                            int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
                           a >> 8 == 0 0 0 0 0 0 0 0  0 0 0 1 0 1 1 0
                           a << 1 == 0 0 1 0 1 1 0 1  1 0 1 0 1 1 0 0
```

a >> 8 betekent schuif de bits 8 plaatsen naar rechts. Dat is hetzelfde als delen door 2^8.
a << 3 betekent schuif de bits 3 plaatsen naar links. Dat is hetzelfde als vermenigvuldigen met 2^3. De lege plaatsen die ontstaan worden opgevuld met nullen.
Je kan het vergelijken met decimaal schuiven. Vermenigvuldigen met 10^3 betekent ook dat alle cijfers 3 plaatsen naar links opschuiven.

Met de opdracht **bitWrite(a, n, b)** kunnen we in het getal **a** het **n**-de bit (van rechts, beginnend bij 0) de waarde **b** (0 of 1) geven:

```
                            int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
                          n=5, b=1                        1
                 bitWrite(a, 5, 1);
                            int a == 0 0 0 1 0 1 1 0  1 1 1 1 0 1 1 0
```

Met de opdracht **bitRead(a, n)** kunnen we het n-de bit van a uitlezen.

```
                            int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
                            n = 3                             0
                    bitRead(a, 3);  == 0
```

Met de opdracht `highByte(a)` kunnen we de hoogste byte uit een unsigned integer halen:

```
                    int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
highByte(a); == 0 0 0 1 0 1 1 0
```

Met de opdracht `lowByte(a)` kunnen we de laagste byte uit een unsigned integer halen:

```
                    int a == 0 0 0 1 0 1 1 0  1 1 0 1 0 1 1 0
lowByte(a); ==                                1 1 0 1 0 1 1 0
```

Vragen en opdrachten

3.8 Bereken 170 & 85
3.9 Bereken 132 & 132
3.10 Bereken 170 | 85
3.11 Bereken 132 | 132
3.12 Bereken 170 ^ 85
3.13 Bereken 132 ^ 132
3.14 Je gaat een 8 bit binaire teller maken. De hardware bestaat uit een Arduino UNO waarvan de digitale uitgangen 2 t/m 9 zijn aangesloten op 8 LEDs (met voorschakelweerstanden).
Teken het schema van de schakeling en bouw de schakeling op een breadboard.
Maak een sketch waarmee de rij LEDs binair gaat tellen van 00000000 tot 11111111.
Gebruik hierin een `byte` variabele en `bitRead`.
Maak de sketch zo kort mogelijk met `for(){}` loops.

3.10 Logische vergelijkingen met andere datatypen

Met de logische bewerkingen & (AND) en | (OR) kunnen de individuele bits in een byte of een int logisch worden bewerkt.
Andere datatypes kunnen ook in hun geheel logisch worden bewerkt. Hiervoor gebruiken we de dubbele symbolen `&&` (AND) en `||` (OR).
De volgende programmaregels komen uit een sketch van een RTC (Real Time Clock) en zorgen er voor dat op zondag 31 maart 2024 om 2 uur de klok automatisch op 3 uur wordt gezet, en op zondag 27 oktober 2024 van 3 uur weer terug naar 2 uur.

```
208   // begin zomertijd, tijd gaat van 2.00 > 3.00
209   if (hour==2&&dayOfWeek==1&&dayOfMonth==31&&month==3&&year==24){
210    setDS3231time(00,00,3,1,31,3,24);}
211   // begin wintertijd tijd gaat van 3.00 > 2.00
212   if(hour==3&&dayOfWeek==1&&dayOfMonth==27&&month==10&&year==24){
213    setDS3231time(00,00,2,1,27,10,24);
214    delay (3610000);}
```

Vragen en opdrachten

3.15 Leg uit waarom in regel 214 van bovenstaande regels een delay zit.
3.16 Hoe groot moet deze delay minimaal zijn?
3.17 Kan `&&dayOfWeek==1` (1=zondag) worden weggelaten? Waarom wel/niet?

4 Geheugen

4.1 Ponskaarten, magneetbanden en USB-sticks

De eerste elektronische computers (rond 1945) werden geprogrammeerd met schakelaars. De programma's konden niet worden opgeslagen. Als je een ander programma wilde gebruiken moest je het programma eerst invoeren met de schakelaars.
Niet lang daarna werd de ponsband gebruikt om programma's op te slaan. Een ponsband is een rol stevig papier met gaatjes. Draaiorgels gebruiken nog steeds ponsbanden om de liedjes op te slaan.
Mijn allereerste computerprogramma heb ik in 1978 geschreven op ponskaarten. Ik studeerde elektrotechniek aan de hogere technische school (HTS) in Rotterdam. Met een typemachine-achtig apparaat moesten we tijdens het computerpracticum de gaatjes in een kaartje 'typen'. Als het programma klaar was werd de stapel ponskaarten in een kaartlezer gedaan, en het programma werd per telefoonlijn naar de computer in Delft gestuurd. Deze stuurde het resultaat via de telefoon terug naar Rotterdam, waar het op kettingpapier werd afgedrukt.
Programma's en data werden ook opgeslagen op magnetische informatiedragers zoals magneetbanden, en later cassettebandjes, floppy disks en diskettes. De diskette wordt in veel apps nog steeds gebruikt als icoontje voor opslaan.

De harde schijf is ook een magnetische informatiedrager en deze wordt tegenwoordig (2019) nog steeds heel veel gebruikt.
In 1981 deed de eerste optische datadrager, de CD zijn intrede. Latere optische systemen zijn de minidisk, de DVD en de BlueRay. Eigenlijk was de ponsband ook een optische drager want hij werd meestal uitgelezen met lichtstraaltjes door de gaatjes.
Tegenwoordig wordt steeds meer gebruik gemaakt van flash-geheugen. De informatie zit hierbij opgeslagen in een chip. Flash-geheugen zit o.a. in USB-sticks, SD-kaartjes, SSD-drives, smartphones, tablets en... de Arduino.

4.2 Bytes, kilobytes, megabytes en gigabytes

Kilo is 1000 (10^3) in de natuurkunde, mega is 1000 x 1000 (10^6) en giga is 1000 x 1000 x 1000 (10^9).
Bij bytes is dat anders. Omdat het getal 1000 (decimaal) in het binaire stelsel geen afgerond getal is wordt de dichtstbijzijnde in het binair stelsel afgeronde waarde van 2^{10} gebruikt als kilo. Dat is 1024 (decimaal).

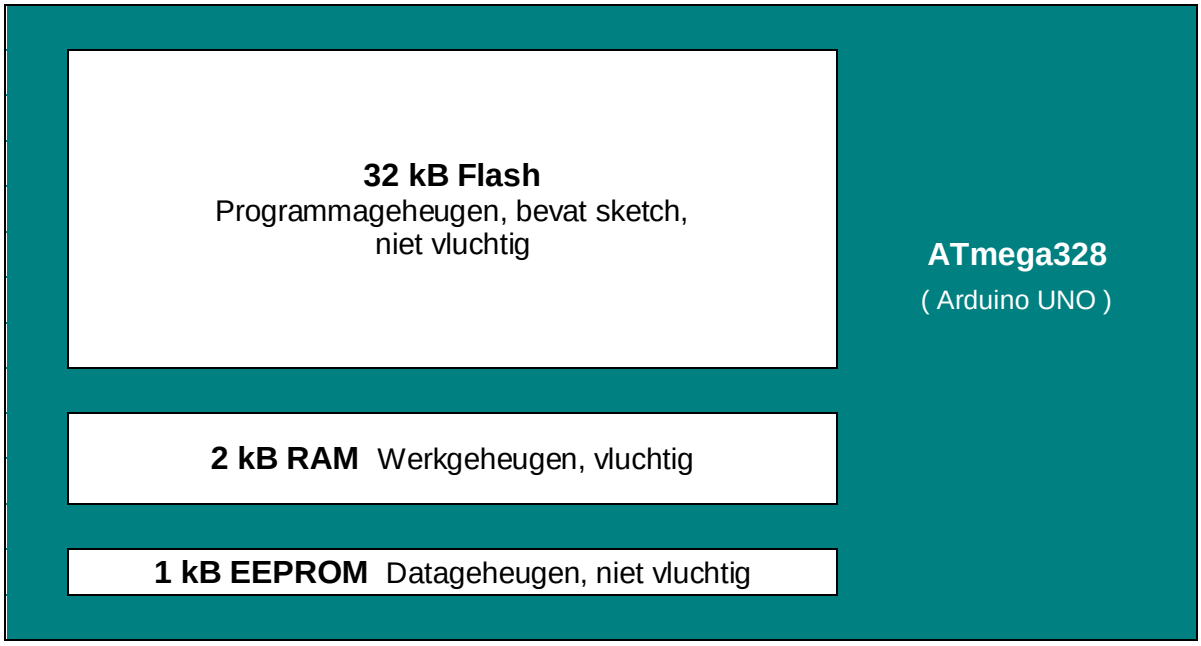
Een kilobyte (kB) is 1024 bytes (2^{10} bytes).
Een megabyte (MB) is 1024 kilobytes en dus 1024 x 1024 = 1048576 bytes (2^{20} bytes).
Een gigabyte (GB) is 1024 megabytes en dus 1024 x 1024 x 1024 = 1073741824 bytes (2^{30} bytes)

Vragen en opdrachten

4.1 Hoe noem je 2^{40} bytes?
4.2 Hoe noem je 2^{50} bytes?
4.3 Hoe noem je 2^{60} bytes?

4.3 De geheugenstructuur van de Arduino UNO

De ATmega328 microcontroller-chip in de Arduino UNO heeft drie soorten geheugen aan boord. Elk heeft z'n eigen functie.

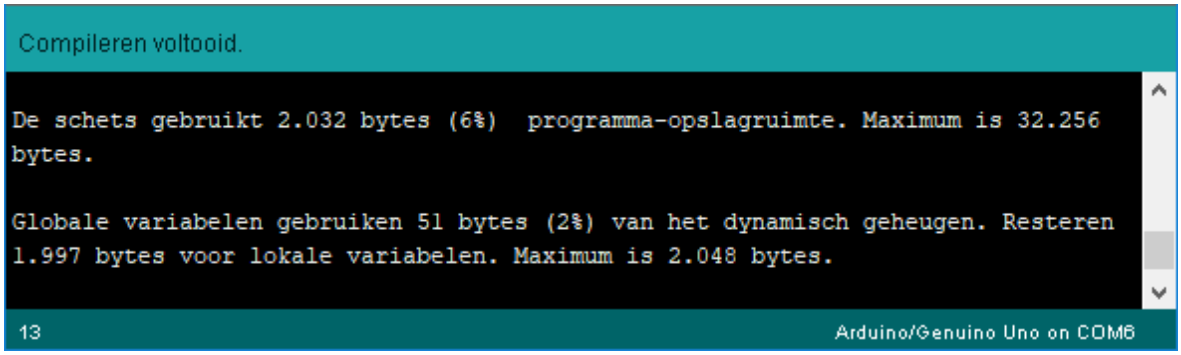


4.4 Flash programmageheugen

Het grootste geheugen van de Arduino UNO is het flash-geheugen van 32 kilobytes. Hierin staat de sketch, samen met de bootloader. De bootloader is een stukje programma wat er door de fabrikant is ingezet. De bootloader zorgt ervoor dat de chip met de Arduino IDE kan samenwerken. Het flash geheugen is niet vluchtig (engels: non volatile). Dat betekent dat de gegevens er in blijven staan als de voedingsspanning wegvalt. De sketch blijft dus gewoon in het geheugen als je de voedingsspanning van de Arduino wegneemt. Pas als je een andere sketch upload wordt het deel van het flash geheugen waar de sketch staat overschreven.

4.5 RAM werkgeheugen

Het RAM (Random Acces Memory) wordt door de microcontroller gebruikt om constanten en variabelen in op te slaan. Ook tussentijdse stappen van berekeningen worden er tijdelijk in opgeslagen. Daarnaast houdt de micrcontroller met een index in het RAM bij waar hij de verschillende gegevens opslaat, zodat hij ze ook kan terugvinden.
Het RAM geheugen is vluchtig (engels: volatile). Dat betekent dat alle in het RAM opgeslagen gegevens verloren gaan als de voedingsspanning wegvalt. Na het compileren van een sketch laat de Arduino IDE onderin het venster zien hoeveel geheugenruimte van het flash (programma-opslagruimte) en van het RAM (dynamisch geheugen) is gebruikt.



4.6 EEPROM datageheugen

Het EEPROM (Electrically Erasable Programmable Read Only Memory) is voor ons de meest interessante omdat we er zelf gegevens in kunnen opslaan. Het EEPROM is niet vluchtig, dus de gegevens die we er in opslaan blijven bewaard, ook als de voedingsspanning van de Arduino wegvalt.

Een apparaat waarin allerlei instellingen gemaakt kunnen worden, zoals een televisietoestel, slaat deze instellingen op in een EEPROM (of een flash) zodat ze behouden blijven, ook als je de stekker van het apparaat er uit hebt getrokken. Als je de TV aan zet, krijg je de laatstgekozen zender te zien en het volume van het geluid staat ook zoals je dat het laatst had ingesteld.
Een dimbaar LED-lampje kan worden aangezet in de laatstgekozen dim-stand, een spelletje kan de highscore opslaan in EEPROM enzovoort.

Om het EEPROM van de Arduino te gebruiken kan je de standaard EEPROM library gebruiken (zie hoofdstuk 5.1, blz 51).
Een getal naar EEPROM schrijven doe je met de opdracht: **`EEPROM.write(adres, getal)`**.
Het EEPROM is 1kB (1024 bytes) groot. Het adres is dus een getal tussen 0 en 1023.
Het getal dat kan worden weggeschreven is 8 bits lang, en dus een byte (0 - 255).
Een getal uit het EEPROM lezen doe je met **`EEPROM.read(adres)`**.

De volgende sketch houdt bij hoe veel keer de Arduino is opgestart, stuurt dit aantal naar de seriële monitor en schrijft het over de vorige waarde in EEPROM adres 0.

```
1  // Sketch 4.1: EEPROM lezen en schrijven ////////////////////////////
2
3  #include <EEPROM.h>              // Library toevoegen aan de sketch.
4
5  void setup(){
6    Serial.begin(9600);
7    byte aantal = EEPROM.read(0);        // aantal ophalen uit EEPROM.
8    aantal++;                            // 1 erbij optellen.
9    Serial.print("Deze Arduino is ");
10   Serial.print(aantal);
11   Serial.println(" keer opgestart");
12   EEPROM.write(0, aantal);             // aantal opslaan in EEPROM.
13 }
14
15 void loop(){}
```

Het EEPROM wordt gegarandeerd om tenminste 100000 keer te kunnen worden overschreven. Dus als je een EEPROM.write opdracht op de verkeerde plek in een lus van je sketch zet kan dat binnen 5 minuten leiden tot permanent geheugenverlies! Om het aantal keren schrijven te beperken kan je de opdracht **`EEPROM.update(adres, getal)`** gebruiken. Met deze opdracht wordt het adres alleen overschreven als het getal veranderd is. Nog steeds moet je oppassen met een waarde die vaak verandert, zoals bij een analoge sensor met ruis die in een lus wordt uitgelezen.

Vragen en opdrachten

4.4 Open de seriële monitor en laad sketch 4.1. Begint de teller bij 1?
Bedenk een manier om de teller te resetten, en reset de teller.

4.5 Maak een prototype en een sketch met een potmeter die een getal tussen 0 en 255 maakt en dit getal maximaal 1x per seconde opslaat in EEPROM. Na opstarten moet de Arduino de laatst opgeslagen potmeterwaarde laten zien.

4.6 Maak een prototype en een sketch met twee druktoetsen. Met toets 1 wordt het hele EEPROM gewist (alle adressen op 0), met toets twee wordt het hele EEPROM gevuld met ruis (alle adressen een willekeurige waarde tussen 0 en 255). Per toetsaanslag mag maar één keer naar elk adres van het EEPROM worden geschreven. Bij opstarten wordt de inhoud van het hele EEPROM getoond op de seriële monitor.

4.7 Een integer opslaan en uitlezen

Omdat een geheugenplaats niet meer dan één byte (8 bits) kan bevatten zal het aantal keren dat de Arduino is opgestart met sketch 4.1 na 255 keer weer bij nul beginnen. Als we verder willen tellen moeten we een integer (16 bits) gebruiken. Daarvoor moeten we de integer eerst omrekenen naar twee bytes, en deze twee bytes vervolgens in twee geheugenplaatsen opslaan.
Het omrekenen van een integer naar twee bytes kan met de bitmanipulaties & en >> (maskeren en bits schuiven (zie ook hoofdstuk 3.9 blz. 42).
In het volgende voorbeeld rekenen we de integer 5846 (=binair 1011011010110) om naar twee afzonderlijke bytes:

```
 byte1    byte2

00010110 11010110     5846     // de integer die we gaan omrekenen
                      &        // alle bits apart & met
00000000 11111111     0xFF     // 0xFF (= binair 11111111)
                      =
00000000 11010110     byte2    // resultaat = byte2 = 214

                               // dan byte1 er uit halen

00010110 11010110     5846     // opnieuw de integer
                      >> 8     // alle bits 8 plaatsen naar links
00000000 00010110     byte1    // resultaat = byte1 = 22
```

Zo ziet het er in de sketch uit:

```
int aantal = 5846;
byte byte2 = aantal & 0xFF;
byte byte1 = aantal >> 8;
EEPROM.write(adres, byte1);
EEPROM.write(adres + 1, byte2);
```

Bij het lezen doen we precies het tegenovergestelde, we lezen de twee geheugenplaatsen en met de twee bytes reconstrueren we de integer:

```
byte byte1 = EEPROM.read(adres);
byte byte2 = EEPROM.read(adres + 1);
aantal = (byte1 << 8) + byte2;
```

Gebruik bij het lezen en schrijven van integers de even adressen. Er kunnen dan 512 integers worden opgeslagen.

4.8 Een long opslaan en uitlezen

Een long opslaan:

```
long aantal = x;
byte byte4 = aantal & 0xFF;
byte byte3 = (aantal >> 8) & 0xFF;
byte byte2 = (aantal >> 16) & 0xFF;
byte byte1 = aantal >> 24;
EEPROM.write(adres, byte1);
EEPROM.write(adres + 1, byte2);
EEPROM.write(adres + 2, byte3);
EEPROM.write(adres + 3, byte4);
```

En reconstrueren:

```
byte byte1 = EEPROM.read(adres);
byte byte2 = EEPROM.read(adres + 1);
byte byte3 = EEPROM.read(adres + 2);
byte byte4 = EEPROM.read(adres + 3);
aantal = (byte1 << 24) + (byte2 << 16) + (byte3 << 8) + byte4;
```

Vragen en opdrachten

4.7 Schrijf een sketch waarmee je een integer naar het EEPROM kan schrijven en uitlezen. Gebruik hierbij geen maskering of schuiven zoals op de vorige bladzijde, maar `highByte` en `lowByte` (zie hoofdstuk 3.9, blz. 42 en 43).

4.9 Gebruik van het flash geheugen

Er kunnen maximaal 256 variabelen van het type long worden opgeslagen in het EEPROM. Dit is niet zo veel. Het zou handig zijn als we ook het flash geheugen konden gebruiken. Er zijn libraries die dit mogelijk maken, maar je moet dan wel weten wat je doet. Als je per ongeluk over een stuk van je sketch schrijft is je sketch kapot en kunnen er rare dingen gebeuren. Nog erger is het als je de bootloader kapot maakt. De microcontroller kan hierdoor volledig onbruikbaar worden.

4.10 Een circulaire buffer

Stel je hebt een sensor waarvan de waarde nogal varieert, zoals een wapperende windrichtingmeter of een windsnelheidsmeter. Om hiervan een betrouwbare meting te maken moet je meerdere metingen doen, bijvoorbeeld iedere seconde één, en dan het gemiddelde van een aantal metingen, bijvoorbeeld 60 metingen (een minuut) uitrekenen en op het display tonen. De gemiddelde waarde wordt dan één keer per minuut uitgerekend en op het display getoond.
Met een circulaire buffer worden de laatste 60 metingen opgeslagen en gebruikt om het gemiddelde uit te rekenen. Elke seconde wordt er één meting in de buffer ververst. Op die manier kan de waarde op het display elke seconde het gemiddelde van de afgelopen minuut laten zien.
In het volgende voorbeeld kan de interval (tijd tussen twee metingen) worden ingesteld in regel 3. In regel 4 wordt de grootte van de buffer ingesteld. Het gemiddelde van een sensorwaarde op analoge pin 0 (regel 19) wordt berekend en naar de seriële monitor gestuurd.

```
1  // Sketch 4.2: Circulaire buffer ////////////////////////////////////////
2
3  int interval=1000;       // Interval (in ms) tussen de metingen
4  const int bufGr=60;      // Buffergrootte, moet const zijn i.v.m. regel 5.
5  int circBuf[bufGr];      // BufGr moet const zijn, geen variabele.
6  unsigned long reset=0;   // Reset voor de intervaltimer.
7  int n=0;                 // Plaats in de buffer.
8  float som;               // Som van alle bufferwaarden.
9  float middel;            // Gemiddelde van alle bufferwaarden.
10
11 void setup(){
12   Serial.begin(9600);    // Seriële monitor starten.
13 }
14
15 void loop(){
16   if((millis()-reset)>=interval){  // Tijd om te meten?
17     reset=millis();                // Intervaltimer resetten.
18     if(n>=bufGr)n=0;               // plaats n in buffer is maximaal bufGr.
19     circBuf[n]=analogRead(0);      // Sensorwaarde in array schrijven.
20     Serial.print(n);               // Print bufferadres.
21     Serial.print("  ");
22     Serial.print(circBuf[n]);      // Print waarde in bufferadres.
23     Serial.print("  ");
24     for(int i=0; i<bufGr; i++){    // Som van alle bufferwaarden berekenen.
25       som=som+circBuf[i];
26     }
27     middel=som/bufGr;              // Gemiddelde berekenen.
28     Serial.println(middel, 2);     // Print gemiddelde met twee decimalen.
29     som=0;                         // Som resetten voor volgende meting.
30     n++;                           // Volgende keer volgende bufferplaats.
31   }
32 }
```

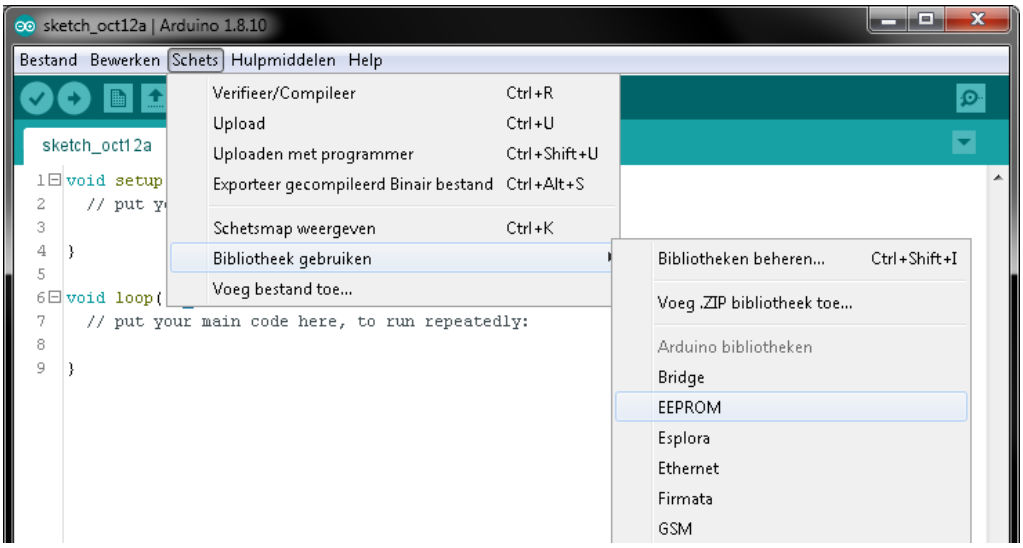
5 Libraries

Een library is een stukje software wat je aan je sketch kunt toevoegen. Met een library krijg je extra functies die het programmeren van sommige sensoren, displays of andere onderdelen makkelijker maakt.
Tijdens de installatie van de Arduino IDE is ook een folder met standaardlibraries geïnstalleerd. Er zijn ook honderden andere libraries te vinden op het internet.
Huidige versies van de Arduino IDE zijn ook in het nederlands dus laten we de library verder maar bibliotheek noemen.

5.1 Een standaard bibliotheek toevoegen

In dit voorbeeld gaan we de EEPROM bibliotheek aan een sketch toevoegen.

Ga naar **Schets/Bibliotheek gebruiken** en klik op **EEPROM**.



Aan het begin van de sketch staat nu `#include <EEPROM.h>`.

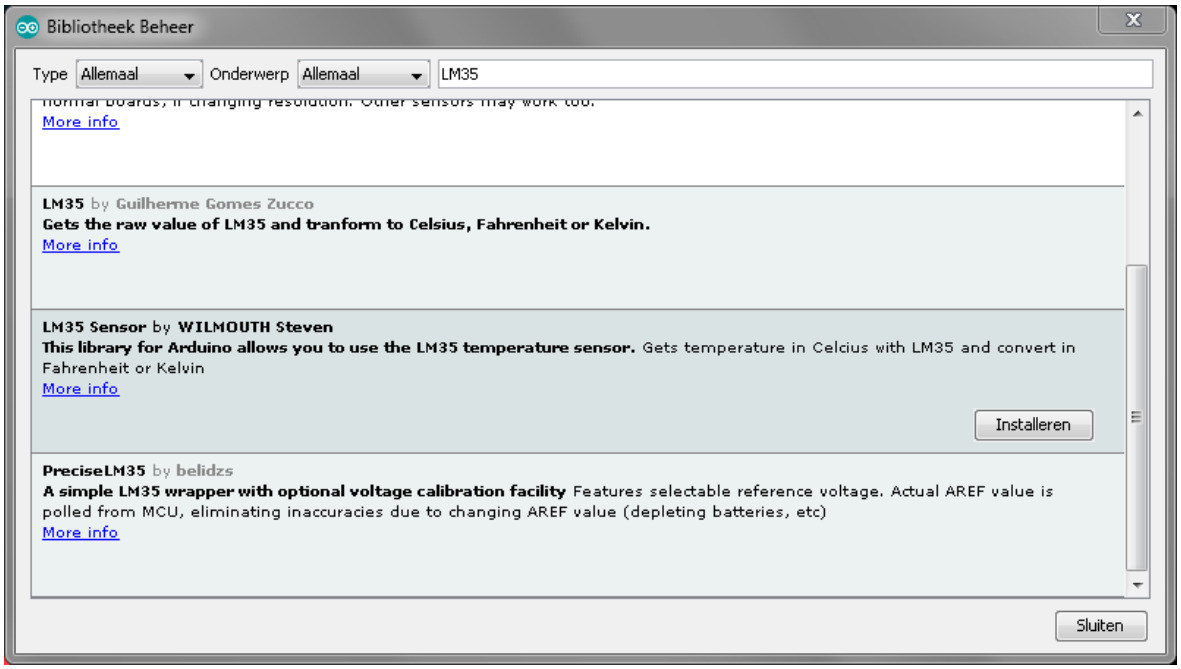
```
#include <EEPROM.h>

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Dit betekent dat de bibliotheek aan je sketch is toegevoegd en dat je de functies van deze bibliotheek kunt gebruiken.
De software zelf is in de sketch niet te zien, je ziet alleen `#include <EEPROM.h>`. Tijdens het compileren van de sketch wordt de bibliotheek uit de folder in de sketch gezet.

5.2 Een standaardbibliotheek installeren

De bibliotheken die je te zien krijgt in de lijst **Schets/Bibliotheek gebruiken** is maar een klein gedeelte van de beschikbare bibliotheken. Het zijn alleen de geïnstalleerde bibliotheken. Zoek je een bibliotheek die niet in de lijst staat, dan moet je hem eerst installeren.
Ga naar **Schets/Bibliotheek gebruiken/Bibliotheek beheren...**

Er opent een venster met een enorme lijst bibliotheken die je kan installeren. In het vak ***Filter je zoekresultaten...*** Kan je een zoekopdracht geven, bijvoorbeeld de naam van de sensor waar je een bibliotheek voor zoekt. Laten we een bibliotheek zoeken voor de LM35 temperatuursensor. Typ LM35 in het zoekvak en druk op enter. Je krijgt nu alléén de bibliotheken met LM35 in de naam te zien. Kies de bibliotheek die je wilt installeren en druk op Installeren.



Achter de naam van de library staat nu INSTALLED. De bibliotheek is geïnstalleerd en staat nu in de lijst **Schets/Bibliotheek gebruiken**, onderaan onder **Bijgedragen bibliotheken** en kan worden gebruikt in je sketch.

Om te zien welke functies de bibliotheek toevoegt kan je naar **Bestand/Voorbeelden**. Hier vind je voorbeeldsketches met de geïnstalleerde bibliotheken.

5.3 Een bibliotheek downloaden van het internet

Staat de bibliotheek die je zoekt niet in de Bibliotheekbeheerlijst, dan kan je op het internet zoeken. Bibliotheken worden meestal aangeboden als geZIPte map. Heb je de bibliotheek gevonden, dan download je de gehele gezipte map. Je hoeft de map niet te unzippen, dat doet bibliotheekbeheer. Ga naar **Schets/Bibliotheek gebruiken** en klik op **Voeg .ZIP bibliotheek toe…**
Ga naar de map waar je het ZIP-bestand hebt gedownload (meestal **Downloads)** en open de ZIP-folder van de bibliotheek. De folder wordt dan ge-unzipped, de bibliotheek wordt toegevoegd aan de lijst in bibliotheekbeheer en geïnstalleerd. De ZIP-folder bevat behave de bibliotheek meestal ook voorbeeldsketches die na installatie onder **Bestand/Voorbeelden** te vinden zijn.

5.4 Problemen met bibliotheken

Als je op verschillende computers werkt met je sketch, dan zul je merken dat de bibliotheken soms niet werken. De IDE kan de bibliotheek niet vinden.
De bibliotheek zit niet in de opgeslagen sketch, maar wordt tijdens het compileren uit de libraries-folder gehaald en aan de sketch toegevoegd. Gebruik je een andere computer, dan moeten de gebruikte bibliotheken ook op die computer worden geïnstalleerd.

Er zijn verschillende bibliotheken met dezelfde naam in omloop. Dit kan heel verwarrend zijn. Zorg er voor dat je de juiste bibliotheek hebt. Heb je een bibliotheek geprobeerd die niet voldoet en wil je een andere bibliotheek proberen, ruim de niet gebruikte bibliotheek dan op om verwarring te voorkomen.
Zet in je sketch een commentaarregel met het internetadres waar de bibliotheek vandaan komt.

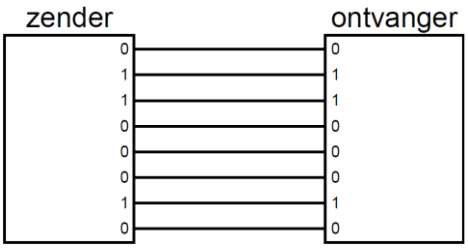
Het de-installeren van een bibliotheek kan (nog) niet in bibliotheekbeheer (IDE v1.8 en eerder). Hiervoor moet je met de windows verkenner naar de libraries-map om de hele foder van die bibliotheek te verwijderen.

Bij gebruik van meerder bibliotheken in een sketch kan het gebeuren dat twee bibliotheken elkaar in de weg zitten omdat ze dezelfde registers of timers gebruiken. Je krijgt dan een foutmelding onderaan je scherm. Dit probleem is het beste op te lossen door één van de conflicterende bibliotheken te vervangen door een andere bibliotheek.

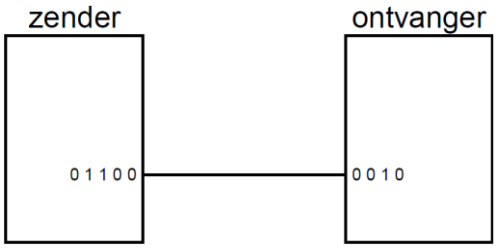
6 Communicatie

6.1 Parallel en serieel

Bij de overdracht van data tussen twee apparaten wordt onderscheid gemaakt tussen parallelle en seriële overdracht. Bij parallelle overdracht wordt de data met hele bytes (8 bits) tegelijk verstuurd over acht afzonderlijke draden. Bij seriële overdracht worden de bits één voor één over één enkele draad verstuurd.



Parallelle dataoverdracht



Seriële dataoverdracht

Het nadeel van parallelle overdracht is dat er dikke, dure kabels nodig zijn. Die kabels mogen bovendien niet te lang zijn omdat de lange draden elkaar dan gemakkelijk kunnen storen. Seriële overdracht kan met een goedkope dunne kabel over grotere afstanden. Serieel is in principe langzamer dan parallel, maar digitale systemen zijn tegenwoordig zo snel dat dat nauwelijks meer een nadeel is.
Er zijn de afgelopen 50 jaar veel seriële protocollen ontwikkeld voor communicatie met harde schijven, muziekapparatuur, sensoren enzovoort. Een heel bekend seriëel protocol is de USB-bus.

6.2 I²C

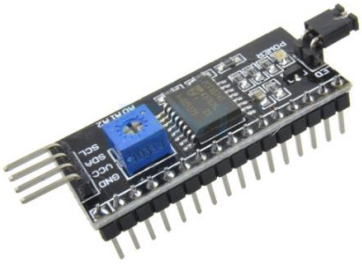
Een seriëel protocol dat veel door microcontrollers wordt gebruikt is I²C (I kwadraat C). Dat staat voor Inter IC-bus. De I²C-bus is ontwikkeld om verschillende IC's binnen één systeem met elkaar te verbinden. De Arduino maakt veel gebruik van I²C om bijvoorbeeld sensoren en displays aan te sluiten. Elk type Arduino heeft tenminste één I²C interface. Je herkent I²C modules aan de SDA en SCL aansluitingen.

SDA is de Serial DAta aansluiting, de draad waardoor de data verstuurd word. Naast de data is er ook een kloksignaal nodig. Dit kloksignaal staat op de SCL (Serial CLock) pin. Naast het SDA en SCL signaal zijn ook de voedingsspanning van 5V (VCC) en 0V (GND) nodig. De I²C verbinding gebruikt dus 4 draden.
Het I²C protocol is een bus protocol. Dat wil zeggen dat er meerdere I²C modules parallel op één enkele bus kunnen worden aangeloten. Om met de juiste module te communiceren heeft iedere module een uniek adres van 0-127. Op de I²C bus is slechts één master aangesloten. De master genereert het SCL-signaal. Meestal is de Arduino de master. Het I²C protocol is bi-directioneel (kan twee kanten op, van master naar slave of van slave naar master).
Voor de meeste I²C modules zijn libraries die het gebruik van de modules eenvoudiger maken.

Seriëel naar parallel

Een LCD display heeft meestal 16 aansluitpinnen, waarvan er 6 worden aangesloten op 6 digitale uitgangen van de Arduino. Dit is een parallelle verbinding. Met een seriëel naar parallel interface tussen de Arduino en het display kunnen we het display aansluiten op twee uitgangen van de Arduino.

Deze interface is kant en klaar te koop in de vorm van een I²C-backpack. Een backpack is een soort shield wat je direct op de display print kan solderen met 16 pinnen.



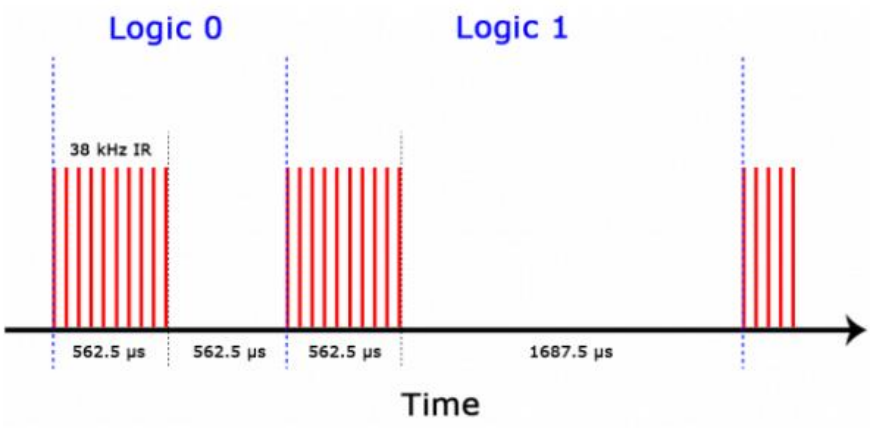
Het I²C Display backpack

Om de backpack te kunnen gebruiken heb je een bibliotheek nodig. De fabrikant of leverancier van het backpack geeft meestal een link naar een goed werkende bibliotheek. In de folder van die bibliotheek vind je meestal ook een gebruiksaanwijzing en voorbeeldsketches. Heb je een voorbeeldsketch ge-upload en zie je geen tekst verschijnen? Dan kan het zijn dat het contrast niet goed staat ingesteld. Draai aan de potmeter (het schroefje in het blauwe blokje op het backpack) tot het contrast goed is ingesteld.
Het kan ook zijn dat het I2C adres niet goed is ingesteld. De meeste LCD-I²C backpacks staan standaard ingesteld op het adres 0x27 of 0x3F.
Wil je meerdere LCD displays aansluiten op één Arduino master, dan moeten de backpacks verschillende I²C adressen hebben. Je kan het adres van het backpack veranderen door één of meer van de soldeereilandjes A0, A1 of A2 op de backpackprint met een druppeltje soldeer door te verbinden.

6.3 IR afstandbediening

Met een afstandbediening kan je vanuit je luie stoel de televisie bedienen. Een afstandbediening kan ook handig zijn om apparaten die niet gemakkelijk bereikbaar zijn te bedienen, zoals een beamer die hoog aan het plafond hangt of een rijdende robot. Een veelgebruikte manier om apparaten op afstand te bedienen is met infrarood (IR).
Een infrarood afstandsbediening herken je aan de IR led aan de voorkant. IR kan je niet zien, maar een digitale camera kan dat wel. Kijk met de camera van je smartphone (of een andere digitale camera) naar de led van een IR afstandsbediening en druk op een toets van de afstandbediening. Zo kan je controleren of de IR led werkt.

Zodra je op een toets van de afstandbediening drukt wordt de code van die toets als binair getal seriëel verzonden in de vorm van IR pulsen. Er bestaan verschillende IR protocollen, maar de meest gebruikte werkt als volgt:
Een logische 0 bestaat uit 562,5 µs (microseconden) hoog, gevolgd door 562,5 microseconden laag.
Een logische 1 bestaat uit 562,5 µs hoog, gevolgd door 1687,5 µs laag. Als het signaal hoog is brandt de IR-led niet voortdurend, maar hij knippert met een frequentie van 38 kHz. Hierdoor kan de IR-ontvanger het signaal herkennen en eventuele IR-stoorsignalen er uit filteren.



Het zelf programmeren van de codes is niet nodig. Hier zijn verschillende IR-libraries voor. Wil je de Arduino met een bestaande IR afstanbediening besturen, dan heb je een IR sensor nodig, bijvoorbeeld de TSOP38238.

De TSOP38238 IR sensor

Deze sensor heeft het 38 kHz filter ingebouwd. Met deze sensor kan de Arduino de meeste IR afstandbedieningssignalen ontvangen. Het decoderen van het signaal doe je met een geschikte bibliotheek.

6.4 433 MHz

Als er geen directe zichtverbinding is tussen zender en ontvanger zal een IR verbinding niet werken. In dat geval kan je zenden en ontvangen met een RF-signaal (Radio-Frequent). De lucht is vol RF signalen van mobieltjes, wifi, draadloze voordeurbel, babyfoon, radio enzovoort. Omdat elk zend-ontvangst systeem een eigen frequentie heeft kunnen ze naast elkaar bestaan zonder elkaar te storen. 433 MHz is een frequentie waarop je zelf mag zenden, mits het zendvermogen niet groter is dan 10 milliwatt. Met dit vermogen is het maximale bereik binnenshuis ongeveer 20 meter, afhankelijk van muren en andere obstakels. Buiten is een afstand van maximaal 50 meter haalbaar. Om het maximale bereik te kunnen halen heb je wel een antenne nodig. De antenne moet een lengte hebben van ¼ golflengte. Bij 433 MHz is dat 17 cm. Deze antenne kan een simpele koperdraad van 17 cm zijn, of een speciale antenne die te koop is.

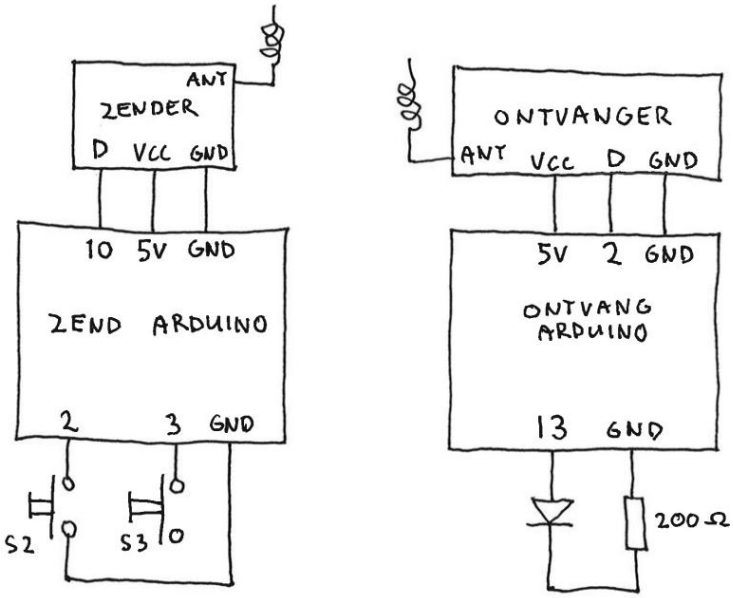
Een 433 MHz zender

Een 433 MHz ontvanger

Vragen en opdrachten

Je gaat een simpele RF afstandbediening maken waarmee je op afstand een LED aan en uit kunt zetten. Hiervoor heb je twee Arduino's nodig. De Arduino met de zender heeft drukknoppen om de LED aan of uit te zetten. De LED is aangesloten op de ontvangende Arduino.

6.1 Bouw een zender en een ontvanger met twee Arduino's en een 433 MHz zender-ontvanger set volgens onderstaand schema:



6.2 Download de gezipte RC-Switch bibliotheek van: *https://github.com/sui77/rc-switch/archive/master.zip* en installeer de bibliotheek met **Schets/Bibliotheek gebruiken/Voeg .ZIP bibliotheek toe…**(zie ook 5.3 op blz 52).

6.3 Upload de volgende sketch in de zend Arduino:

```
// Sketch 6.3: 433 MHz zender ////////////////////////////////////////////
#include <RCSwitch.h>                    // De RCSwitch library
const int drukknop2 = 2;                 // Drukknop 2 op pin 2
const int drukknop3 = 3;                 // Drukknop 3 op pin 3
RCSwitch mySwitch = RCSwitch();          // Definieer mySwitch

void setup(){
  pinMode(drukknop2, INPUT_PULLUP);      // Drukknop tussen pin en GND
  pinMode(drukknop3, INPUT_PULLUP);      // Drukknop tussen pin en GND
  mySwitch.enableTransmit(10);           // Zender op Arduino pin 10
}

void loop(){
  if (digitalRead(drukknop2) == LOW){    // Als knop 2 is ingedrukt
    mySwitch.send(2, 8);                 // Zend getal 2 in 8-bit protocol
    delay(100);                          // Tijd nodig om te zenden
  }
  if (digitalRead(drukknop3) == LOW){ // Als knop 3 is ingedrukt
    mySwitch.send(3, 8);                // Zend getal 3 in 8-bit protocol
    delay(100);                         // Tijd nodig om te zenden
  }
  delay(200);
}
```

6.4 Upload de volgende sketch in de ontvangende Arduino:

```
// Sketch 6.4: 433 MHz ontvanger ////////////////////////////////////
#include <RCSwitch.h>              // De RCSwitch library
int led = 13;                      // LED op pin 13
RCSwitch mySwitch = RCSwitch();  // Definieer mySwitch

void setup(){
 mySwitch.enableReceive(0);  // Ontvanger op ext. interrupt 0 (pin 2)
 pinMode(led,OUTPUT);
 digitalWrite(led,LOW);
}

void loop(){
  if (mySwitch.available()){             // Klaar om te ontvangen?
    if (mySwitch.getReceivedValue()){    // Als waarde niet nul is
      byte value = mySwitch.getReceivedValue();
      if (value == 2)digitalWrite(led,HIGH);
      if (value == 3)digitalWrite(led,LOW);
    }
    mySwitch.resetAvailable();           // Klaarzetten om de volgende
  }                                      // byte te ontvangen
  delay(100);
}
```

6.5 Verander de zendersketch zodat met drukknop2 de LED op de ontvanger kan worden aangezet én uitgezet (toggle).

6.6 Voeg een LED toe aan de ontvangende Arduino.

6.7 Verander de zendsketch én de ontvangsketch zodat drukknop2 LED 13, en drukknop3 de extra LED aanzet en uitzet (toggle).

7 Tabellen

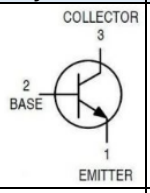
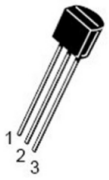
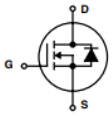
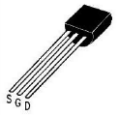
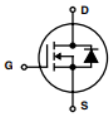
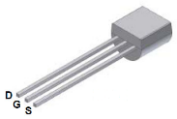
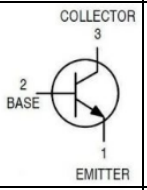
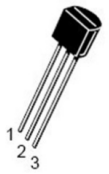
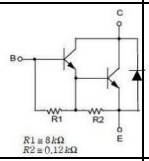
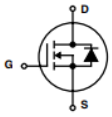
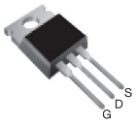
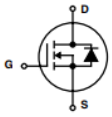
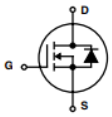
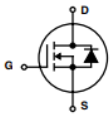
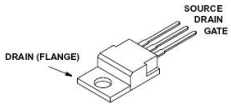
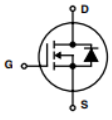
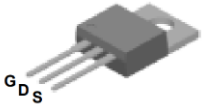
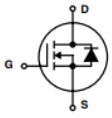
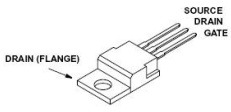
7.1 ASCII tabel

32	SP	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

7.2 Arduino boards vergeleken

Arduino board	UNO	MEGA 2560	DUE	Leonardo	Nano	Micro	Pro Micro
Microcontroller	ATmega 328	ATmega 2560	ATSAM3 X8E	ATmega 32U4	ATmega 168	ATmega 32U4	ATmega 32U4
Printmontage	DIL voet	SMD	SMD	SMD	SMD	SMD	SMD
Klokfrequentie (MHz)	16	16	84	16	16	16	16
Ingangsspanning (V)	7-12	7-12	7-12	7-12	7-9	7-12	7-12
Werkspanning (V)	5	5	3,3	5	5	5	5 of 3,3
Flash (kB)	32	256	512	32	16	32	32
RAM (kB)	2	8	96	2,5	1	2,5	2,5
EEPROM (kB)	1	4	-	1	0,512	1	1
Digitaal I/O	14	54	54	20	24	20	12
PWM UIT	6	15	12	7	6	7	5
Analoog IN	6	16	12	12	8	12	4
Analoog UIT	-	-	2	-	-	-	-
Interrupts	2	5	54	5	2	4	4
USB	Regular	Regular	2 Micro	Micro	Mini	Micro	Micro

7.3 Veelgebruikte transistoren

Transistor	Type	Symbool	Pinout	V_{max}	I_{max}	Prijs
BC547	NPN			50 V	100 mA	€ 0,08
2N7000	MOSFET			60 V	200 mA	€ 0,20
BS170	MOSFET			60 V	500 mA	€ 0,50
2N2222	NPN			40 V	600 mA	€ 0,50
TIP120	NPN Darlington			60 V	5 A	€ 0,50
IRL510PBF				100 V	5 A	€ 0,60
IRL520	MOSFET			100 V	10 A	€ 0,60
IRL540	MOSFET			100 V	20 A	€ 0,70
BUZ11	MOSFET			50 V	30 A	€ 0,80
IRF520	MOSFET					
BD139	NPN					
FQP27P06	MOSFET					
FQP30N06L	MOSFET			60 V	30 A	€ 1,00
IRF640N	MOSFET			200 V	18 A	€ 1,00

7.4 Rekenkundige bewerkingen met Arduino

```
y=4;              // waarde van y wordt 4
y=a;              // y wordt gelijk aan a
y=y+1;            // 1 optellen bij y
y++;              // hetzelfde als y=y+1
y=y+a;            // a optellen bij y
y+=a;             // hetzelfde als y=y+a
y=y-1;            // 1 aftrekken van y
y--;              // hetzelfde als y=y-1
y=y-a;            // a aftrekken van y
y-=a;             // hetzelfde als y=y-a
y=y*a;            // y vermenigvuldigen met a
y*=a;             // hetzelfde als y=y*a
y=y/a;            // y delen door a
y/=a;             // hetzelfde als y=y/a
y=min(a,b);       // y wordt kleinste waarde van a en b
y=max(a,b);       // y wordt grootste waarde van a en b
y=abs(a);         // y wordt absolute waarde van a (haalt minteken weg)
y=constrain(a,b)  // y wordt minimaal a en maximaal b
y=pow(a,b);       // y wordt a tot de macht b
y=sq(a);          // y wordt het kwadraat van a (zelfde als y=a*a)
y=sqrt(a);        // y wordt de vierkantswortel uit a
y=log10(a);       // y wordt de logaritme (grondtal 10) van a
y=log(a);         // y wordt de natuurlijke logaritme van a
y=exp(a);         // y wordt exponentiele waarde van a
y=sin(a);         // y wordt sinus van a (in radialen)
y=cos(a);         // y wordt cosinus van a (in radialen)
y=tan(a);         // y wordt tangens van a (in radialen)
y=atan(a);        // y wordt arctangens (in radialen) van a
y=atan2(a,b);     // y wordt arctangens (in radialen) van a/b
```

Index