

Fe Cube: Maak het! Versta het!

Ingegno Team: M.C. Ciocchi, B. Malengier

December 22, 2018

1 Introductie

In dit boekje zullen we je leren hoe je de Fe Cube kunt maken. Je zal leren over LEDs, de Arduino, multiplexing, solderen, kleurentheorie, en meer! Je kan een kit kopen met

alle onderdelen van Ingegno.be, of je kan je eigen stukken gebruiken.

De code van deze handleiding kan gevonden worden op [de Ingegno github pagina](#).

1.1 De componenten

We zullen volgende componenten gebruiken:

1. Een schakelbord (Breadboard)
2. 9 RGB-LED + 3 extra (we veronderstellen gemeenschappelijke kathode voor de kubus)
3. 1 grote RGB-LED (we veronderstellen gemeenschappelijke anode)
4. 3 NPN + 1 extra (2N3904 of gelijkaardig)
5. 1 drukknop
6. 9 220 Ω weerstanden + 3 extra

7. 1 10k-20k Ω oftewel 10000-20000 Ω weerstand

8. veel draden

9. lasercut basis voor de kubus

10. ijzerdraad pilaren voor de kubus

11. soldering plaat voor vaste componenten + connectiestukken

Je zal ook een soldeerijzer nodig hebben, en optioneel kun je een afdekking maken voor de kubus, bv via 3D printen of in plexi met de lasercutter.

Voor een overzicht van de betekenis van de componenten, zie het overzicht die je apart gekregen hebt.

1.2 Arduino

We zullen intensief gebruik maken van Arduino, dus moet je de [Arduino software](#) installeren op je laptop. Eenmaal geïnstalleerd kun je nu je Arduino bord koppelen aan je computer via de USB poort. In de pro-

grammeeromgeving van Arduino kun je programma's schrijven en die opladen naar de Arduino, waar ze dan uitgevoerd worden zolang er stroom is.

1.3 Programmeren met Arduino

Een programmeertaal moet verstaan worden door een computer. Spijtig genoeg zijn computers nog altijd een beetje dom, dus mag je geen enkele fout maken! Het is zoals een dictée waar je altijd 10/10 moet halen. Neem

dus de tijd om de structuur en taal van de Arduino te leren. Zie de blaadjes 'Programmeren met Arduino' die je bij deze gids kan downloaden.

2 Les 1: Één LED circuit

We maken een circuit met één enkele LED, welke we zullen programmeren om te pinken. De Arduino levert 5 Volt, terwijl de LED gemaakt is voor 3 Volt. We moeten

dus een weerstand gebruiken om het voltage over de LED te verminderen. Je kan de weerstand die je nodig hebt berekenen met <http://led.linear1.org/1led.wiz>.

2.1 Één pin

Maak het circuit zoals gegeven in Fig. ?? . Gebruik je schakelbord en draden om de connecties te maken. Dus hier, een draad van pin 2 naar rij 10 op je schakelbord. Dan de

resistor met weerstand $R=330\ \Omega$ van rij 10 naar rij 6, de LED anode van rij 6 naar rij 5, en uiteindelijk een draad van rij 5 naar de GND pin op de Arduino.

Nu moeten we een programma schrijven dat de stroom naar pin 2 aan en af zet elke

seconde, zodat de LED pinkt. Je vindt de code in Code ??.

Code 1

```
1  /*
2  Controlling a simple led via anode current interrupt
3
4  Schematic:  GND to LED kathode (-), LED anode to resistor R1,
5              R1 to pin led1
6              Use correct R1, Arduino at 5V and led of 3V will require typically R1
              =220 - 500 Ohm
7  */
8
9  // led anode (+) is connected to 2,
10 // led kathode is connected to GND
11 int led1=2;
12
13 // we can control the led in one way:
14 // 1. we can interrupt the current coming from the anode via pin led1
15
16 void setup() {
17   pinMode(led1, OUTPUT);
18 }
```

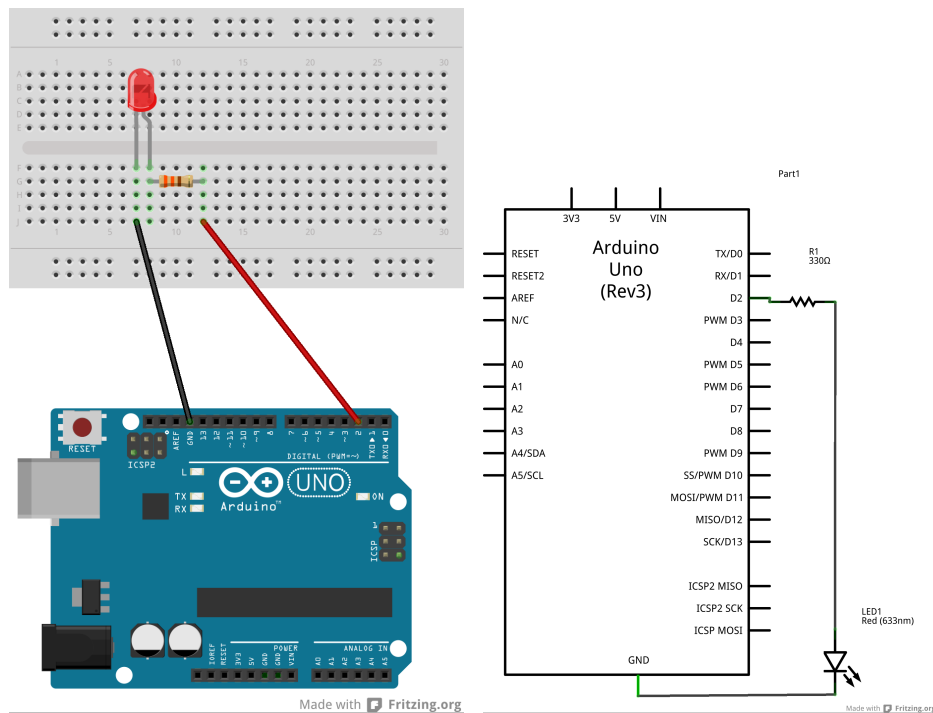


Figure 1: Met een pin en de grond kunnen we een LED doen branden. Links met breadboard, rechts als schema.

```

19
20 void loop(){
21   digitalWrite(led1,HIGH); //current on
22   delay(1000); // 1 sec = 1000 millisec delay
23
24   digitalWrite(led1,LOW); //current off
25   delay(1000);
26 }

```

Proficiat, je hebt je eerste programmeerbare LED gemaakt.

- Opdracht 1**
1. Verander nu je programma (Code 1) zodat de LED elke 3 seconden pinkt, met 1 seconde af.
Vervolgens, wat gebeurt er als je elke 40 milliseconden pinkt en als het elke 10 milliseconden pinkt?
 2. Wat moet je doen als je twee LED's wilt doen pinken? of afwisselend pinken? Probeer het eens.

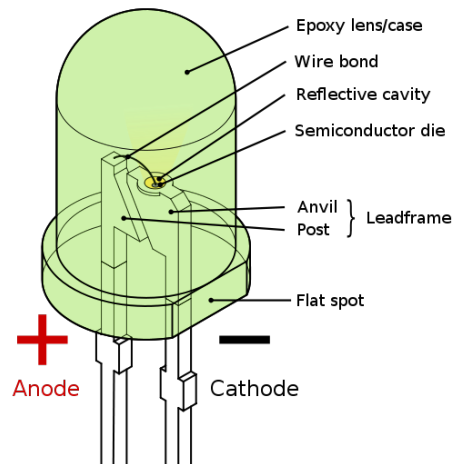


Figure 2: Een LED in het groot

Hoe werkt ons circuit? Wel, de pinnen van 1 tot 13 kunnen een HIGH staat en een LOW staat hebben. HIGH betekent dat de pin 5V krachtbron heeft, terwijl LOW betekent dat de pin zich in de grond staat bevindt. We zeggen dat stroom sroomt van de positieve voltage van 5V (de + zijde) naar de GND (de - zijde). dat is een beetje vervelend gezien de echte kwantumdeeltjes die hier gebruikt worden, electronen genoemd, eigenlijk de andere kant op stromen, dus van de min naar de plus. Dit omdat elektronen zelf negatief zijn, en dus afgestoten worden door de negatieve lading van de GND. Theoretisch zou je evenwel andere kwantumdeeltjes kunnen gebruiken! Hoe dan ook, de richting van de stroom, van + naar -, is gewoon een afspraak tussen wetenschappers om geen verwarring te doen ontstaan.

De krachtbron is hier 5 Volt (5V), wat een indicatie is van hoeveel werk je kan doen. Veel kracht betekent dat je veel werk kan doen! Om een LED te doen branden hebben we evenwel niet veel kracht nodig, dus beperken we die door een resistor toe te voegen die de kracht over de LED zal beperken.

Dit verhoogt de levensduur van de LED.

De resistor of weerstand doet wat de naam zegt: het resisteert of weerstaat de stroom. Bijgevolg verliezen de kwantumdeeltjes een deel van hun kracht als ze door een weerstand stromen. Niet alleen zullen de deeltjes minder kracht hebben erna, er zullen ook minder deeltjes passeren elke seconde. De stroom is dus verminderd. Een weerstand of resistor zal dus de stroom verminderen en na het passeren zal er ook nog eens minder kracht over zijn om door de rest van het circuit te stromen.

Daarna komt de LED voor de stroom. Alle overblijvende kracht zal gebruikt worden om door de LED te gaan en licht te genereren. Een LED maakt alleen licht als de anode (lange been, +) is geconnecteerd naar het voltage (de positieve zijde), en de kathode (korte been, -) naar de GND zijde. Als je de LED ronddraait (probeer het!) zal er geen licht schijnen. Er zal ook maar heel weinig stroom vloeien dan, gezien de LED niet enkel geen licht maakt dan maar ook de stroom sterk zal weerstaan. Een LED in het groot zie je in Figuur ??.

Opdracht 2 Bestudeer het schema aan de rechterkant van Figuur ???. Een schema is vaak duidelijker om mee te werken dan de draden die je aan de linkerkant ziet. Elke component heeft zijn eigen symbool. Een LED is een driehoek die eindigt op een korte verticale lijn, en twee kleine pijlen die het licht voorstellen die de LED uitzendt. De driehoek in dit symbool kun je interpreteren als een grote pijl die de richting van de stroom aangeeft: van + naar -. Het licht verlaat de LED in dezelfde richting in het symbool.

Je wilt weten hoe de LED licht produceert? Wel, dat is kwantumfysica, en de meeste mensen leren daar nooit over. Laat ons het een beetje uitleggen. Om kwantumfysica te verstaan moet je met kwantumdeeltjes werken, dus hier de elektronen. Weet je nog dat de elektronen stromen van de GND naar de pin 2, dus van de kathode (-) naar de anode (+) van de LED?

Aan de kathode zijde zit wat we een n-type materiaal noemen. Dit is een materiaal die een overschot aan negatieve elektronen heeft, daarom n. Aan de anode zijde zit een p-type materiaal. Dat is een materiaal die een tekort aan elektronen heeft, wat we gaten noemen: het is de afwezigheid van een elektron in het materiaal. Het p-type materiaal laat stroom door door gaten rond te bewegen. Het n-type materiaal laat stroom door elektronen rond te bewegen.

Dus, als elektronen toekomen aan de kathode creëren ze daar een surplus aan elektronen, en de elektronen worden geduwd in de richting van het p-materiaal. Daar aangekomen vallen ze in een van de gaten, en het is dit kwantumeffect dat een klein kwantum aan energie creëert, welke een foton is wat we waarnemen als het licht van de LED. Dit effect noemen we *Electroluminescentie*, en het gebeurt enkel bij overgang van een elektron van een n-type materiaal naar een p-type

materiaal. Tussen andere materialen die elektronen doorlaten kunnen de elektronen altijd gradueel hun energie verliezen, in plaats van via een kwantum.

Nu, na het vallen in het gat zijn er minder gaten in het p-type materiaal aan de kant van het n-type materiaal, dus zullen de gaten bewegen in die richting om terug een gelijke balans in het materiaal te hebben. Hierdoor eindigen we met een tekort aan gaten aan de kathode zijde, welke op zijn beurt ervoor zal zorgen dat een elektron het materiaal zal verlaten (het materiaal houdt van zijn aantal gaten en wil het zo houden), waardoor een nieuw gat ontstaat daar.

Als je de LED omdraait, connecteer je het n-type materiaal aan de positieve krachtbron. De elektronen van het n-type materiaal stromen dan gewoon naar deze krachtbron, zo een tekort aan elektronen veroorzakend daar. Dit kan enkel gecorrigeerd worden door een elektron dat het p-type materiaal verlaat, en zo een nieuw gat creëert, en een nieuw elektron in het n-type materiaal. Dit evenwel zal duidelijk geen licht produceren (gezien het elektron niet in een gat valt), maar de elektronen bieden ook heel veel weerstand in het bewegen van het p-type materiaal naar het n-type materiaal (ze zijn bij wijze van spreken comfortabel waar ze zijn, het materiaal houdt van zijn aantal gaten en wil het zo houden).

2.2 LED met NPN, twee pinnen

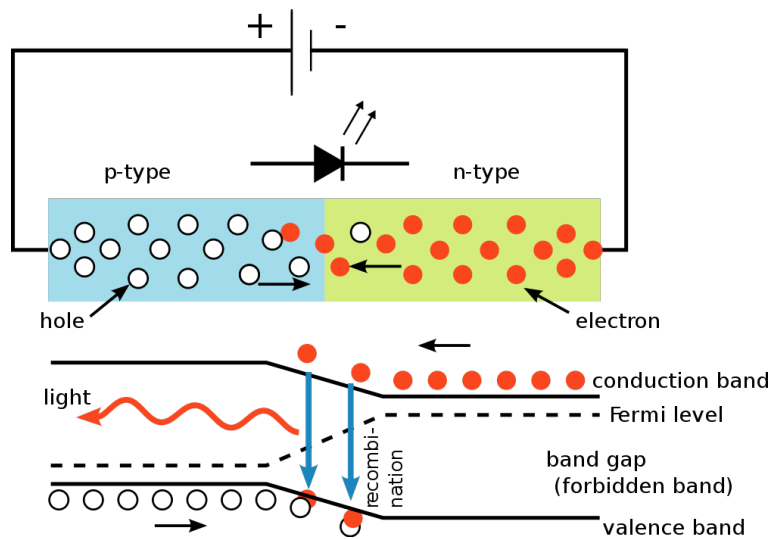


Figure 3: Een PN overgang zoals gebruikt in een LED (©Wikimedia Commons S-kei)

De LED controleren werkte perfect, maar later zullen we de mogelijkheid willen om de stroom af te sluiten aan de Kathode zijde (-), dus aan de grond (GND). In deze sectie zullen we dat leren. Een mogelijkheid is om een schakelaar te gebruiken: een component die aan of af is afhankelijk van een signaal. Net

zoals de lichtschakelaars die je dagelijks gebruikt, en waar de input de druk van je hand is. We zullen de kleinst mogelijke schakelaar gebruiken: een NPN brug. Dit componentje is de halve zwarte cilinder met 3 poten in je doos, zie Fig. ??.

Zoals je in die figuur ziet is er geen standaard om de beentjes van een transistor te identificeren. Normaal zal het als volgt zijn voor degene die jij zult gebruiken: hou je het rechtop en kijk je naar het vlakke stuk van de NPN, dan is de linkerpoot de in-zijde voor de stroom (C, van collector, waar er gecollecteerd wordt), de rechterpoot de uitgangzij-

jde (E, van emitter of extruder) die naar de GND gaat, en het middelste been is de P of positieve controle (B, van basis). Als er een positieve stroom is op P, laat de NPN brug stroom door. Is er geen stroom op de P, dan kan er geen stroom door en is het circuit dus onderbroken. Het complete circuit is gegeven in Fig. ??.

Nu moeten we ons programma aanpassen zodat ook de pin naar de NPN gebruikt wordt om de LED te doen blinken. Om vast te stellen dat de NPN werkt zoals verwacht laten we de LED door drie standen gaan, elk voor een seconde:

1. Beide Arduino pinnen aan

2. Enkel de anode pin aan

3. Enkel de pin naar de NPN aan.

Correct gedrag zou dan moeten zijn dat de LED aan is voor een seconde en uit voor twee seconden.

Code 2

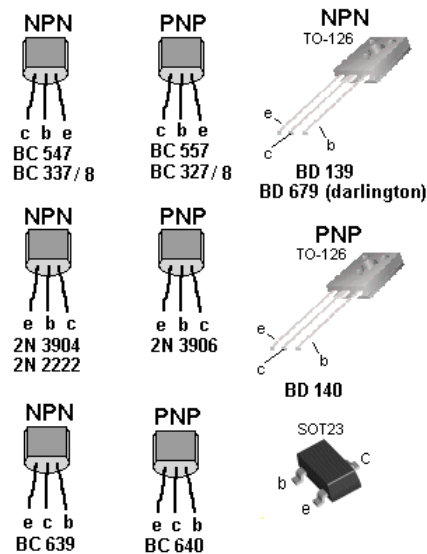


Figure 4: Verschillende NPN en PNP transistoren. Er is geen standaard voor de beentjes, refereer naar een datasheet, of probeer het met een LED zoals wij hier doen. Als het niet werkt, gebruik andere beentjes zoals in deze Figuur!

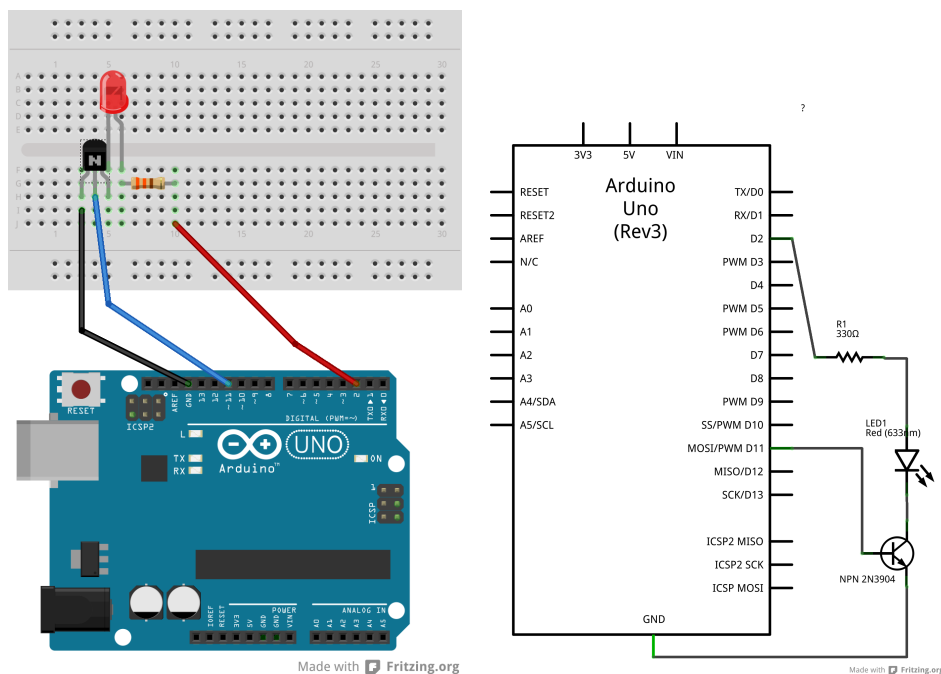


Figure 5: Met twee pinnen en een NPN brug kunnen we de LED afzetten aan beide kanten.

```

1  /*
2  Controlling a simple led via an npn switch and a anode current interrupt
3  */
4
5  // led anode (+) is connected to 2,
6  // led kathode is connected to GND via an npn transistor
7  int led1=2;
8  // npn transistor control is connected to 11 (middle leg)
9  int npn=11;
10
11 // we can control the led in two ways:
12 // 1. we can interrupt the GND current to the led via pin npn
13 // 2. we can interrupt the current coming from the anode via pin led1
14
15 void setup() {
16   pinMode(led1,OUTPUT);
17   pinMode(npn,OUTPUT);
18 }
19
20 void loop(){
21   //we cycle through 3 states.
22   //This is on:
23   digitalWrite(led1,HIGH);
24   digitalWrite(npn,HIGH);
25   delay(1000);
26   //This is off, no current to led
27   digitalWrite(led1,LOW);
28   digitalWrite(npn,HIGH);
29   delay(1000);
30   //This is off due to NPN
31   digitalWrite(led1,HIGH);
32   digitalWrite(npn,LOW);
33   delay(1000);
34 }

```

Je zou graag weten hoe de NPN werkt? Wel, dit is een transistor, specifiek een Bipolaire transistor. De LED die we hiervoor gezien hebben is een PN overgang. Onze NPN nu is inderdaad een combinatie van 3 materialen die geconnecteerd worden: een n-type materiaal, een p-type materiaal, en dan opnieuw een n-type materiaal. Deze keer heeft het p-type materiaal maar weinig gaten. Als we het verbinden met de GND (-) zullen er nog minder gaten zijn. Zoals we uitgelegd hebben, een p-type materiaal geleidt stroom via de gaten, dus kan er geen stroom vloeien, en is de overgang gesloten.

Als we een voltage over het p-type materiaal plaatsen, verliest het materiaal zijn elektronen, en krijgt dus meer gaten. Dit

betekent dat we nu stroom kunnen hebben van het p-materiaal welke we de basis noemen, naar de emitter, welke een van de twee n-materialen is. Dit is hetzelfde als de PN overgang in de LED. De emitter zendt elektronen in de basis (*emit* is Engels voor uitzenden).

Goed, hoe krijgen we dan stroom van het n-type materiaal naar het andere n-type materiaal? Dat zou niet mogelijk mogen zijn! De truck is dat het p-type materiaal eenmaal verbonden met een voltage (+) extra gaten krijgt, en het kan een lijn van gaten vormen van het emitter n-type materiaal naar het collector n-type materiaal. Nu gebeurt iets speciaals: de elektronen kunnen springen van gat naar gat via deze lijn van de emitter naar

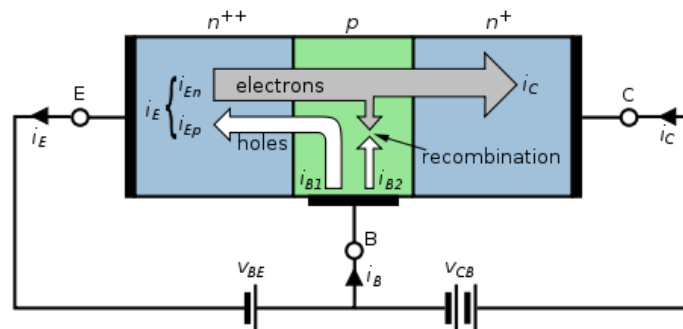


Figure 6: Een NPN transistor als de overgang open staat.

de collector. En wat meer is, ze kunnen dat veel, veel, vlugger doen dan dat de gaten zelf kunnen bewegen. In andere woorden, we krijgen een versterking van de stroom aan de basis, zie Fig. ?? . Als we de stroom aan de basis verhogen, stromen er nog meer gaten in het p-type materiaal, en kunnen er meer 'verbindinglijnen' tussen emitter en collector. We kunnen dus een klein signaal aan de

basis omvormen tot een groot signaal over de NPN.

Merk op dat we in ons circuit een 5V signaal over de basis plaatsen, welke op zijn eentje al genoeg is om de LED te doen branden. Hierdoor is onze stroom aan de basis al genoeg om de LED aan te sturen. Dit is geen probleem, gewoon iets waar je je bewust van moet zijn.

Opdracht 3 We kunnen een LED aan en af schakelen met een NPN transistor. We zouden in staat moeten zijn om de stroom over de Basis van de NPN te verminderen en toch een werkend circuit overhouden. Probeer dit, voeg een weerstand toe tussen pin 11 en de basis van de NPN. Hoe hoog kan je gaan met de resistor? Zou dit een manier kunnen zijn om de sterkte van de resistor te testen?

2.3 Fade: Analooq puls breedte moduleren

Als je de opdracht uit de eerste sectie gedaan hebt, weet je dat door vlug te blinken, je de illusie kunt creëren dat de LED altijd aan is. Evenwel, de LED is niet de hele tijd aan, bijgevolg was de LED wel minder helder! Dus, we kunnen beïnvloeden hoe helder een LED is door deze te doen blinken. De truck is te verhinderen dat de mensen hem effectief zien blinken. Gelukkig hebben mensen persistentie van beelden, denk aan de [Phenakistiscoop](#), uitgevonden door de Belg Plateau in Gent. Persistentie van beelden is

het fenomeen waarbij een beeld dat maar een fractie van een seconde gezien is, door je brein zal blijven gezien worden, zelfs nadat het originele beeld verdwenen is of bewogen heeft. Dit is het principe achter film en televisie. Door de LED vlug aan en uit te schakelen, kunnen we onze brein bedotten in het zien van een "gemiddelde" waarde van helderheid. Mensen zien 24 beelden per seconde, als we dus aan en uit schakelen in ongeveer 40 milliseconden, zullen we niet in staat zijn te zien dat de LED aan het blinken is, maar zullen

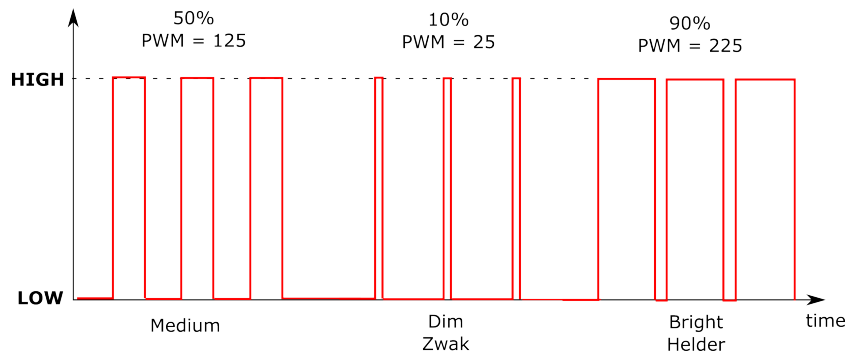


Figure 7: Hoe PWM in de praktijk werkt: over de pin wordt een blok golf gegenereerd.

we in plaats daarvan een gedimd licht zien!

We zouden de `delay` functie van Arduino kunnen gebruiken om dit te programmeren: de LED aan en uit zetten voor een bepaalde tijd. Maar het kan eenvoudiger: Puls Breedte modulatie (PWM). In deze wordt een blok golf gegenereerd over een digitale pin van de Arduino. Een blok golf is een golf tussen 0 (uit) en 1 (aan). In Arduino doe je dit met de `analogWrite` functie op een digitale pin. Zie het schematisch in Fig. ???. Merk op dat oudere Arduino's dit enkel toelaten op pin

9,10, en 11! Controleer waar jouw versie van Arduino dit toelaat, je herkent de pins op het bord met het teken ~

De functie wordt `analogWrite` genoemd omdat het een golf is over een digitale pin. Analooq refereert altijd aan golven, terwijl digitaal refereert naar nullen en eenen. Als je dus een golf maakt met 0 en 1 (zie de figuur dat het inderdaad op een golf lijkt), noemen we het opnieuw analooq. De functie `analogWrite` laat waarden toe van 0 (=altijd af) tot 255 (=altijd aan).

De volgende vraag is op welke pin we de `analogWrite` moeten toepassen. Zoals we geleerd hebben in het vorige stuk: het doet er niet toe, beide pin 2 en pin 11 zullen de LED onderbreken. Wij zullen het toepassen op de NPN pin, gezien dat degene is die we bij de Fe cube zullen gebruiken. Het circuit is gegeven in Fig. ???. Als je vergelijkt met Fig. ??, zie je dat het circuit hetzelfde is. Zie dan de code hieronder om te zien hoe PWM gebruikt wordt .

Merk op dat we in de code enkele nieuwe trucks gebruiken. We gebruiken een willekeurig (=random) getal om te bepalen hoe helder de LED moet zijn, en we wijzigen de helderheid elke seconde. Hiertoe hebben we 2 nieuwe begrippen nodig:

- `randomSeed(analogRead(0))`. Een random

getal is enkel volledige willekeurig indien we niet kunnen raden waar het start. Hier-toe moeten we de random getallen generator zaaïen (seed in het Engels) met een begin-getal. We doen dit hier door het lezen van de waarde op de niet gebruikte analoge pin 0. Gezien deze analoge pin bij ons met niets geconnecteerd is, zal er enkel willekeurige storing erop zitten, welke we niet vooraf kunnen raden. Dit zal ons ongekend begin zijn, zodat we zeker zijn dat de serie helderheid die we zien telkens anders is als we de Arduino heropstarten.

- `random`. Deze Arduino functie genereert een willekeurig getal. Door te schrijven `random(256)` bekomen we een willekeurig getal tussen 0 en 255, zie <http://Arduino.cc/en/Reference/random>.

Code 3

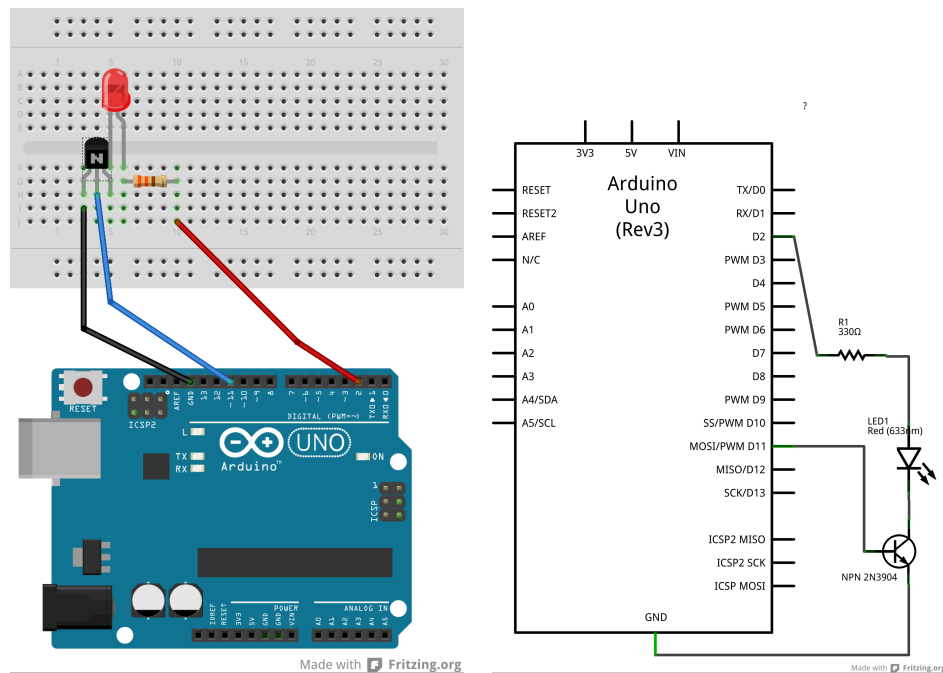


Figure 8: Het circuit om PWM te gebruiken om de helderheid te wijzigen. Ja, het is identiek aan het eerdere circuit, alle wijzigingen zijn in software.

```

1  /*
2  Brightness is controlled via PWM of a single LED
3  */
4
5  int led1=2;
6  // npn transistor control is connected to 11 where we do PWM
7  int npn=11;
8
9  void setup() {
10   // we use analog pin 0 as input for random seed generator.
11   // randomSeed() will shuffle the random function.
12   randomSeed(analogRead(0));
13   pinMode(led1, OUTPUT);
14   //nnp pin will be used for PWM, no setup needed!
15 }
16
17 void loop(){
18   //we obtain a brightness as a random value between 0 and 255
19   int rand = random(256);
20   digitalWrite(led1, HIGH);
21   // Apply PWM to pin npn, this will result in varying brightness
22   analogWrite(npn, rand);
23   delay(1000);
24 }

```

2.4 Draai het om! Hoe een digitale pin gebruiken als grond.

We hebben een digitale pin gebruikt voor de plus, en de grond voor de min. In realiteit was de pin op 5 Volt boven de grond. De grond kunnen we dus interpreteren als 0 Volt. Je zou kunnen denken dat dit betekent dat de pinnen altijd positief moeten zijn, maar dat is niet waar. Een pin kan zowel positieve stroom leveren als negatieve stroom, afhankelijk van de plaats in het circuit. Het is evenwel belangrijk om er rekening mee te houden dat een digitale pin maar een beperkte hoeveelheid stroom kan leveren: 40 mA (milliamps) is beschikbaar voor gebruik in het circuit. Dit is genoeg stroom om een LED helder te doen oplichten (vergeet evenwel geen weerstand in serie te plaatsen), of om verschillende sensoren aan te drijven, maar niet genoeg om een relay of motor aan te drijven. Meer stroom trekken uit een digitale pin zal de pin kapot maken, en misschien wel je volledige Arduino bord! Wees dus voorzichtig in hoe je ze gebruikt.

Als je meer stroom nodig hebt, gebruik de beschikbare volt pinnen op het bord. Je ziet een 3.3V (tot 50mA stroom beschikbaar) en 5V pin (stroom afhankelijk van hoe je Arduino stroom krijgt, via USB of batterij).

Test dit.

1. Maak het circuit en laad de code van Code ???. In plaats van de positieve zijde te verbinden met pin 2, connecteer ze met 3.3V en dan met 5V. Je zal zien dat de LED zonder probleem werkt, maar je kan natuurlijk niet langer de

LED afzetten door signalen te sturen naar pin 2.

Wat je moet onthouden is dat je voor grote projecten de voltage pinnen op de Arduino moet gebruiken om stroom te leveren: 3.3V, 5V en GND; terwijl je de genummerde pinnen gebruikt voor kleine dingen: een sensor, een LED, of een NPN transistor om stroom te onderbreken.

2. Om te experimenteren met de mogelijkheden, laat ons het vorige voorbeeld helemaal omdraaien, en de grond verwijderen uit het circuit door een digitale pin te gebruiken als grond. Jou circuit zou eruit moeten zien als de linkse uit Fig ??. Belangrijk hier is dat we pin 2 moeten plaatsen op 0V. Dit doen we met `digitalWrite(2, LOW)`; . Dat is de enige wijziging in de code, maar het circuit is wel omgedraaid!
3. Gezien dit een LED is gebruiken we minder dan 40mA stroom, en kunnen we dus ook het input voltage vervangen door een pin, zoals gedaan in de rechterfiguur van Fig ??.

Het zou duidelijk moeten zijn dat we de digitale pinnen op verschillende manieren kunnen gebruiken. Wees je evenwel goed bewust van de limitatie van de pinnen: je kan enkel 40mA uit een digitale pin krijgen. Als je meer nodig hebt, gebruik de voltage pinnen en de GND pin.

Merk het volgende op in Code ???: gezien pin 2 zich als grond moet gedragen dienen we

`digitalWrite(2, LOW)` op de pin uitvoeren.

Code 4

```
1  /*
2  Brightness is controlled via PWM of a single LED
3  */
4
5  // led - is connected to 2,
```

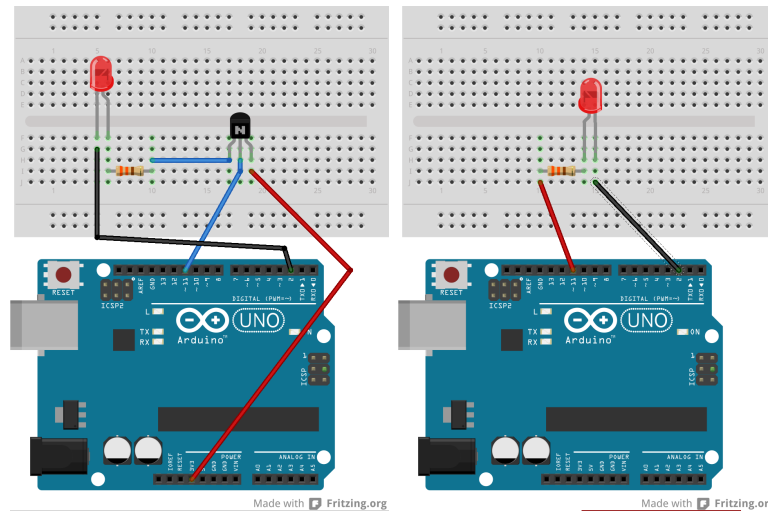


Figure 9: Met twee pinnen en een NPN brug kunnen we de LED afzetten aan beide kanten. We gebruiken nu pin 2 zodat we de pin als grond kunnen gebruiken. Rechts: De NPN verwijderen zorgt voor een eenvoudiger circuit.

```

6 // led + is connected to 3.3V via an npn transistor and a resistor
7 int led1=2;
8 // npn transistor control is connected to 11 where we do PWM
9 int npn=11;
10
11 void setup() {
12 // we use analog pin 0 as input for random seed generator.
13 // randomSeed() will shuffle the random function.
14 randomSeed(analogRead(0));
15 pinMode(led1,OUTPUT);
16 }
17
18 void loop(){
19 //we obtain a brightness as a random value between 0 and 255
20 int rand = random(256);
21 digitalWrite(led1,LOW);
22 // Apply PWM to pin npn, this will result in varying brightness
23 analogWrite(npn, rand);
24 delay(1000);
25 }

```

3 Les 2: Een RGB LED circuit

3.1 Het RGB circuit

LED bestaan in verschillende kleuren. Sommige LED kunnen een kleurenbereik produceren. Dit zijn RGB LEDs, zie een afbeelding in Fig. ???. Ze zijn in essentie een rode,

groene en blauwe LED in een enkel pakket, met een gedeelde anode of een gedeelde kathode. Om deze LED aan te drijven breiden we ons 1 LED met NPN circuit (Code 2) uit.

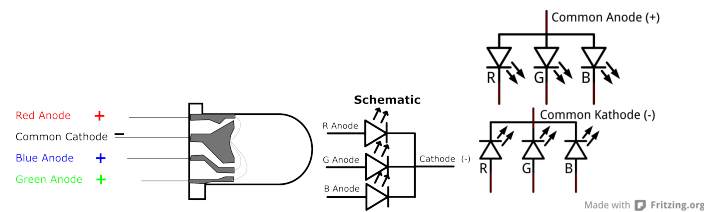


Figure 10: Links een vergroting van een RGB LED met gedeelde anode, Rechts een schema zoals je in circuits kan vinden.

We veronderstellen dat je een RGB LED hebt met 3 kathoden en 1 anode, dus zullen we 3 NPN bruggen nodig hebben, en 1 resistor.

Merk Op: Indien je een LED gebruikt

met een gedeelde kathode dien je het circuit te inverteren zoals gezien in de vorige les. Zie ook de kubus later waar dit het geval is.

Opdracht 4 Test je RGB LED. Er zijn dus 4 beentjes aan de RGB LED. Het langste been is normaal de gemeenschappelijke kathode (-, verbindt met GND) of de gemeenschappelijke anode (+, verbindt met een voltage). Je zal je LED moeten testen om te weten wat wat is. Doe dit nu: plaats de RGB LED op je breadboard, connecteer een 330Ω weerstand van de rij met het langste been naar een andere vrije rij op je bord. Neem nu twee draden, en plaats een in de GND, en de andere in de 3.3V output. Je Arduino zet je aan, verbindt dus de USB kabel met je PC. Onderstel eerst dat het een gemeenschappelijke kathode LED is, dus verbindt de GND draad met de weerstand. Met de 3.3V draad test je de beentjes van de LED. Het werkt niet? Verwijder de draden van het schakelbord en test of het een gemeenschappelijke anode LED is. Dus, verbindt de 3.3V draad met de weerstand, en test de andere LED beentjes met de GND draad.

Tip: Maak een schets op papier van je LED, waarbij je noteert welk been Rood, Groen en Blauw is, alsook de kathode/anode.

Het circuit is gegeven in Fig. ?? . Enkele dingen om rekening mee te houden: De NPN uit (E) gaan allemaal naar de grond (GND), plaats ze dus op een enkele rij op je schakelbord. Voor de andere connecties heb je verschillende draden nodig: 4 controle pinnen en de grond, dus 5 draden.

Eenmaal je circuit gemaakt is, is het vervolg van deze les het programmeren van de LED. Start met een simpel programma om de LED te laten lopen door Rood, Groen en Blauw. Als je klaar bent, verifieer dat je inderdaad deze orde ziet, indien niet heb je

een fout gemaakt met het bedraden van de kathoden. Als je twee kathoden terzelfdertijd onder stroom plaats, zul je soms een mooie gemengde kleur krijgen, en soms niet. Dit is omdat de weerstand via een kleur niet identiek is aan die van een andere kleur, waardoor de stroom via een preferentieel pad zal lopen. Goedkope RGB LED zijn niet zo goed om kleuren correct te mengen door twee of meer kathoden terzelfdertijd aan te sturen. We zullen later enkele trukjes moeten gebruiken om kleuren mooi te mengen. Hier onze Arduino sketch voor RGB:

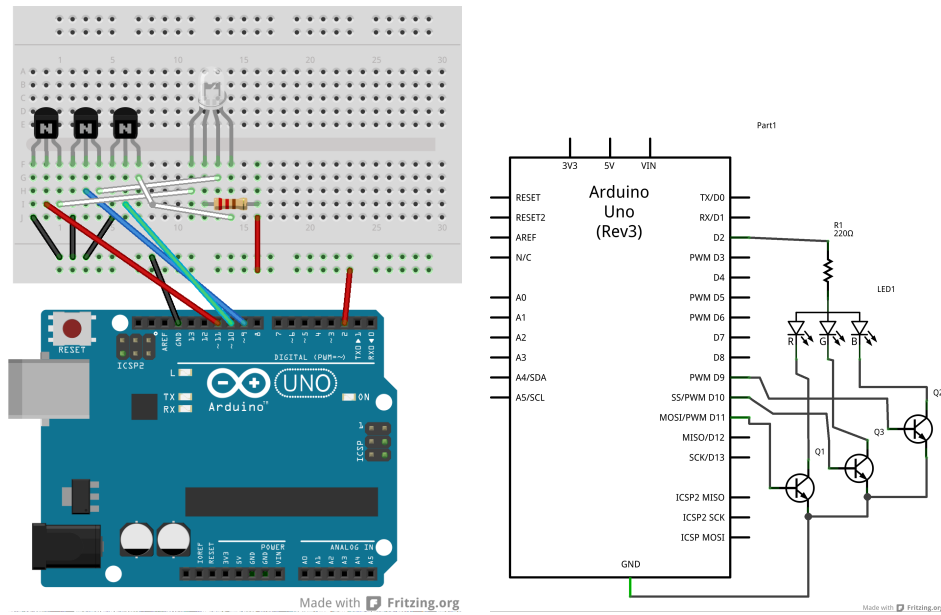


Figure 11: Circuit voor een gedeelde anode RGB LED.

Code 5

```

1  /*
2  Controlling an RGB led via npn switches
3  */
4
5  // pins used:
6  int ledR=11;
7  int ledG=10;
8  int ledB=9;
9  int led1=2;
10
11 // we can control the led in two ways:
12 // 1. we can interrupt the GND current to the led via pin npn
13 // 2. we can interrupt the current coming from the anode via pin led1
14
15 void setup() {
16   pinMode(ledR,OUTPUT); pinMode(ledG,OUTPUT);
17   pinMode(ledB,OUTPUT); pinMode(led1,OUTPUT);
18 }
19
20 void loop(){
21   digitalWrite(led1,HIGH);
22   //red
23   digitalWrite(ledR,HIGH);
24   digitalWrite(ledG,LOW);
25   digitalWrite(ledB,LOW);
26   //green
27   delay(1000);
28   digitalWrite(ledR,LOW);

```

```

29 digitalWrite(ledG,HIGH);
30 digitalWrite(ledB,LOW);
31 delay(1000);
32 //blue
33 digitalWrite(ledR,LOW);
34 digitalWrite(ledG,LOW);
35 digitalWrite(ledB,HIGH);
36 delay(1000);
37 //Red and Green: normally not working, shows as red
38 digitalWrite(ledR,HIGH);
39 digitalWrite(ledG,HIGH);
40 digitalWrite(ledB,LOW);
41 delay(1000);
42 //Green and blue
43 digitalWrite(ledR,LOW);
44 digitalWrite(ledG,HIGH);
45 digitalWrite(ledB,HIGH);
46 delay(1000);
47 }

```

3.2 Kleuren mengen

In Code ?? hebben we getoond hoe we een LED licht kunnen dimmen. We gebruiken dit nu om kleuren te mengen. Als we erin slagen om eerst rood, dan groen, en dan blauw te tonen, met de kleuren correct gedimd, dan kunnen we kleuren mengen. Bevoorbeeld, als we blauw afzetten zullen we rood+groen=geel krijgen. De truck om dit te doen werken zal weer zijn dat we dit allemaal zo snel moeten doen dat door persistentie van visie we ons ook kunnen doen geloven dat we op een niet pinkende manier kleuren mengen

Maak dus een Arduino sketch om een willekeurige gemengde kleur te tonen voor een enkele seconde, om daarna een andere willekeurige kleur te tonen. Om een kleur te dimmen gebruiken we weer `analogWrite`. We hebben gezien dat `analogWrite` een blokgolf

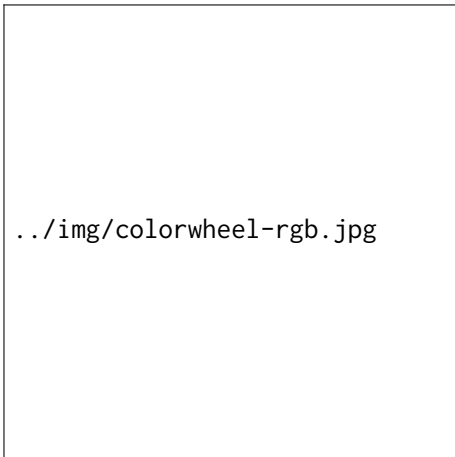
maakt en dat we 256 verschillende blokgolven kunnen verkrijgen, van 0 tot 255. Het nummer 256 komt niet zomaar uit de lucht vallen. We hebben dat $256 = 2^8 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$, het is dus 2 die 8 keer met zichzelf vermenigvuldigd wordt. Dat is het grootste nummer die je kunt maken met een enkele byte (=8 bits) van het Arduino geheugen. De zo genaamde 8 bit byte is de eenheid gebruikt in de allereerste computers om informatie mee op te slaan.

Het is erg handig dat 256 het verloop is dat gebruikt wordt in `analogWrite`, gezien het originele **RGB kleurenmodel** gebaseerd is op dit getal. De kleuren die wij zullen maken met onze RGB LED zullen dus overeenkomen met die van het RGB kleurenmodel, zie Fig. ??.

De gebruikte code vind je in Code ??. Verschillende nieuwe concepten zijn aanwezig.

- `Serial.begin` and print: `Serial` laat toe om je Arduino te observeren vanuit je PC. Het opent een trage connectie naar waar je prints kunt sturen. In de sketch doen we dat via een vlag: `bool test`, dit is een vari-

abele die we een vlag noemen: het kan twee waarden hebben: waar of vals, welke in het Engels zijn *true* of *false*. Als de variabele *true* is, dan printen we op de seriële connectie de waarden van de willekeurig bepaalde Rood, Groen, en Blauw. We doen dit met het commando `Serial.print`. Om ervoor te zorgen dat de volgende waarde van RGB op



../img/colorwheel-rgb.jpg

Figure 12: Een deel van de mogelijke kleuren met RGB LEDs

een nieuwe lijn komt, eindigen we met `Serial.println`, die laatste *ln* staat voor 'lijn'

- `millis()`: Dit is een functie die het aantal milliseconden sinds de opstart van de Arduino teruggeeft. Je kan functies opzoeken in de [Reference guide](#). Het maximum getal die je kunt terugkrijgen komt overeen met 50 dagen. Daarna herstart het van 0.

- Een `if` test. Dit is een conditie. Als het stuk achter de `if` in normale haakjes waar (true) is, dan wordt het stuk in gekrulde haakjes uitgevoerd, anders niet. De syntax is:

```
if (conditie){... commando's ...}
```

dus 'commando's' worden enkel uitgevoerd als conditie waar is.

- Een `while` lus: Dit is een repetitie gebaseerd op een conditie. Zolang de conditie waar is zal het stuk in de gekrulde haakjes uitvo-

erd worden. De syntax is:

```
while (conditie){... commando's ...}
```

dus 'commando's' zullen uitgevoerd worden zolang conditie waar is. Opgelet hier, het is gemakkelijk om een while loop te maken die nooit stopt. Het is belangrijk dat de conditie iets is dat regelmatig aangepast wordt. In onze code is de conditie:

```
currentTime - prevTime < 1000
```

dus zolang de huidige tijd (*currentTime*) min the vorige tijd (*prevTime*) minder dan een seconde is, herhalen we de commando's. Er zal een moment komen waarop de conditie vals is en de lus stopt. Dit omdat je als laatste lijn in de gekrulde haken van de while ziet:

```
currentTime = millis();
```

dus de variabele *currentTime* neemt constant toe, en de while loop zal inderdaad stoppen.

Code 6

```
1  /*
2  Controlling an RGB led via an npn switches and PWM
3  Here a random color is shown for 1 sec
4  */
5  // pins used:
6  int ledR=11;
7  int ledG=10;
8  int ledB=9;
9  int led1=2;
10 // delay for each PWM. Play with it: 1, 4, 10
```

```

11 int delayDuration = 4;
12 // testing
13 bool test = true; //set = true for monitor!
14
15 // global variables to hold the color state of a led1
16 long randR, randG, randB;
17
18 //we need access to current time frequently
19 unsigned long currentTime = millis();
20 unsigned long prevTime;
21
22 void setup() {
23     //for debugging we set up serial communication. This uses pins 0 and 1!
24     Serial.begin(9600);
25     // we use analog pin 0 as input for random seed generator.
26     // randomSeed() will shuffle the random function.
27     randomSeed(analogRead(0));
28     // define the pins we use for the RGB led, not needed for PWM feeds !
29     pinMode(led1, OUTPUT);
30 }
31
32
33 void loop(){
34     digitalWrite(led1, HIGH);
35     analogWrite(ledR, 0);
36     analogWrite(ledG, 0);
37     analogWrite(ledB, 0);
38     randR = random(256); // randR = 5;
39     randG = random(256); // randG = 252;
40     randB = random(256); // randB = 9;
41     if (test) {
42         Serial.print("color: "); Serial.print(randR); Serial.print(" - ");
43         Serial.print(randG); Serial.print(" - "); Serial.println(randB);
44     }
45
46     // showing we need access to the time:
47     currentTime = millis();
48     prevTime = currentTime;
49     // For 1 second we show the randR,randG,randB color:
50     while (currentTime - prevTime < 1000) {
51         // We cycle through the R, G and B part, and show each color for
52         // a fixed delayDuration time
53         analogWrite(ledR, randR);
54         delay(delayDuration);
55         analogWrite(ledR, 0);
56         analogWrite(ledG, randG);
57         delay(delayDuration);
58         analogWrite(ledG, 0);
59         analogWrite(ledB, randB);
60         delay(delayDuration);
61         analogWrite(ledB, 0);
62         currentTime = millis();
63     }
64 }

```

De LED zou moeten mooi branden, en van kleur wisselen elke seconde. Als je speelt met de variabele `delayDuration`, zul je evenwel opmerken dat de LED gemakkelijk begint te flikkeren als je niet vlug kleur wisselt.

Om te testen hoe accuraat de kleuren zijn, doe het volgende:

1. wijzig de tijd van een enkele kleur in 60 seconden, dus 60000 milliseconden
2. verifieer dat je hebt in je sketch: `bool test=true;`
3. upload de sketch, een kleur zal nu een minuut duren
4. Open de seriële monitor, dit is het icoon met het vergrootglas in de rechterbovenhoek van de Arduino Editor. Een nieuw venster gaat open. Elke minuut zul je de kleur waarden zien verschijnen van de kleur die getoond wordt.
5. Om te zien dat dit plus-minus correct is kun je een kleurenkiezer pro-

gramma gebruiken. Bijvoorbeeld www.colorpicker.com heeft velden voor R, G and B die je kunt ingeven om te zien welke kleur ermee overeenkomt.

6. Verwacht niet teveel accuraatheid in de kleuren! Vooreerst, als je geen donkere kamer gebruikt zal de LED altijd zijn eigen kleur hebben. Zwarte kleuren zijn dan niet mogelijk, gezien donker gewoon afwezigheid van licht is. Verder worden rood, groen en blauw in verschillende regio's van de LED opgewekt waardoor het mixen niet perfect kan zijn. Dan komen ook het pure rood, groen en blauw niet precies overeen met die van het RGB kleurenmodel, dus is mixen wat anders. Als laatste, de LED maakt maar weinig licht. De LED bevat of een lenze om het licht te focussen in één richting, of een wazige rand om het licht diffuus te verspreiden. In beide gevallen krijg je artefacten bij het mengen.

Opdracht 5 *Draai de test rond. Selecteer een kleur via de kleuren kiezer. Wijzig de sketch zodat je de kleur krijgt die je gekozen hebt.*

3.3 Moeilijk: Kleuren mengen op de expliciete manier

Voor onze kubus zullen we 9 LEDs hebben met 3 kleuren. Als we PWM willen gebruiken, zouden we $9 \times 3 = 27$ analogen pinnen nodig hebben om de 9 LEDs elk een andere kleur te geven. Dit is niet mogelijk met de Arduino, er zijn niet zoveel pinnen. We zouden in plaats daarvan de kubus kunnen zien als een enkele lamp die altijd dezelfde kleur zal tonen. Dan hebben we maar 3 pinnen met PWM nodig.

Vervolgens, door een LED na de andere aan te schakelen, zouden we toch nog in staat zijn om elke LED een andere kleur te geven!

Het zou evenwel lang duren tussen branden voor een LED. Veronderstel dat we 0.5 milliseconden een kleur van een LED tonen via PWM. Een enkele LED heeft dan 1.5 ms nodig voor een kleur. Daarna pas kan de volgende LED. Dus zijn er 13 ms nodig voor de eerste LED opnieuw kan branden. In andere woorden, een kleur wordt 0.5 ms van een cyclus van 13.5 ms getoond, of $5/135 = 1/27$ van de tijd. Het zou beter zijn als we $1/3$ van de tijd kunnen gebruiken, gezien onze kubus dan 9 keer helderder zal zijn. Dat is de reden dat we kleuren zullen mengen op een moeili-



Figure 13: 4 frames om een bewegend paard te maken. De volledige animatie: [Wikipedia Horse_gif](#)

jke manier: zonder PWM.

Voor we hieraan beginnen, moeten we begrijpen hoe we animaties zullen tonen, zodat

we een robuuste basis hebben voor de effecten die je wilt tonen.

We beschouwen opnieuw onze RGB-LED. Welke effecten kunnen we tonen met één enkele RGB? Enkele voorbeelden:

- Cycle: rood, groen, blauw altijd sneller tot op het punt dat we wit zien.
- Ga van een kleur naar een andere kleur op een mooie gladde manier.
- Doe morse code van een zin
-

Dus, hoe zullen we dat programmeren? We zullen doen zoals in de filmen: een specifiek effect bestaat uit verschillende shots of acties. Een actie heeft een zekere duur: t_a . De actie verdelen we in frames F , welke een enkel beeld zijn, zoals een foto genomen op een specifiek moment, zie Figuur ???. Een actie is dus een functie, welke voor een specifieke tijd ons dient te vertellen welk frame te tonen: $F = \text{actie}(t)$

Om persistentie van beeld te krijgen, tonen we een frame 40 ms, waarna de volgende frame getoond wordt. Voor een enkele LED zal een frame een specifieke kleur zijn. Om deze kleur te tonen, dienen we te itereren over

de 3 basiskleuren: rood, groen, blauw. Bijgevolg zullen we een frame F onderverdelen in subframes S . Een subframe is een beeld die onze LED echt kan maken, dus een helderheid in rood, een helderheid in groen, of een helderheid in blauw. Dus, een frame is dan een aantal subframes die we in serie tonen:

$$F = S_B \circ S_G \circ S_R.$$

We hebben nog een laatste truck nodig. Als we een frame in 3 gelijke subframes zouden delen, dan duren die elk $40/3 \approx 13.3\text{ms}$. Dat is niet optimaal, gezien het vlug kan overkomen als geflikker, zeker als we maar een enkele basis kleur, bv rood, moeten tonen. Daarom zullen we een optimale tijd opnemen om een subframe te tonen. Dan, voor zo lang als een frame moet getoond worden, herhalen we de cyclus van subframes, we krijgen dus:

$$F = S_B \circ S_G \circ S_R \circ \dots \circ S_B \circ S_G \circ S_R \circ S_B \circ S_G \circ S_R.$$

Wat is de beste duur van een subframe? Voor een enkele LED zal een derde van een ms niet slecht zijn. We drukken dit uit in microseconden (μs). A μs is een duizendste van een milliseconde, terwijl een ms een duizendste van een seconde is.

Laat ons dit in de praktijk brengen met hetzelfde effect dat we eerder via PWM deden: toon een willekeurige kleur voor een sec-

onde. Gezien de LED niet evenveel kleuren kan tonen als het volledige RGB spectrum, limiteren we de 256 gradaties in $256/4=64$

gradaties. We drukken de scene uit in de Rood, Groen en Blauw component in 64 gradaties, en een duur om de kleur te tonen:

scene = (red base 64, green base 64, blue base 64, duration).

De actie van deze scene is het tonen van een enkele rood, groen, blauw kleur, dus elke frame is identiek:

$$F = (\text{rood}, \text{groen}, \text{blauw}).$$

Een subframe bestaat in het tonen van rood, groen, of blauw. Laat ons rood nemen.

We zullen een nummer van 0 tot 64 hebben voor rood. Laat ons dit nummer R noemen, bijvoorbeeld $R = 35$. We hebben reeds gezegd dat de duur van het subframe rond de $333\mu\text{s}$ moet zijn. Laat ons $64 \cdot 5 = 320$ nemen als duur van het subframe.

Als de waarde van rood 64 is, tonen we rood voor $320\mu\text{s}$, als de waarde 0 is tonen we rood $0\mu\text{s}$ in dit subframe. Dus, voor $R = 35$, tonen we het $35 \cdot 5 = 175\mu\text{s}$.

The code om dit te implementeren is in Code ??.

Code 7

```

1  /*
2  Controlling an RGB led via an npn switch and digital switching
3  using a frame abstraction
4  */
5
6  // pins used
7  int ledR=11;
8  int ledG=10;
9  int ledB=9;
10 int led1=2;
11
12 bool test = false; //use serial monitor for testing.
13
14 void setup() {
15     if (test) {
16         Serial.begin(9600);
17     }
18     randomSeed(analogRead(0));
19     // define the pins we use for the RGB led
20     pinMode(ledR, OUTPUT); pinMode(ledG, OUTPUT);
21     pinMode(ledB, OUTPUT); pinMode(led1, OUTPUT);
22 }
23
24 /*****
25 We consider the effect (a random color for 1 second) to be a shot or action.
26 A shot consists of frames which are shown 40ms. We determe in the loop if
27 a new frame must be drawn.
28
29 General outline animation architecture
30
31 1. A movie is a sequence of shots.
32 2. A shot has a duration expressed in ms. The action will be split up in
33 frames to give the illusion of smooth change.
34 2. A frame is a fixed pattern that repeats for 1/25 of a second. The human
35 eye can only see 24 images per second, so one frame every 1/25 seconds
36 can give the illusion of smooth movement/change
37 Frames themself are drawn from subframes
38 3. To create a frame, multiplexing will be needed. This is the act of

```

```

39 combining individual light outputs to create a single frame.
40 Eg, an RGB led cannot show R, G and B at the same time, so these must be
41 shown in serie. For the same reason, a led cube might not have to power
42 output to drive a complete cube in one subframe. So a subframe is one
43 output of a loop in arduino. A cycle of subframes creates a frame.
44 The cycle is repeated for the duration of the frame.
45 *****/
46 unsigned long movietime = 0UL;
47 //our first shot: a random color for a specific duration
48 long random_colorR = 0;
49 long random_colorG = 0;
50 long random_colorB = 0; // global variables
51
52 void random_color(unsigned long framenr, int frame[3]){
53 //update the frame for the time shown
54 if (framenr == 0) {
55 //determine the random color we will use
56 random_colorR = int(random(65));
57 random_colorG = int(random(65));
58 random_colorB = int(random(65));
59 if (test) {
60 Serial.print("New random color:");
61 Serial.print(random_colorR * 4);Serial.print(" ");
62 Serial.print(random_colorG * 4);Serial.print(" ");
63 Serial.print(random_colorB * 4);Serial.print(" at time ");
64 Serial.println(millis()/1000.);
65 }
66 }
67 //store the random color in the frame
68 frame[0] = random_colorR;
69 frame[1] = random_colorG;
70 frame[2] = random_colorB;
71 //to test: hard code colors here:
72 //frame[0] = 0; frame[1] = 0; frame[2] = 64;
73 }
74
75 void (*movie(unsigned long *shotduration))(unsigned long, int[3]){
76 // when a shot is finished, movie() is called to obtain the next shot.
77 // Here we have a single random color, and a fixed duration of 1000 ms.
78 *shotduration = 1000;
79 return random_color;
80 }
81
82
83 /*****
84 THAT WAS IT! SHORT NO?
85
86 THE REST IS THE GENERAL FRAMEWORK TO SHOW FRAMES ACCORDING TO
87 WHAT YOU WANTED
88
89 *****/
90 //global variables needed
91 unsigned long startTime = 0UL;
92 unsigned long currentTime;
93 unsigned long shotduration = 0UL;
94 //shotptr curshot;
95 void (*curshot)(long unsigned int, int*);

```

```

96  int          curframe[3];
97  unsigned long  framenr = 0UL;
98  unsigned long  curframenr = 1UL;
99  boolean        newframe = true;
100 unsigned long  starttimeframe = 0UL;
101 int            subframecolor;
102 unsigned long  subframecycle, cursubframecycle;
103 unsigned long  subframestarttime;
104 unsigned long  curmicrotime;
105
106 void loop(){
107     //first we determine if a new shot must start or not
108     currentTime = millis();
109     if (currentTime - startTime >= shotduration) {
110         //a new shot
111         curshot = movie(&shotduration);
112         startTime = millis();
113         currentTime = startTime;
114         curframenr = 1000UL; //dummy value
115     }
116     //we obtain the framenummer we should show, one frame=40ms
117     unsigned long framenr = (currentTime - startTime)/40UL;
118     if (curframenr != framenr){
119         //a new frame to show, obtain it
120         curshot(framenr, curframe);
121         newframe = true;
122         starttimeframe = micros();
123         curframenr = framenr;
124     } else {
125         newframe = false;
126     }
127     // we continue showing a cycle of subframes as needed for the current frame
128     curmicrotime = micros();
129     if (newframe) {
130         //reinit the subframes
131         subframecolor = 0;
132         subframestarttime = curmicrotime;
133         subframecycle = 0UL;;
134         cursubframecycle = 0UL;
135     } else {
136         // determine if a new subframecolor is needed
137         cursubframecycle = (curmicrotime - starttimeframe) / 960UL; //960=3*5*64
138         if (cursubframecycle != subframecycle){
139             //new subframecycle
140             subframecycle = cursubframecycle;
141             subframestarttime = curmicrotime;
142             subframecolor = 0;
143         } else {
144             subframecolor = subframecolor + 1;
145             subframecolor = subframecolor % 3;
146         }
147     }
148     //we know the frame, the color, and how far in the subframe we are
149     show_subframe_color(subframecolor, curmicrotime - subframestarttime);
150 }
151
152 void show_subframe_color(int color, long microtime){

```

```

153 // all off
154 digitalWrite(led1, HIGH);
155 digitalWrite(ledR, LOW);
156 digitalWrite(ledG, LOW);
157 digitalWrite(ledB, LOW);
158 if (color == 0) {
159     if (microtime < 15 * curframe[0]) {
160         digitalWrite(ledR, HIGH);
161     }
162 } else if (color == 1) {
163     if (microtime < 15 * curframe[1]) {
164         digitalWrite(ledG, HIGH);
165     }
166 } else {
167     if (microtime < 15 * curframe[2]) {
168         digitalWrite(ledB, HIGH);
169     }
170 }
171 }

```

De code ziet er complex uit, maar is eigenlijk verdeelt in twee delen: shots of acties die jij maakt, en een algemeen deel welke beslist wanneer een nieuw frame nodig is, dit frame opvraagt de shot, en dan subframe cyclussen start om dit specifieke frame 40ms te tonen. Dit laatste deel zal niet meer wijzigen, als je wil kun je proberen het te verstaan, als niet, geen probleem, het volstaat in staat te zijn je eigen shots te schrijven.

Laat ons over de code gaan. Zoals steeds beginnen we met de pinnen aanduiden die we zullen gebruiken. Dan komt de `setup()` functie, welke de Arduino klaarmaakt. Niets nieuws dus.

We bekijken hier hoe een shot gemaakt wordt. Hiervoor hebben we een functie nodig: bv. een willekeurige kleur. Deze functie heeft een specifiek functievoorschrift die al onze shots zullen moeten hebben:

`void random_color(unsigned long frameNr, int frame[3]).`

Dus, een shot geeft geen waarde terug. Dit duiden we aan met `void`. Het heeft wel twee parameters nodig: het framenummer van het shot, en een lijst van 3 gehele getallen: `int`

`frame[3]`. De functie dient deze lijst op te vullen met het frame dat getoond moet worden. Dat is het! Gezien onze kleuren lopen van 0 tot 64, gebruiken we `random(65)`.

De volgende functie beschrijft onze film (*movie* in het Engels). De functie definitie ziet er heel complex uit. In werkelijkheid duidt deze gewoon aan dat de `movie` functie een shot functie teruggeeft als waarde, zoals bevoorbeeld onze `random_color` functie van hierboven. Daarnaast krijgt de functie een variabele mee, `shotduration`, welke moet opgevuld worden met het aantal milliseconden van de duurtijd van je shot. De code zal dan telkens als een shot gedaan is, deze functie `movie` opnieuw oproepen om een volgend shot te bekomen en te tonen.

Onder deze twee functies komt dan de interne Arduino `loop` om een shot te tonen. Deze loop zal dus `movie` oproepen om nieuwe shots te bekomen, en zal dan de shot oproepen om een frame nummer te bekomen. Het zal er dan voor zorgen dat deze frame getoond wordt voor de duur van een frame (40ms).

Opdracht 6 Voorgeprogrammeerde kleuren: een vaste kleur tonen. Om te testen dat je alle kleuren kunt tonen, wijzig Code 7 zodat je een vaste kleur toont i.p.v. een random color, bv. altijd rood.

Om dit te doen moet je de functie `random_color` aanpassen zodat je een functie `fixed_color` hebt. Test het dan met bv. enkel rood, enkel groen, of enkel blauw.

Opdracht 7 Twee shots tonen. Pas nu Code 7 aan zodat je 1 seconde een vaste kleur toont en 2 sec een random kleur.

3.4 Gladde overhangen

We breiden nu de vorige code uit. We maken een nieuw shot en we zien zo hoe we shots kunnen toevoegen. Dit shot is een gladde overgang van een eerste kleur naar een andere kleur.

De code voor dit shot, en de `movie` functie die ze roept kun je vinden in Code ???. Dit vervangt het shot en movie deel van de vorige sectie.

Enkele opmerkingen. We verdelen de volledige duur in frames, en berekenen welke we nodig hebben. Als we een transitie doen blenden we de twee kleuren: dat is een nummer tussen 0 en 1, noem het b . We mixen dan begin en eindkleur als:

$$(1 - b) \times \text{beginkleur} + b \times \text{eindkleur}.$$

Als we dus starten met $b = 0$, hebben we de beginkleur, en voor $b = 1$ de eindkleur.

Als je twee gehele getallen deelt, is het resultaat opnieuw een geheel getal. Daarom gebruiken we de `float` functie om een geheel getal (integer) in een reëel getal (float) te veranderen. Zo vinden we de blend tussen 0 en 1 voor een specifieke frame. Maar, ons frame bestaat wel uit kleuren die we aanduiden met een getal tussen 0 en 64, daarom passen we de functie `round` toe om de float te veranderen in de dichtstbijgelegen integer. Bijvoorbeeld: `round(1.3)=1` en `round(1.6)=2`.

Code 8

```
47 long random_colorR = 0; // global variables
48 long random_colorG = 0;
49 long random_colorB = 0;
50
51 void fixed_color(unsigned long framenr, int frame[3]){
52     //shot: show a fixed color stored in global variables:
53     frame[0] = random_colorR;
54     frame[1] = random_colorG;
55     frame[2] = random_colorB;
56 }
57
58 void random_color(unsigned long framenr, int frame[3]){
59     //shot: a random color for a specific duration
```

```

60  if (framenr == 0) {
61      //determine the random color we will use
62      random_colorR = int(random(65));
63      random_colorG = int(random(65));
64      random_colorB = int(random(65));
65  }
66  //store the random color in the frame
67  frame[0] = random_colorR;
68  frame[1] = random_colorG;
69  frame[2] = random_colorB;
70  }
71
72  long smooth_colorR, smooth_colorG, smooth_colorB; // global variables
73  unsigned int smooth_color_transition_duration = 10000;
74  unsigned int fixed_color_duration = 5000;
75
76  void smooth_color(unsigned long framenr, int frame[3]){
77      //shot: shows a color, then goes smooth to a new color
78      unsigned int last_fixed_frame = fixed_color_duration/40;
79      unsigned int last_trans_frame = (smooth_color_transition_duration +
80          fixed_color_duration) / 40;
81      unsigned int smooth_frame_length = last_trans_frame - last_fixed_frame;
82      if (framenr < last_fixed_frame) {
83          //show the fixed color
84          fixed_color(framenr, frame);
85      } else if (framenr == last_fixed_frame){
86          // finished first color, we determine what will be our next color
87          smooth_colorR = int(random(65));
88          smooth_colorG = int(random(65));
89          smooth_colorB = int(random(65));
90          fixed_color(framenr, frame);
91      } else if(framenr < last_trans_frame) {
92          // we compute how much to mix both colors
93          float blend = (framenr - last_fixed_frame) / float(smooth_frame_length);
94          frame[0] = round((1-blend) * random_colorR + blend * smooth_colorR);
95          frame[1] = round((1-blend) * random_colorG + blend * smooth_colorG);
96          frame[2] = round((1-blend) * random_colorB + blend * smooth_colorB);
97      } else {
98          //we have (framenr >= last_trans_frame)
99          //so finished, we end with last color showing it fixed
100         random_colorR = smooth_colorR;
101         random_colorG = smooth_colorG;
102         random_colorB = smooth_colorB;
103         fixed_color(framenr, frame);
104     }
105 }
106
107 void (*movie(unsigned long *shotduration))(unsigned long, int[3]){
108     // when a shot is finished, movie() is called to obtain the next shot.
109     // a color, smooth transition, and another color, so duration:
110     *shotduration = smooth_color_transition_duration + 2 * fixed_color_duration;
111     return smooth_color;
112 }

```

Opdracht 8 Joww shot. Maak je eigen shot. Bevoorbeeld, knipper de LED altijd maar sneller en sneller.

Tip: Bevoorbeeld, als je shot 60 seconden is, en je start van een 2 seconden interval, en je wil eindigen met een 0.001 seconden interval, dan moeten je intervallen lineair afnemen:

$$y = 2 + \frac{0.001 - 2}{60} \text{tijd}$$

Mooier: mengen, dit is twee kleuren afwisselend knipperen, en dit versnellen. Je zou de kleuren moeten zien vermengen. Nog mooier: rood, groen en blauw hoeveelheden mengen

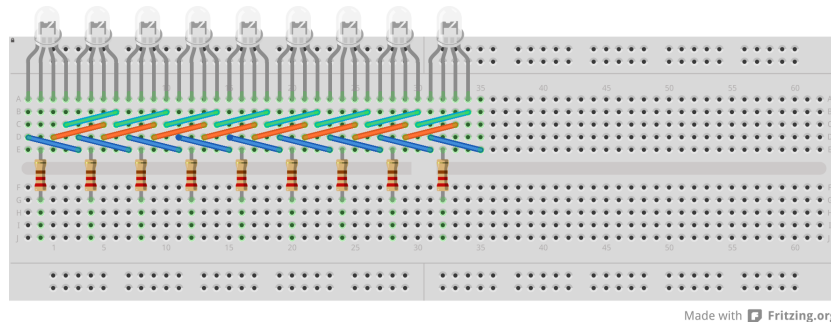


Figure 14: Plaats de 9 LED op het bord. Connecteer alle gelijke kleuren met elkaar. De gemeenschappelijke kathode of anode gaat met een $220\ \Omega$ weerstand naar een nieuwe lijn.

4 Les 3: LED Cube constructie

4.1 Opstelling schakelbord

We hebben alles gezien om aan onze Fe Kubus te beginnen. We starten met alle componenten op het schakelbord te zetten. Dan testen we de schakeling met een programma die onze vorige RGB LED schets aanpast voor de kubus. Alleen als we zeker zijn dat alles werkt zullen we de effectieve constructie doen.

Plaats dus de volgende componenten voor jou:

1. Jouw schakelbord
2. 9 RGB-LED (we veronderstellen gemeenschappelijke kathode in wat volgt)
3. 3 NPN (2N3904 normaal)
4. 9 $220\ \Omega$ weerstanden
5. veel draden.

Begin met de RGB-LED op je bord te zetten, verbindt de kleurpinnen met elkaar via draden, en connecteer de gemeenschappelijke kathode via een weerstand naar een andere lijn op je bord, zie Fig. ??.

Verbind nu de weerstanden met pinnen 1 tot 9. Deze pinnen zullen de LED van stroom voorzien. Dus, voor een gemeenschappelijke kathode zullen we deze pinnen op **HIGH** zetten om de LED te doen branden. Plaats dan de 3 NPN op het bord. De draden van een kleur gaan naar de Emitter zijde (-) van de NPN voor gemeenschappelijke kathode LEDs. De Collector zijde (+) gaat naar de Voltage lijn op je schakelbord. Deze Voltage lijn gaat dan naar een 5V pin op de Arduino.

Het Basis been van de NPN zal bepalen of de NPN junctie open is of niet. We zullen pinnen 10, 11, 12 gebruiken voor dit been. Je volledige schakeling zou er moeten uitzien als Fig. ??.

4.2 Schakelbord Fe Cube uittesten

We zullen enkel controleren dat we deze schakelbord variant van de Fe kubus kunnen doen werken. Hiertoe passen we onze RGB schets aan. We zullen een film maken die

eerst 4 seconden rood toont, dan 4s groen, dan 4s blauw, dan 1 minuut een willekeurige kleur, en uiteindelijk gladde overgangen voor een minuut.

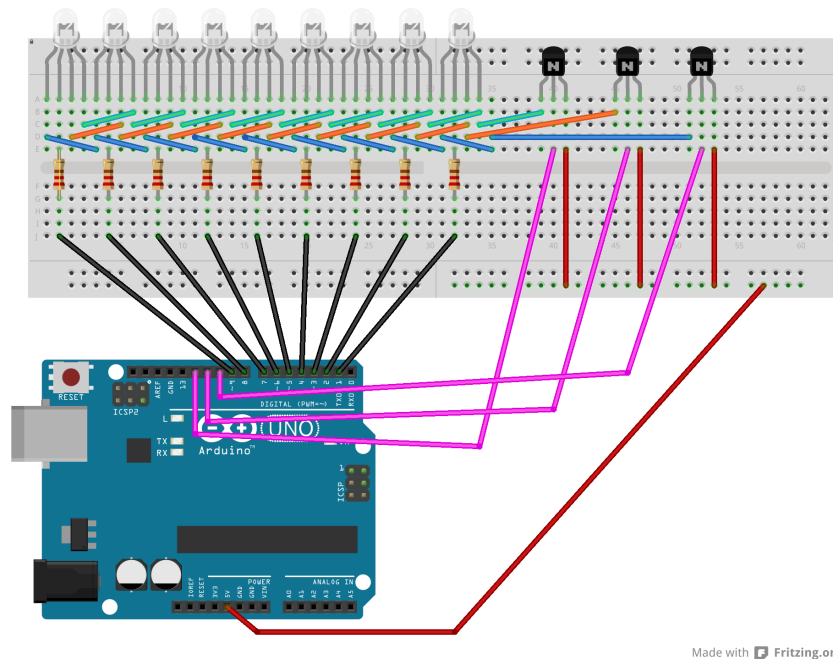


Figure 15: Verbind de weerstanden met pinnen 1 tot 9. De kleuren gaan via een npn naar de GND (kathode) of 5V (anode). Pinnen 10, 11, 12 gaan naar de NPN om de kleuren te schakelen.

Dit betekent dat we al de shots die we gemaakt hebben voor de RGB LED in de vorige lessen zullen gebruiken en aanpassen. De movie functie zullen we zo maken dat al deze shots elkaar opvolgen. In plaats van één RGB led hebben we er nu 9, dus dienen we deze 9 nu samen aan of af schakelen.

In de vorige schetsen hadden we ledR, ledG en ledB, welke gemakkelijk te verstaan zijn. De LED noemden we led1. We zouden nu led1 tot led9 kunnen gebruiken, maar dat is niet handig als we later lichteffecten maken. Laat ons ze dus betere namen geven. We hebben een top (T) en een bodem (B) laag, en een enkele LED in het midden. Dan kunnen we de kubus opdelen in een links (L) en een rechts (R), en in een frontaal (F) en een achter (A). Daarom noemen we de LEDs:

1. ledTLF
2. ledTLA
3. ledTRF
4. ledTRA
5. ledBLF
6. ledBLA
7. ledBRF
8. ledBRA
9. ledMID

Open Code ??, en sla het op onder een andere naam. De eerste wijziging is alle pinnen definiëren:

```

5 // pins used
6 int ledR=10; int ledG=11; int ledB=12;
7 // led anodes are connected to pins via a resistor
8 // Bottom/Top - Left/Right - Aft/Front
9 int ledBLA=1; int ledTLA=2; int ledBLF=3; int ledTLF=4;
10 int ledMID=5; int ledTRF=6; int ledBRA=7; int ledTRA=8;

```

```
11 int ledBRF=9;
```

Vervolgens dienen we in `setup` deze pin- nen als digitale output pinnen te initialiseren:

```
15 void setup() {
16     if (test) {
17         Serial.begin(9600);
18     }
19     randomSeed(analogRead(0));
20     pinMode(ledR, OUTPUT); pinMode(ledG, OUTPUT); pinMode(ledB, OUTPUT);
21     pinMode(ledBLA, OUTPUT); pinMode(ledBLF, OUTPUT); pinMode(ledBRA, OUTPUT);
22     pinMode(ledBRF, OUTPUT); pinMode(ledMID, OUTPUT); pinMode(ledTLA, OUTPUT);
23     pinMode(ledTLF, OUTPUT); pinMode(ledTRA, OUTPUT); pinMode(ledTRF, OUTPUT);
24 }
```

We testen de kubus als een enkele vervangen door een functie die alle LED
LED, dus alle LED zijn aan, of allemaal aan zet, welke we plaatsen in de functie
af. Bijgevolg zullen we led1 aanzetten `show_subframe_color`

```
217 digitalWrite(ledR, LOW);
218 digitalWrite(ledG, LOW);
219 digitalWrite(ledB, LOW);
220 all_led_on();
221 if (color == 0) {
```

Deze nieuwe functie om de pinnen **aan** gezien ze verbonden zijn aan de LED kathode
te zetten stuurt `LOW` signalen naar de pinnen, (-)

```
245 digitalWrite(ledBLA, LOW); digitalWrite(ledBLF, LOW);
246 digitalWrite(ledBRA, LOW); digitalWrite(ledBRF, LOW);
247 digitalWrite(ledMID, LOW); digitalWrite(ledTLA, LOW);
248 digitalWrite(ledTLF, LOW); digitalWrite(ledTRA, LOW);
249 digitalWrite(ledTRF, LOW);
250 }
```

Als laatste dienen we de movie functie aan Deze functie dient te zijn
te passen zodat het zou doen wat we willen.

```
109 int maxshot = 78; // number of shots in our movie
110 int shotnr = 0; // empty start
111 void (*movie(unsigned long *shotduration))(unsigned long, int[3]){
112     // when a shot is finished, movie() is called to obtain the next shot.
113     shotnr += 1; //the shot we want to show next
114     if (shotnr > maxshot) { shotnr = 1; } //if over the maximum, we start again
115     unsigned long curmovietime = millis();
116     switch (shotnr){
117         case 1:
118             //4 seconds red
119             *shotduration = 4000;
120             random_colorR = 64; random_colorG = 0; random_colorB = 0;
121             return fixed_color;
122             break;
```

```

123     case 2:
124         //4 seconds green
125         *shotduration = 4000;
126         random_colorR = 0; random_colorG = 64; random_colorB = 0;
127         return fixed_color;
128         break;
129     case 3:
130         //4 seconds blue
131         *shotduration = 4000;
132         random_colorR = 0; random_colorG = 0; random_colorB = 64;
133         return fixed_color;
134         break;
135 }
136 if (shotnr <= 63) {
137     //60 shots of a random color for 1 second
138     *shotduration = 1000;
139     return random_color;
140 }
141 //all remaining shots (up to 78) smooth transitions each 4 sec
142 smooth_color_transition_duration = 3000;
143 fixed_color_duration = 500;
144 // a color, smooth transition, and another color, so duration:
145 *shotduration = smooth_color_transition_duration + 2 * fixed_color_duration;
146 return smooth_color;
147 }

```

In de functie `movie` zie je een `switch` structuur. Dit is een programmeerstructuur die werkt als een test van een variabele. De syntax is:

NIET BESCHIKBAAR

In deze switch structuur worden de commando's voor dewelke de variabele gelijk is aan een label uitgevoerd vanaf dat label, en dit tot op het `break;` commando. Het is mooier om te lezen in code dan een grote `if` structuur om te testen op een variabele, maar werkt anders precies als een `if` structuur.

Opdracht 9 Één LED. Om te testen dat een LED alleen inderdaad niet helderder brandt, wijzig de `all_led_on` functie zodat maar één LED aan is.

4.3 Constructie

De schakeling werkt, tijd om de kubus te construeren. Om dit te doen zullen we de componenten van ons schakelbord een voor een nemen en solderen in de vorm van een Fe Cubus.

We noemen onze kubus een Fe Cubus omdat het georganiseerd is zoals het Ijzer (Fe)

metaal kristalrooster. Ijzermetaal heeft een specifieke structuur welke uitgebeeld wordt in het [Atomium gebouw](#) in Brussel, zie Figuur ??.

We noemen deze structuur een [Lichaamsgecentreerd kubisch kristalrooster](#), in het kort **bcc**.

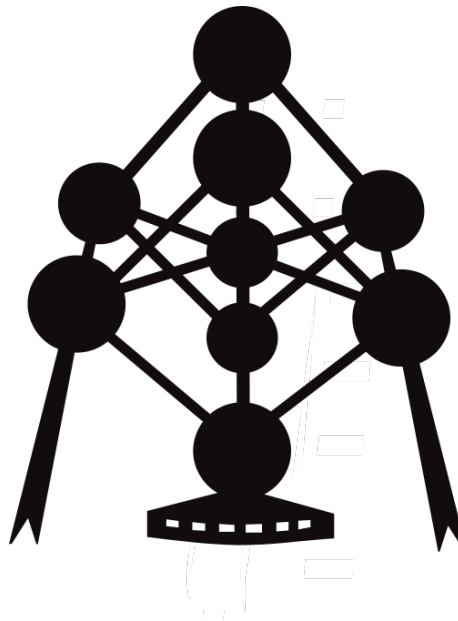


Figure 16: Wegmarkering in Brussel voor het atomium gebouw.

5 Les 4: De LED Cube gebruiken

5.1 De Fe Cube testen

De constructie is af. We dienen nu te testen of alle lichten correct werken. Start met dezelfde schets die je gebruikte voor de schakelbord kubus. Als je alles correct verbindt zou het moeten werken!

Als er dingen niet werken heb je extra solderwerk voor de boeg: vervang een gesprongen LED, maak verbindingen sterker, Nu niet opgeven! Dit zijn de laatste constructieproblemen voor je LED kubus.

5.2 Algemene patronen ... en een slang

We hebben een werkende LED kubus, maar we kunnen enkel maar vaste kleuren tonen met de huidige code. Laten we dat veranderen. We willen lichteffecten kunnen programmeren. Laat ons dus een `movie` functie maken die patronen kan laden. Op deze manier kunnen we een lichtpatroon opschrijven en dat door deze functie laten laden op de kubus. Het eerste patroon die we willen maken is een slangenparoon: een enkel lichtje dat springt van LED naar LED in de 3 basiskleuren.

We moeten een data structuur maken voor het patroon. Waaruit moet een patroon bestaan?

1. We dienen voor de 9 LED te weten welke kleur ze moeten hebben. Een kleur is een RGB waarde zoals (13,50,2), ervanuit gaande dat we nog steeds waarden tussen 0 en 64 gebruiken. We hebben nu dus $3 \times 9 = 27$ nummers nodig om een frame van een shot te maken.

2. We moeten dan ook nog weten hoe lang we een enkel frame moeten aanhouden vooraleer naar het volgende frame in het patroon te schakelen. We hebben dus nog een extra nummer nodig voor het aantal milliseconden dat een frame moet getoond worden, voor een totaal van 28 nummers.

Een patroon zal dus lijken op een lijst van nummers welke aanduiden welk frame te tonen en voor hoe lang. Voor een enkele frame zal dit zijn:

```
{R1,G1,B1, R2,G2,B2, R3,G3,B3, R4,G4,B4,
R5,G5,B5, R6,G6,B6, R7,G7,B7, R8,G8,B8,
R9,G9,B9, duurtijd}
```

De orde van de LEDs in het frame zal ledTLF, ledTLA, ledTRF, ledTRA, ledBLF, ledBLA, ledBRF, ledBRA, ledMID zijn. Als een shot bestaat uit een patroon met 20 vaste frames, dan zullen we een enkele grote lijst maken met alle frames, de ene na de andere. Gezien Arduino code niet zo slim is moeten we op een of andere manier doorgeven welke het laatste frame is. We zouden dit zelf kunnen tellen en doorgeven, maar dat leidt vlug tot fouten. Gemakkelijker is om te eindigen

met een vals patroon met duurtijd 0. Als we deze 0 lezen weten we dat het patroon gedaan is.

In onze code zal de `movie` functie nu een `moviepattern` functie oproepen. Deze zal het patroon afspelen die je doorgeeft. De functie zal het patroon intern verdelen in een shot per gegeven frame. Dit shot zal dan zo lang duren als de duurtijd die je opgegeven hebt in het patroon. Dit shot is de tegenhanger van de functie `fixed_color` bij de enkele RGB LED: `fixed_pattern`, welke een shot is dat een enkel vast beeld op de kubus toont.

Er is evenwel nog een complicatie. De Arduino heeft niet veel geheugen om mee te werken. Er is SRAM geheugen welke de Arduino mee werkt, en Flash geheugen waar het programma opgeslagen wordt. Alle globale variabelen moeten altijd beschikbaar zijn en bevinden zich dus in SRAM. Onze patronen kunnen evenwel groot zijn, en het geheugen zal dat vol geraken, waarna de Arduino stopt met werken. Om dit te vermijden zullen we de patronen opslaan in flash geheugen, en enkel het frame dat we nodig hebben laden in SRAM geheugen. Daarom zul je het volgende terugvinden in de code:

NIET BESCHIKBAAR

Dit definieert de functies die we nodig hebben om data te verplaatsen van flash naar SRAM. Het `PROGMEM` gedeelte vervangt een gewone lijst (`array`) constructie en maakt voor ons een lijst van integers (gehele getallen, ons patroon dus). Het duidt aan dat we een lijst integers willen, maar dat we die in `PROGMEM` willen, dus in flash.

In de functie `moviepattern` zie je twee keer een `switch` structuur.

In de eerste switch structuur dien je de startcondities van je patroon te schrijven. In de tweede switch structuur dien je je patroon te kopiëren in de `shotpattern` variabele, alsook de `nextduration` variabele opvullen met de duur van het *volgende* shot. De functie `pgm_read_word_near` is de functie die zal

copiëren van programma geheugen (flash) naar SRAM. Het enige dat je hier normaal moet wijzigen is de naam van je patroon, of het toevoegen van een switch case voor jouw patroon.

De startcondities van een patroon zijn:

1. **`patternscale_start`**: met welke tijdschaling wil je beginnen. deze schaal wordt vermenigvuldigd met de duur van een frame.
2. **`patternscale_speedup`**: een waarde tussen 0 en 1. Elke keer een patroon gedaan is kun je het hiermee vlugger doen gaan. Deze speedup wordt ook vermenigvuldigd met de duur van een frame.

3. **patternscale_min**: je kan niet altijd maar sneller gaan, je moet ook eens stoppen. Deze parameter bepaald de minimum tijdsschaling, na dewelke het patroon volledig kan stoppen.
4. **patternrepeatmin**: gezien de snelste weergave met patternscale_min vaak zo vlug gaat dat je het niet kunt zien, kun je hier opgeven hoe vaak deze snelste weergave moet herhaald worden. Daarna zullen we het patroon defenitief als gedaan beschouwen, en wordt NRPATTERN met 1 verhoogd. NRPATTERN is de variabele die in de switch structuren gebruikt wordt. Het ver-

hogen met een zal dus zorgen dat een volgend patroon uitgevoerd wordt.

5. **totalpatterns**: gebruik deze variabele om aan te duiden hoeveel patronen er in totaal zijn.

Om te tonen wat de invloed van deze parameters is bevat de code een extra patroon: PatternCircle. Deze heeft andere startwaarden en gedraagt zich dus anders.

De rest van de extra code is dan om de correcte versnelling van de shots in een patroon te berekenen, en om te bepalen of een patroon gedaan is en een nieuwe kan beginnen.

De vollige code kun je vinden op https://github.com/ingegno/fecube/blob/master/sketches/old_versions/v01/Fe_cube_03_cube_pattern/Fe_cube_03_cube_pattern.ino

Opdracht 10 Jouw patroon. *Tijd om je eigen patroon te maken. Wat denk je van een zwaailicht? Of kerstboom flikkering? Of ...*

5.3 Intelligente patronen ... en een kloppend hart

We kunnen nu patronen maken, maar dat is wel veel werk. Dit is niet verschillend van animatiefilms met speciale effecten. Veel tijd is nodig om die te maken. Om dit te versnellen zullen ontwikkelaars zoveel mogelijk programmeren, om zo manueel werk te besparen.

Bevoorbeeld, als we ons slangenparoon bekijken zien we in essentie hetzelfde patroon 3 keer herhaald, maar wel voor een andere kleur. Het zou makkelijker zijn het patroon één keer schrijven, en dan met een programma het op een andere kleur toe te passen. Als een ander voorbeeld, stel dat je een kloppend hart patroon wil maken. Dan ga je van geen licht naar alle LED die rood worden. Zodra alles rood gaan we opnieuw naar geen licht, we moeten dus het patroon

omdraaien. Als we dit omdraaien gewoon kunnen programmeren dan kunnen we vermijden een lang patroon te schrijven!

Vooraleer nieuwe code te schrijven is het belangrijk eerst over het ontwerp na te denken. We eindigen nu een patroon met een valse lijn met duur 0. Gezien een duur kleiner dan 0 geen zin heeft, kunnen we een negatieve duur gebruiken om een effect aan te duiden om het patroon voort te zetten. Bevoorbeeld, het patroon herhalen met een andere kleur. We zullen dan wel moeten bijhouden hoeveel keer we het patroon moeten uitbreiden met een effect, zodat we zouden weten wanneer het echte einde bereikt is.

Laat ons volgende defenities invoeren: met de variabele `extend_pattern` duiden we aan hoeveel keer we het patroon uitbreiden.

$$: R \rightarrow B, G \rightarrow R, B \rightarrow G$$

-90 : roteer het shot over -90 de-
grees. Ideaal om roterende animaties
te maken.

-180 : flip het shot, dus TLF wordt TRA

-270 : roteer het shot over 90 graden

Al deze wijzigingen zullen de hoeveelheid code doen toenemen. Onze patronen kunnen evenwel kleiner zijn, waardoor we minder geheugen moeten gebruiken.

onze slang van eerder, en een kloppend hart.

```

18 #define ledTRF 8
19 #define ledBRA 7
20 #define ledTRA 6
21 #define ledBRF 9
22
23 const int ledorder[] = {ledTLF,    ledTLA,    ledTRF,    ledTRA,    ledBLF,    ledBLA,
                        ledBRF,    ledBRA,    ledMID};
24 const int colorder[] = {ledR, ledG, ledB};
25
26 const PROGMEM prog_int16_t PatternSnakeRGB[] = {
27 //order led:
28 //ledTLF,   ledTLA,   ledTRF,   ledTRA,   ledBLF,   ledBLA,   ledBRF,   ledBRA,
                                ledMID, duration
29 //snake red
30 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
31 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
32 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
33 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
34 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                64, 0, 0, 1000,
35 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
36 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
37 0, 0, 0, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
38 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 1000,
39 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                64, 0, 0, 1000,
40 // dummy to end the pattern, with duration the effect for the next repeat, see
    effect

```


[Fe_cube_03_cube.intpattern/Fe_cube_03_](#)
[cube.intpattern.ino](#). Enkel de start van
de moviepattern functie met de twee **switch**

structuren zou je moeten wijzigen om je eigen
patronen te gebruiken. Zie Code ?? voor deze
start.

Code 9

```
211 void (*moviepattern(unsigned long *shotduration))(unsigned long, int[27]){
212     // we obtain the current pattern:
213     if (PATTERNFINISHED) {
214         if (NRPATTERN > 5) {
215             NRPATTERN = 0;
216         }
217         switch (NRPATTERN) {
218             case 0:
219                 //first call, we load the snake pattern
220                 // red, green, blue
221                 patternscale_start = 1.;
222                 patternscale_speedup = 0.75;
223                 patternscale_min = 0.002;
224                 patternrepeatmin = 250;
225                 extend_pattern = 2;
226                 starteffect = 0;
227                 break;
228             case 1:
229                 // blue and red
230                 starteffect = -12;
231                 extend_pattern = 1;
232                 break;
233             case 2:
234                 // green and blue
235                 starteffect = -11;
236                 break;
237             case 3:
238                 // green back - forth
239                 starteffect = -11;
240                 extend_pattern = 1;
241                 break;
242             case 4:
243                 // beating heart
244                 patternscale_start = 0.15;
245                 starteffect = 0;
246                 extend_pattern = 1;
247                 patternscale_speedup = 0.9;
248                 patternscale_min = 0.01;
249                 patternrepeatmin = 200;
250                 break;
251             case 5:
252                 patternscale_start = 1.;
253                 starteffect = 0;
254                 extend_pattern = 3;
255                 patternscale_speedup = 1.;
256                 patternscale_min = 1.;
257                 patternrepeatmin = 50;
258                 break;
259         }
260     }
```

```

261     //reset start in case it changed
262     effect = starteffect;
263     patternscale = patternscale_start;
264     PATTERNFINISHED = false;
265 }
266 int nextduration = 0;
267 for (int ind=0; ind < 28; ind++){
268     switch (NRPATTERN) {
269         case 0:
270         case 1:
271         case 2:
272         shotpattern[ind] = pgm_read_word_near(PatternSnakeRGB +28*curpattern +
273                                             ind);
274         nextduration = pgm_read_word_near(PatternSnakeRGB +28*(curpattern + 2) -
275                                             1);
276         break;
277     case 3:
278         shotpattern[ind] = pgm_read_word_near(PatternSnakeRGB +28*curpattern +
279                                             ind);
280         nextduration = pgm_read_word_near(PatternSnakeRGB +28*(curpattern + 2) -
281                                             1);
282         // we override the effect as we want reverse!
283         if (nextduration <= 0) {
284             nextduration = -1;
285         }
286         break;
287     case 4:
288         shotpattern[ind] = pgm_read_word_near(PatternHeart +28*curpattern + ind);
289         nextduration = pgm_read_word_near(PatternHeart +28*(curpattern + 2) - 1);
290         break;
291     case 5:
292         shotpattern[ind] = pgm_read_word_near(PatternSiren +28*curpattern + ind);
293         nextduration = pgm_read_word_near(PatternSiren +28*(curpattern + 2) - 1);
294         break;
295     }
296 }
297 apply_shot_effect();

```

Opdracht 11 Meer patronen. De effecten werken op je Fe Kubus? Voeg een nieuw effect toe, bijvoorbeeld een kleurverandering. Of schrijf gewoon een nieuw patroon dat van enkele effecten gebruik maakt.

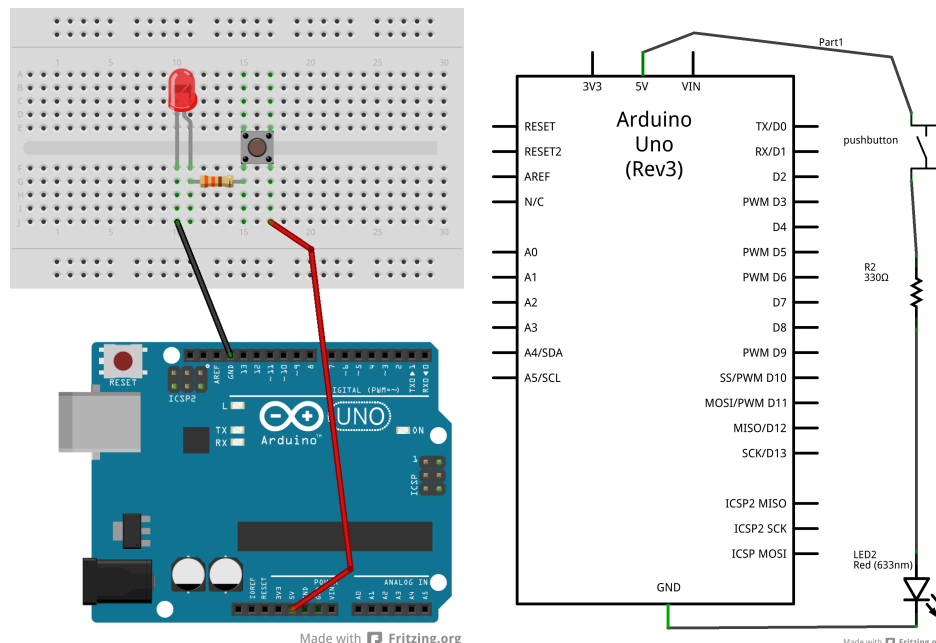


Figure 17: Stroom, een drukknop en een LED, en je can de LED sturen. Geen Arduino code nodig!

6 Les 5: Interactie met een drukknop

Je kan prachtige dingen doen met je kubus. Maar we kunnen het nog wat gebruiksvriendelijker maken. Ware het niet super als je kon interageren met je kubus?

Bevoorbeeld van effect veranderen als je een knop drukt? Of je kubus omtoveren in een dobbelsteen die je rolt als je op de knop drukt...

6.1 Hoe werkt een drukknop?

Neem een drukknop. Je ziet 4 pinnen en een knop die je kan indrukken, maar welke terugkeert naar de beginpositie zodra je loslaat. Er zijn twee pinnen aan een zijde, en twee aan de overstaande zijde. Leg de knop zo dat een zijde met pinnen naar je toe, en de andere van je weg. Normaal gezien zijn de pinnen links verbonden met elkaar, alsook die rechts. Als je de knop indrukt worden ook links en rechts met elkaar verbonden.

Waarom 4 pinnen en niet 2? Dit is omdat het makkelijker solderen is zo. Je zal vaak meer dan 2 draden aan een knop moeten bevestigen, en dus zijn twee extra pinnen

handig.

Om te zien of je goed begrijpt welke pin wat doet maken we een klein circuit met een LED, een 330Ω weerstand en een drukknop. Gebruik de drukknop om de stroom naar de LED te onderbreken, en gebruik de 5V en GND pinnen van de Arduino om stroom te leveren. Je hebt geen code nodig, en je zou ook een 3V knoopbatterij kunnen gebruiken voor deze oefening.

Het schema is gegeven in Figuur ???. Als je de knop indrukt zou het licht aan moeten gaan.

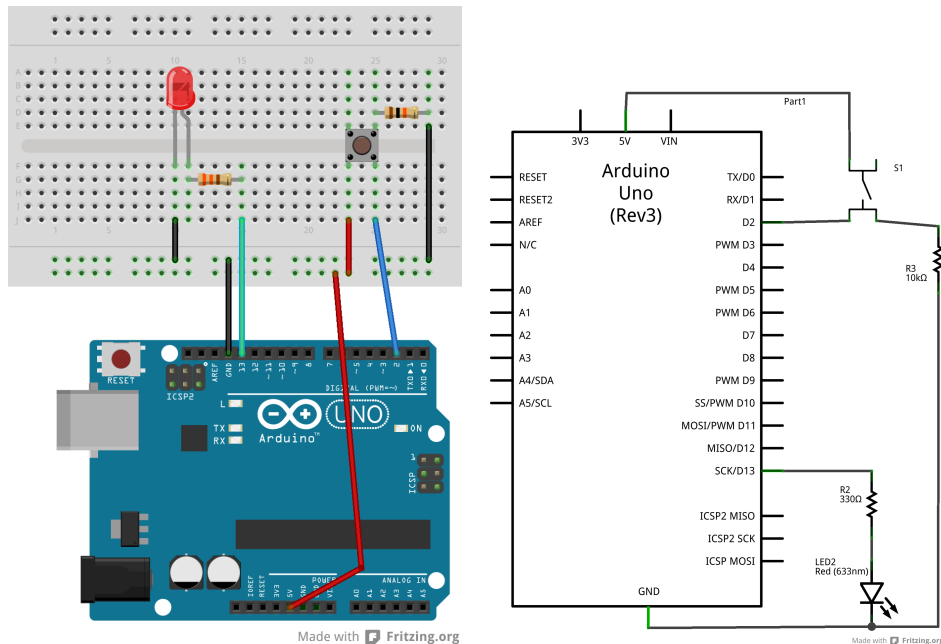


Figure 18: De status van een drukknop lezen om de staat van een LED te beheren.

6.2 Drukknop status lezen

In de vorige oefening hebben we geleerd dat een drukknop de stroom kan onderbreken. Dit kan nuttig zijn in sommige omstandigheden, maar niet voor onze Fe Kubus. In de plaats willen we een knop waardoor we input kunnen krijgen van de gebruiker. In andere woorden, we willen weten als de gebruiker de knop drukt of niet, en we willen daarop reageren via Arduino code.

De Arduino heeft een speciale functie om input te lezen van een digitale

pin: `digitalRead(pin)`. Om deze methode te gebruiken moeten we een pin definiëren als `INPUT` in de `setup` functie: `pinMode(pin, INPUT);`.

De functie zal `HIGH` lezen als de pin op 5V staat en `LOW` als ze op GND is. De truck is dus om een circuit te maken die `LOW` is als de knop niet ingedrukt is en `HIGH` als de knop ingedrukt is. Zo'n circuit is gegeven in Figuur ??.

Laat ons dit circuit analyseren. De pin om de status van de knop te lezen is pin 2. Als deze knop niet is ingedrukt zien we dat de pin verbonden is via zwart met de GND. Het zal dus in de GND staat of `LOW` zijn. Je ziet dat er een weerstand is, maar deze lijkt geen functie te hebben.

Beschouw nu het geval dat de knop ingedrukt is. Er is nu een connectie van links naar rechts in de drukknop. Zwart (GND) is dus verbonden met rood die naar de 5V op

de Arduino gaat. Moest er geen weerstand zijn dan zouden we een kortsluiting krijgen: stroom die vrij stroomt van 5V naar GND. Aldus **moeten** we een weerstand hebben om kortsluiting te voorkomen. Welke weerstand hebben we nodig? Wel, we willen dat pin 2 op 5V staat als de knop ingedrukt is, **én** we willen niet veel stroom gebruiken om dit te verkrijgen. Bijgevolg willen we een grote weerstand! Een grote weerstand zal zorgen dat de weerstand in de draden van de druk-

knop naar de 5V pin verwaarloosbaar is ten opzichte van de grote weerstand, en dus zal pin 2 op 5V staan. Terzelfdertijd zal een grote weerstand ervoor zorgen dat er maar heel weinig stroom zal vloeien over de drukknop, en ons circuit zal dus heel efficiënt zijn. Neem daarom als weerstand eentje van 10000Ω , welke we ook schrijven als $10K\Omega$ of $10k\Omega$. We noemen zo'n weerstand op deze manier gebruikt een *pull-down* (naar GND) of *pull-up* (naar 5V) weerstand.

We kunnen nu Arduino code schrijven die

de status van pin 2 leest, en de LED laten schijnen op basis van die status. Om dit te doen dient ons LED circuit van de vorige oefening aangepast te worden om stroom te trekken van een pin, bv pin 13. Zo kunnen we pin 13 HIGH of LOW zetten om de LED te beheren. Merk op dat de meeste Arduino borden een kleine LED op het bord hebben die verbonden is met pin 13, je zou dus dat kunnen gebruiken ipv een echte LED. De code is gegeven in Code ??.

Code 10

```
1  /*
2  Controlling a simple led via a pushbutton
3
4  Note: on most Arduinos there is already a LED on the board
5  attached to pin 13 so you could leave the LED out
6  */
7
8  #define buttonPin  2 // the number of the pushbutton pin
9  #define ledPin     13 // the number of the LED pin
10
11 // variable that changes
12 int buttonState = 0;
13
14 void setup() {
15     // initialize the LED pin as an output:
16     pinMode(ledPin, OUTPUT);
17     // initialize the pushbutton pin as an input:
18     pinMode(buttonPin, INPUT);
19 }
20
21 void loop(){
22     // read the state of the pushbutton value:
23     buttonState = digitalRead(buttonPin);
24
25     // if the pushbutton is pressed, the buttonState is HIGH:
26     if (buttonState == HIGH) {
27         // turn LED on:
28         digitalWrite(ledPin, HIGH);
29     } else {
30         // turn LED off:
31         digitalWrite(ledPin, LOW);
32     }
33 }
```

6.3 Maar één knop?

Wat kunnen we doen met maar één knop? Meer dan je denkt! Denk aan een aanraakscherm op een tablet. Wat kun je doen met één vinger? Denk aan de computermuis. Wat kun je doen met één knop?

Ja, het antwoord is: veel! Je kan een normale druk hebben, een lange druk, een dubbele druk Je zou zelf kunnen reageren op basis van bepaalde patronen (bv SOS in morse). Het verschillend reageren op verschillende drukken is een kwestie van software. Hoe beter je programmeert hoe beter je kan reageren op dingen die gebeuren. We moeten dus een beetje nadenken over de methode om met een programma te reageren op een knop.

Laat ons enkele concepten definiëren:

- **Input sampling** of **Invoertesten**.

We willen niet teveel tijd besteden met het observeren wat er gebeurt met de knop. De status van de knop elke 40 ms testen (de tijd van een frame in onze Fe Cubus) zou voldoende moeten zijn. Het is misschien een goed idee om de gewone code die loopt op de Arduino niet te blokkeren, en we zouden dus die code kunnen laten aangeven wanneer we de invoer mogen testen, in ons geval de knop.

- Een **event** of **gebeurtenis** is iets dat gebeurt met de knop. Er zijn maar twee mogelijke events : *buttonpress* and *buttonrelease*, in het Nederlands: Knop indrukken en Knop loslaten. Zoals we in de vorige oefening gezien hebben kunnen we deze herkennen door een verandering in de status van de knop. De

eerste is LOW naar HIGH en de tweede HIGH naar LOW.

- Vervolgens hebben we **event handling**. Event handling is het omzetten van events naar *button press type*. Om event handling te doen moeten we de tijd weten sinds een vorig event alsook het type van de vorige druk. Op deze manier kunnen we onderscheiden:

1. **short press** of **korte druk**: een druk en loslaten binnen de 2 seconden
2. **long press** of **lange druk**: een druk en loslaten die langer dan 2 seconden duurt
3. **double press** of **dubbele druk**: a korte druk binnen de 0.5 seconden na een vorige korte druk
4. **extreme long press** of **extreem lange druk**: een druk die langer dan 4 seconden duurt. Het is dan niet nodig te wachten op een loslaten om het druk type vast te leggen.

- Een **INTERRUPT**. Dit is het Engels voor *Onderbreking*. We zullen code hebben lopen op de Arduino, en een knop die gedrukt wordt. We dienen in staat te zijn de code te onderbreken om iets anders te doen. Onze code moet dus het button press type observeren en op basis daarvan de code kunnen onderbreken om te reageren op de druk.

Dit zijn veel concepten om op te nemen in onze code. Laat ons nu code maken die verschillende dingen doet met de LED, om te zien hoe we dit in de praktijk kunnen doen, zie Code ??.

In deze code bevat het eerste gedeelte een functie die invoertesten doet, `inputsampling()` als dit toegelaten is door de variabele

`allowInput`. De invoer wordt dan verwerkt met de functie `eventhandling`. Deze functie bepaalt eerst het **event**, en daarna het type druk, welke opgeslagen wordt in de variabele `presstype`.

Het tweede deel van de code is dan ons programma die een LED doet branden. We kunnen via een lange druk de LED aan of af

doen. Een korte druk dimt de LED een kleine fractie. Na 15 keer een korte druk zal de LED maximaal gedimd zijn, om daarna weer op volle sterkte te branden. We gebruiken PWM om te dimmen, dus gebruiken we bv pin 11 voor de LED. Een dubbele druk zullen we vertalen in het flikkeren van de LED: een halve seconde aan, een halve uit.

Deze code zal dus de variabele `presstype` observeren, en als deze verschillend is van `NOPRESS` de gewenste actie ondernemen. Na het behandelen van een druktype zetten we `presstype=NOPRESS`, zodat we weer wachten op de volgende input.

Je hebt je eerste programma nu af die je toelaat te sturen hoe je Arduino werkt!

Code 11

```
1  /*
2  Controlling a simple led via a pushbutton
3  */
4
5  #define btnpin  2 // the number of the pushbutton pin
6  #define ledPin  11 // a PWM pin for the LED!
7
8  bool test      = false;
9  // variable that changes
10 bool allowInput = true;
11 int btnStatePrev = LOW;
12 int btnState     = LOW;
13
14 // events
15 #define BTNFIXED      0 //no change
16 #define BTNPRESS      1
17 #define BTNRELEASE    2
18 unsigned long timepressed = 0;
19 unsigned long timereleased = 0;
20 // button press types
21 #define NOPRESS        0
22 #define SHORTPRESS     1
23 #define LONGPRESS      2
24 #define DOUBLEPRESS    3
25 #define EXTREMEPRESS   4
26
27 int presstypePrev = NOPRESS;
28 int presstype     = NOPRESS;
29 unsigned long prevpresstime = 0;
30
31 //sensitivity constants
32 #define SHORTPRESSMAXDURATION 2000UL
33 #define LONGPRESSMAXDURATION 4000UL // 4 seconds
34 #define DOUBLEPRESSWAIT      500UL //half a second!
35
36
37 void setup() {
38   // initialize the pushbutton pin as an input:
39   pinMode(btnpin, INPUT);
40   // LED on PWM pin, no setup needed.
41   if (test) {
42     Serial.begin(9600);
43   }
```

```

44 }
45
46 void inputsampling(){
47     //read the pushbutton if allowed
48     if (allowInput) {
49         if (presstype == NOPRESS) {
50             btnStatePrev = btnState;
51             btnState = digitalRead(btnpin);
52             // handle the events if needed
53             eventhandling();
54         } else {
55             // problem, the previous press type has not been handled yet!
56             // note: in our design a SHORTPRESS will be handled even it
57             // later we change this into a DOUBLEPRESS !
58             //if 5 seconds a press is not handled, we remove it
59             if (millis()-prevpresstime > LONGPRESSMAXDURATION){
60                 presstype = NOPRESS;
61                 presstypePrev = NOPRESS;
62             }
63         }
64         allowInput = false;
65     }
66 }
67
68 void eventhandling(){
69     // process the events to set button press types
70     unsigned long currt = millis();
71     unsigned long prevtimereleased;
72     int event = BTNFIXED;
73     if (btnStatePrev != btnState) {
74         // an event happened, handle it
75         if (btnState == HIGH){
76             event = BTNPRESS;
77             timepressed = currt;
78         } else {
79             event = BTNRELEASE;
80             prevtimereleased = timereleased;
81             timereleased = currt;
82         }
83     }
84     switch (event) {
85         case BTNFIXED:
86             // if no event, we can have a long press or certain short press
87             if (timereleased > 0 && timepressed > timereleased
88                 && currt - timepressed > LONGPRESSMAXDURATION){
89                 presstype = EXTREMEPRESS;
90                 presstypePrev = NOPRESS;
91                 //avoid double handling of same events
92                 timepressed = timereleased;
93             } else if (timereleased > 0 && presstypePrev == SHORTPRESS
94                 && currt - timereleased > DOUBLEPRESSWAIT){
95                 presstype = SHORTPRESS;
96                 presstypePrev = NOPRESS;
97             }
98             break;
99         case BTNPRESS:
100             // button is pressed in, we don't sent a press type for this

```

```

101     break;
102 case BTNRELEASE:
103     // button is released, set press type
104     if (timereleased - timepressed <= SHORTPRESSMAXDURATION){
105         //a short press
106         if (presstypePrev == SHORTPRESS
107             && (timepressed - prevtimereleased) < DOUBLEPRESSWAIT) {
108             //a double press
109             presstype = DOUBLEPRESS;
110             presstypePrev = NOPRESS;
111         } else {
112             //a normal short press, store, will be handled if no double press
113             presstype = NOPRESS;
114             //remember it in case it will be double press:
115             presstypePrev = SHORTPRESS;
116         }
117     } else if (timereleased - timepressed <= LONGPRESSMAXDURATION){
118         // a long press
119         presstype = LONGPRESS;
120     } // no need for extremelong test, BTNFIXED handles it
121 }
122 if (test && event != BTNFIXED){
123     Serial.print(presstype);Serial.print(" ");
124     Serial.println(timepressed - prevtimereleased);
125 }
126 // presstype has been set. The app should process it and set it to NOPRESS
127 }
128
129 // our app: a led that burns, dims and flickers
130 unsigned long looptime = 0;
131 #define LEDOFF      0
132 #define LEDON       1
133 #define LEDFLICKER  2
134 int ledstate = LEDOFF;
135 unsigned long flickertime = 0;
136 // 16 dim values
137 int dimvalue = 16;
138
139 void loop(){
140     // every 40 ms we read input, and then process it to set the light
141     unsigned long currt = millis();
142     if (currt - looptime > 40UL) {
143         allowInput = true;
144         inputsampling();
145         looptime = currt;
146     }
147
148     // handle button press if present
149     switch (presstype) {
150     case LONGPRESS:
151     case EXTREMEPRESS:
152         //switch on or off
153         if (ledstate == LEDOFF){
154             analogWrite(ledPin, 255);
155             dimvalue = 16;
156             ledstate = LEDON;
157         } else {

```

```

158     analogWrite(ledPin, 0);
159     ledstate = LEDOFF;
160 }
161 break;
162 case SHORTPRESS:
163     //short press dims light
164     if (ledstate != LEDOFF){
165         dimvalue -= 1;
166         if (dimvalue < 1) { dimvalue = 16;}
167         analogWrite(ledPin, dimvalue*4);
168     }
169     break;
170 case DOUBLEPRESS:
171     //we flicker the led
172     if (ledstate == LEDFLICKER) {
173         ledstate = LEDON;
174     } else if (ledstate != LEDOFF){
175         ledstate = LEDFLICKER;
176         flickertime = currt;
177     }
178     break;
179 }
180 presstype = NOPRESS;
181
182 //if flicker, we flicker once every half second
183 if (ledstate == LEDFLICKER) {
184     if (currt - flickertime == 0) {
185         analogWrite(ledPin, dimvalue*4); //on
186     } else if (currt - flickertime > 1000) {
187         analogWrite(ledPin, dimvalue*4); // restart
188         flickertime = currt;
189     } else if (currt - flickertime > 500) {
190         analogWrite(ledPin, 0); //off
191     }
192 }
193 }

```

6.4 Fe Cube + Drukknop = sturing

We hebben onze Fe Kubus, en we hebben een circuit om de drukknop te gebruiken. Het is dus enkel een kwestie van beiden te combineren, en om dan ons Kubus programma

aan te passen om te reageren op specifieke druk input. De gezamenlijke bekabeling is te zien in Figuur ?? voor het schakelbord. Integreer ze in je Fe Kubus.

In de code dienen we de kubus code te combineren met de drukknop code. De volledige drukknop logica kan letterlijk overgenomen worden. In de oorspronkelijke kubus code moeten we wel nog volgende wijzigingen doen:

- We passen de variabele NRPPATTERN niet langer aan als het patroon af is. De ge-

bruiker zal moeten de knop drukken om een ander effect te zien.

- In de `loop()` functie beginnen we met invoerlezing via de `inputsampling` functie. Vervolgens controleren we het `presstype`. Als de drukknop type `NOPRESS` is en het shot niet gedaan,

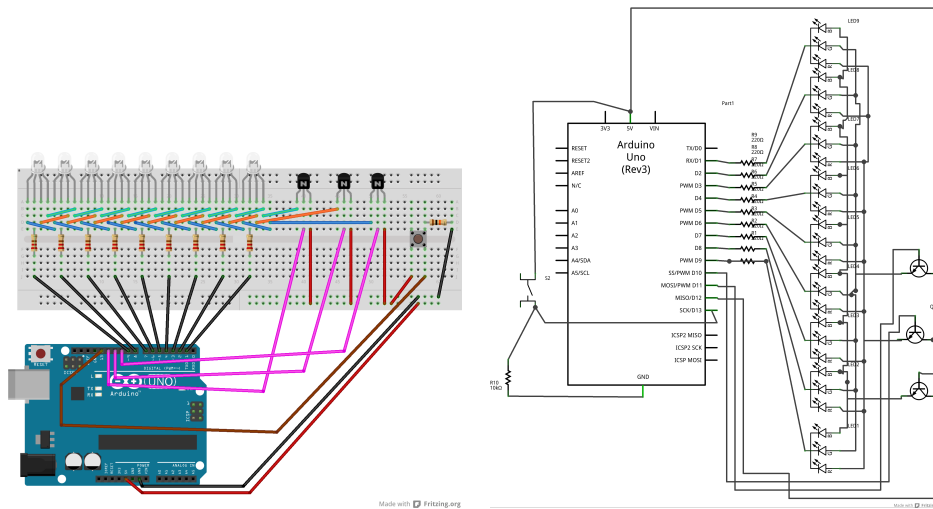


Figure 19: Dezelfde drukknop bekabeling kan gebruikt worden met de Fe Kubus gezien we nog een vrije digitale pin hebben

zal er geen nieuw shot started. In alle andere gevallen moeten we het drukknop type behandelen. We zullen het volgende doen: bij een korte druk zullen we NR_PATTERN verhogen, opdat een nieuw patroon zou getoond worden. Indien het type EXTREMEPRESS is zullen we de kubus af- of aanzetten. Bij DOUBLEPRESS zullen we een nieuw effect tonen: 3 dobbelstenen gooien.

- We schrijven nieuwe shots: een om te

kubus af te zetten, en een om de dobbelsteen te tonen

- We passen de movie functie aan: Als we de dobbelsteen moeten tonen of de LED moeten uitzetten, geven we die shots terug. Anders doen we zoals vroeger: het juiste patroon teruggeven.

De volledige code kun je vinden op https://github.com/ingegno/fecube/blob/master/sketches/old.versions/v01/Fe_cube_05_cube_pbtn/Fe_cube_05_cube_pbtn.ino.