Microcontrollers

Les 3

Getallen

- kunnen decimaal, hexadecimaal of binair afgebeeld worden
- waarde v/h getal blijft steeds gelijk
- wordt binair bewaard op microcontroller
- opletten met omzetten naar ASCII:
  - `Serial.write()` stuurt getal als byte door
  - `Serial.print()` zet dit om naar leesbare tekst

Serial.write()

```
1  void loop(){
2    Serial.write(65); // stuur een byte met waarde 65 (ascii-waarde van "A")
3    Serial.write(0x0A);  // LF
4    Serial.write(0x0D);  // CR
5    int bytesSent = Serial.write("hallo"); //stuur de string "hallo"
6  }
```

- resultaat in seriële monitor:

```
A
hallo
```

Serial.print()

```
1  int x = 65;
2  Serial.println(x);          // stuurt "65" ↘
     →door (byte 0x36 en 0x35) + LF en CR
3  Serial.println(x, DEC);     // standaard ↘
     →decimaal
4  Serial.println(x, HEX);     // hexadecimaal
5  Serial.println(x, BIN);     // binair
```

- resultaat in seriële monitor:

```
65
65
41
1000001
```

Omzetting naar getal

- veel meettoestellen sturen waarden door als string
- string moet op correcte manier in getal omgezet worden:
  - `toInt()`: omzetting naar integer
  - `toFloat()`: omzetting naar kommagetal
- de string moet starten met een cijfer
- conversie stopt bij einde getal

Voorbeeld

```
1  String getal1 = "65.3km";
2  Serial.println(getal1.toInt());
3  Serial.println(getal1.toFloat());
```

- resultaat in seriële monitor:

```
65
65.30
```

Afronden

- standaard stuurt `Serial.println()` van een float 2 cijfers na de komma door
- er wordt correct afgerond
- gedrag kan gewijzigd worden

Voorbeeld

```
1  String getal1 = "65.3691";
2  Serial.println(getal1.toFloat());
     →// weergave standaard 2-cijfers
3  Serial.println(getal1.toFloat(),4);
     →// weergave volledig getal
4  Serial.println(getal1.toFloat(),1);
     →// weergave afronden op 1 cijfer
```

- resultaat in seriële monitor:

```
65.37
65.3691
65.4
```

Strings vergelijken

- in een string moet soms gezocht worden naar een patroon
- een vergelijking is dan nodig
- dit kan met == of met `equals()`
- alfabetisch vergelijken kan met < en >
- verder nog `!=` en `equalsIgnoreCase`

Voorbeeld

```
1  String stringOne = "This";
2  String stringTwo = "this";
3  if (stringOne != stringTwo) Serial.println↘
     →("niet gelijk");
4  if (stringOne.equalsIgnoreCase(stringTwo))↘
     → Serial.println("gelijk (↘
     →hoofdletterongevoelig)");
5  stringOne = "999", stringTwo = "1000";
6  if (stringOne > stringTwo) Serial.println(↘
     →"1 alfabetisch groter dan 2");
```

```
niet gelijk
gelijk (hoofdletterongevoelig)
1 alfabetisch groter dan 2
```

Substrings

- wanneer in een uitlezing meer dan één waarde zit moet er gefilterd worden
- bij de meeste (meet)toestellen is er een vaste lengte per waarde
- men kan dan de functie `substring()` gebruiken
- let op: telling vanaf 0

Voorbeeld

```
1  String stringOne = "Content-Type: text/↘
     →html";
2  if (stringOne.substring(19) == "htm") ↘
     →Serial.println("gelijk aan htm");
3  if (stringOne.substring(14,18) == "text") ↘
     →Serial.println("gelijk aan text");
```

- resultaat in seriële monitor:

```
gelijk aan text
```

Datatypes

- let op voor foutieve resultaten bij tussenberekeningen
- bij deling: als noemer = int dan is resultaat = int
- bij (links) shiften: als bron = 8-bits dan is resultaat 16-bits
- kan opgevangen worden m.b.v. *typeconversie*

Voorbeeld

```
1 Serial.println(16/3);
2 Serial.println(16/3.0);
3 Serial.println(16/float(3));
4 Serial.println(0xF0 << 1, HEX);
5 Serial.println(byte(0xF0 << 1), HEX);
```

- resultaat in seriële monitor:

```
5
5.33
5.33
1E0
E0
```

Interne registers

- beginnende gebruiker moet weinig afweten van de gebruikte processor (Atmega328)
- pin aansturen: `pinMode()`, `digitalRead()` en `digitalWrite()`
- heel handig in gebruik en werkt bij verschillende processors
- meer controle en inzicht door aansturing op processorniveau

Interne aansturing soms noodzakelijk

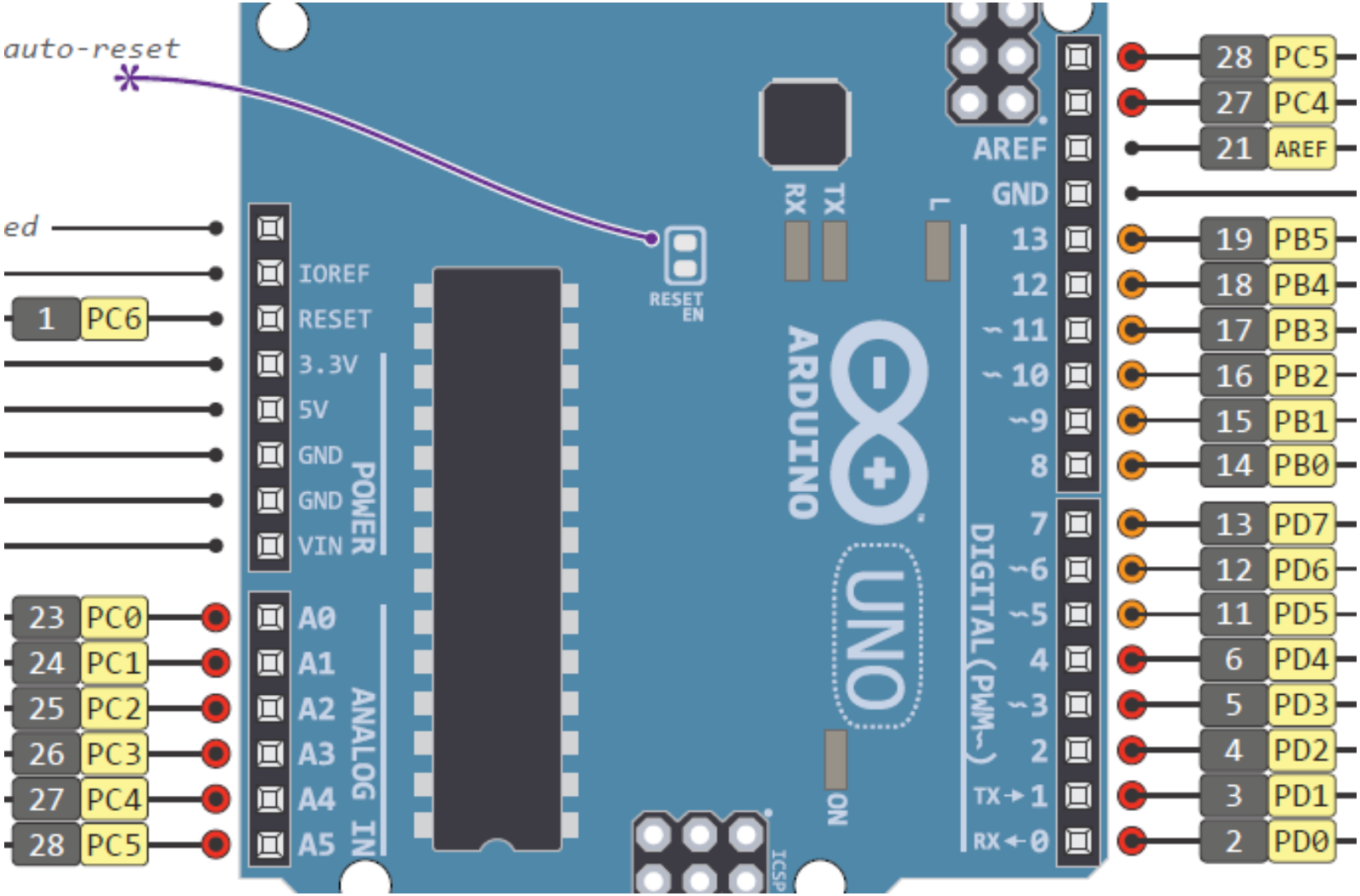
- snelheid: deze aansturing laat een veel snellere controle van de pinnen toe
- volledige poort in één keer aansturen i.p.v. één per één
- geheugen: het gevolg is ook dat de code van je sketch kleiner zal zijn

Poorten

- poorten Atmega328 (Arduino UNO) onderverdeeld in B, C en D
- elke poort bevat een aantal pinnen:
  - B: digitale pinnen 8 tot 13
  - C: analoge pinnen
  - D: digitale pinnen 0 tot 7

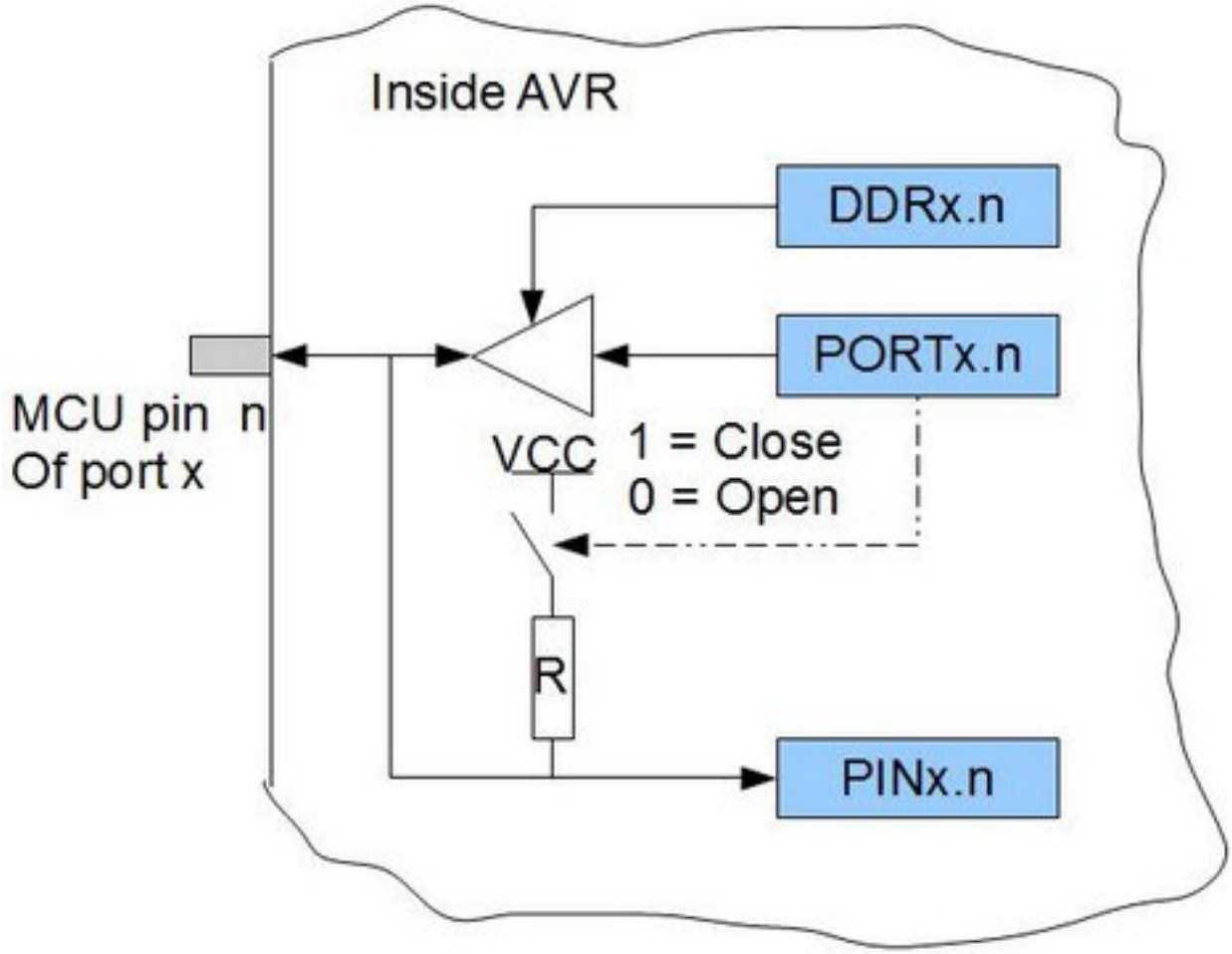
Arduino UNO pinout

vb. PB4 komt overeen met digitale pin 12



DDR-register

Elke poort bevat 3 registers: instellen (DDR), uitlezen (PIN) of aansturen (PORT)



Pinmode

- mode wordt voor alle pinnen van een poort ingesteld (in DDR)
- alle pinnen moeten tegelijk ingesteld worden
- met logische OR- en AND-functies 1 bit manipuleren en mode van 1 pin veranderen

PORT-register

- pin moet in schrijfmodus staan
- aansturen via het PORT-register
- bit hoog maken in PORT-register: pin wordt hoog
- PORT-register stuurt volledige poort aan
- terug met OR- en AND-functie werken om 1 pin aan te sturen

Voorbeeld

```
1  void setup() {
2    DDRD = B11111111; // zet PORTD (digital ↘
       →7-0) als outputs
3  }
4
5  void loop() {
6    PORTD = B11110000; // digital 7-4 HIGH, ↘
       →digital 3-0 LOW
7    delay(1000);
8    PORTD = B00001111; // digital 7-4 LOW, ↘
       →digital 3-0 HIGH
9    delay(1000);
10 }
```

Arduino equivalent

alles individueel: meer tijd en code

```
1  void setup() {
2   for (i=0;i<=7;i++) pinMode(i, OUTPUT);
3  }
4
5  void loop() {
6   for (i=0;i<=3;i++) digitalWrite(i,LOW);
7   for (i=4;i<=7;i++) digitalWrite(i,HIGH);
8   delay(1000);
9   for (i=0;i<=3;i++) digitalWrite(i,HIGH);
10  for (i=4;i<=7;i++) digitalWrite(i,LOW);
11  delay(1000);
12 }
```

Een pin aansturen

- LED (één pin) aansturen via interne registers
- in `setup()` pin 5 instellen in DDR: een 1 5x naar links shiften
- daarna 1 op de juiste positie toevoegen (d.m.v. OR) aan byte
- zelfde methode in `loop()` om led hoog en laag te maken

Een pin aansturen

```
1  #define LED_NUMMER   PB5   // pin 13
2
3  void setup() {
4   DDRB |= (1<< LED_NUMMER );
5  }
6
7  void loop() {
8    PORTB |= (1<< LED_NUMMER );
9    delay(500);
10   PORTB &= ~(1<< LED_NUMMER );
11   delay(500);
12 }
```

Berekening

```
in setup():
-----------
 1d   = B00000001
 1<<5 = B00100000
 DDRB |= B00100000 = Bxx1xxxxx
   (waarbij de bits met een x ongewijzigd blijven)

in loop():
----------
 1<<5    = B00100000
 ~(1<<5) = B11011111
 PORTB &= B11011111 = Bxx0xxxxx
```

Arduino equivalent

```
1  #define LED_NUMMER 13
2  void setup() {
3   pinMode(LED_NUMMER, OUTPUT);
4  }
5
6  void loop() {
7    digitalWrite(LED_NUMMER,HIGH);
8    delay(500);
9    digitalWrite(LED_NUMMER,LOW);
10   delay(500);
11 }
```

Merk op dat wanneer we slechts één pin moeten aansturen het weinig nut heeft om de interne registers te gebruiken.

Input

Arduino moet gegevens kunnen ontvangen:

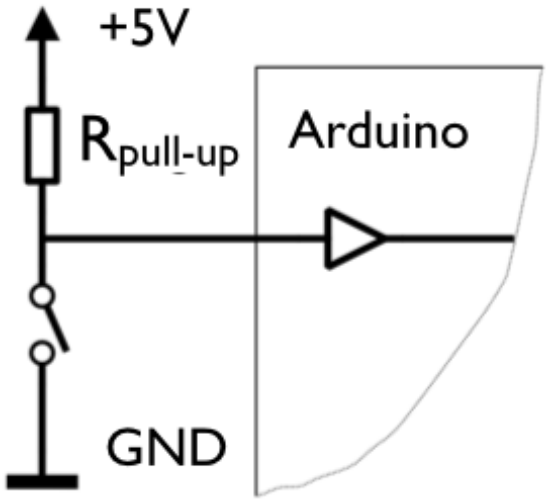
- knoppen inlezen
- sensoren uitlezen
- gegevens ontvangen via seriële poort
- ...

Knoppen

- kunnen ingelezen worden via digitale input
- pin definiëren als input met `pinMode()`
- toestand uitlezen met `digitalRead()`

Pull-up

- input knoppen v/d shield zijn verbonden met GND
- niet ingedrukt: zwevende toestand
- pull-up weerstand maakt deze hoog



Interne pull-up

- shield bevat geen pull-up weerstanden
- oplossing: Arduino heeft deze ingebouwd
- activeren met `pinMode()`
- twee mogelijkheden: `INPUT` en `INPUT_PULLUP`

Voorbeeld

```
1  #define K1   9    // K1 verbonden met pin 9
2  #define LED1 2    // LED1 op pin 2
3
4  void setup() {
5    pinMode(K1, INPUT_PULLUP); // pull-up aan
6    pinMode(LED1, OUTPUT);     // LED uitgang
7    digitalWrite(LED1,LOW);    // LED uit
8  }
9
10 void loop() {
11   if (digitalRead(K1) == LOW) digitalWrite(↘
       →LED1,HIGH);
12 }
```

Oefening

- pas de code aan zodat de LED kan uitgezet worden
- met een druk op knop K2 gaat de LED terug uit

PIN-register

- uitlezen INPUT-pin gebeurt in PIN-register
- ook hier volledige byte lezen
- gewenste bit(s) uitfilteren
- eerst pin in leesmodus plaatsen: bit in DDRB laag
- daarna in `loop()` PINB uitlezen
- vervolgens gebruikte pin uitfilteren

Voorbeeld

```
1  #define K1   PB1    // pin 9
2  #define LED1 PD2    // pin 2
3
4  void setup() {
5   DDRB &= ~(1<< K1);   // K1 input
6   PORTB |= (1<< K1);   // pull-up activeren
7   DDRD |= (1<< LED1); // LED1 uitgang
8   PORTD &= ~(1<< LED1); // LED1 uit
9  }
10 void loop() {
11   if((PINB & (1<<K1)) == 0) PORTD |= (1<< ↘
       →LED1);
12 }
```

Oefening

- pas de code aan zodat de LED kan uitgezet worden
- met een druk op knop K2 gaat de LED terug uit

Polling

- methode in voorgaand voorbeeld gebruikt polling
- in `loop()` wordt continu de toestand van de knoppen uitgelezen
- dit belast de CPU vrij intensief, kan dan weinig andere zaken doen
- bij veel ander werk worden de knoppen niet continu uitgelezen: trage reactie
- oplossing: interrupts (zie later)

Tellen

- bij iedere druk op de knop wordt een teller verhoogd
- zonder delay zal één druk ongeveer 40x geteld worden
- dit komt doordat de software sneller is dan het indrukken van de knop
- ideale wachttijd is ongeveer 100ms
- bij het blijven indrukken wordt er om de 100ms geteld

Tellen

```
1 void loop() {
2   if (digitalRead(K1) == LOW) {
3       Serial.println(teller);
4       teller++;
5   }
6   delay(100);
7 }
```

Aantallen tellen

- één druk moet als één geteld worden
- dit onafhankelijk van de indruktijd
- oplossing: wachten totdat de knop terug losgelaten wordt

Aantallen tellen

```
1  void loop() {
2    if (digitalRead(K1) == LOW) {
3       Serial.println(teller);
4       teller++;
5       while(digitalRead(K1) == LOW);
6     }
7   // LED togglen
8     digitalWrite(LED1,!digitalRead(LED1));
9     delay(50);
10 }
```

Aantallen tellen

- voorgaand voorbeeld telt correct (op dender na)
- reageert onmiddellijk
- andere code wordt geblokkeerd tijdens wachten op loslaten
- LED toggelt niet tijdens indrukken van knop
- oplossing: code vrijgeven en variabele togglen

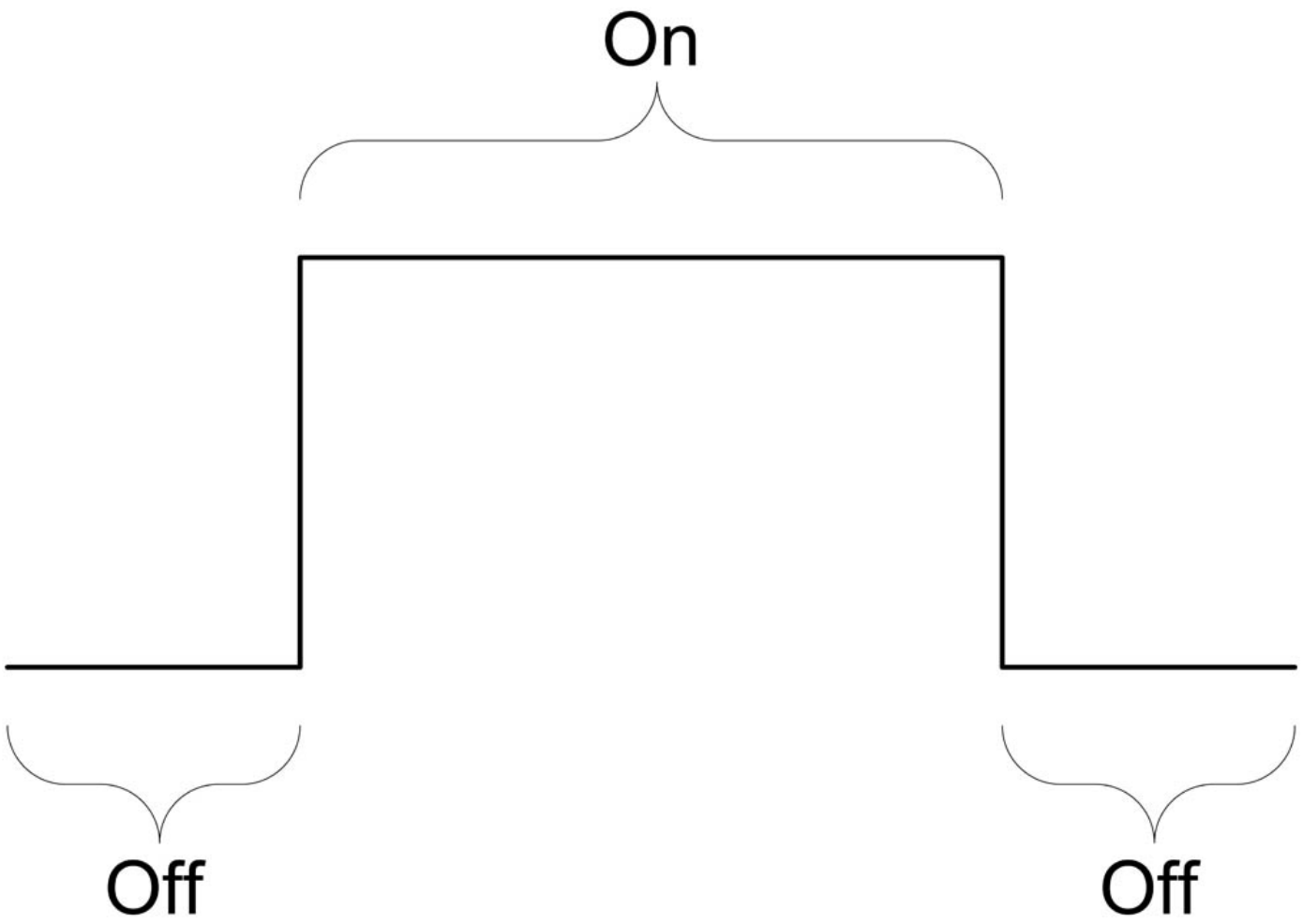
Aantallen tellen

```
1  boolean statusK1;
2
3  void loop() {
4    if (digitalRead(K1)==HIGH) statusK1=false;
5    if ((digitalRead(K1)==LOW)&&(statusK1==↘
       →false)) {
6      Serial.println(teller);
7      teller++;
8      statusK1=true;
9    }
10   digitalWrite(LED1,!digitalRead(LED1));
11   delay(50);
12 }
```

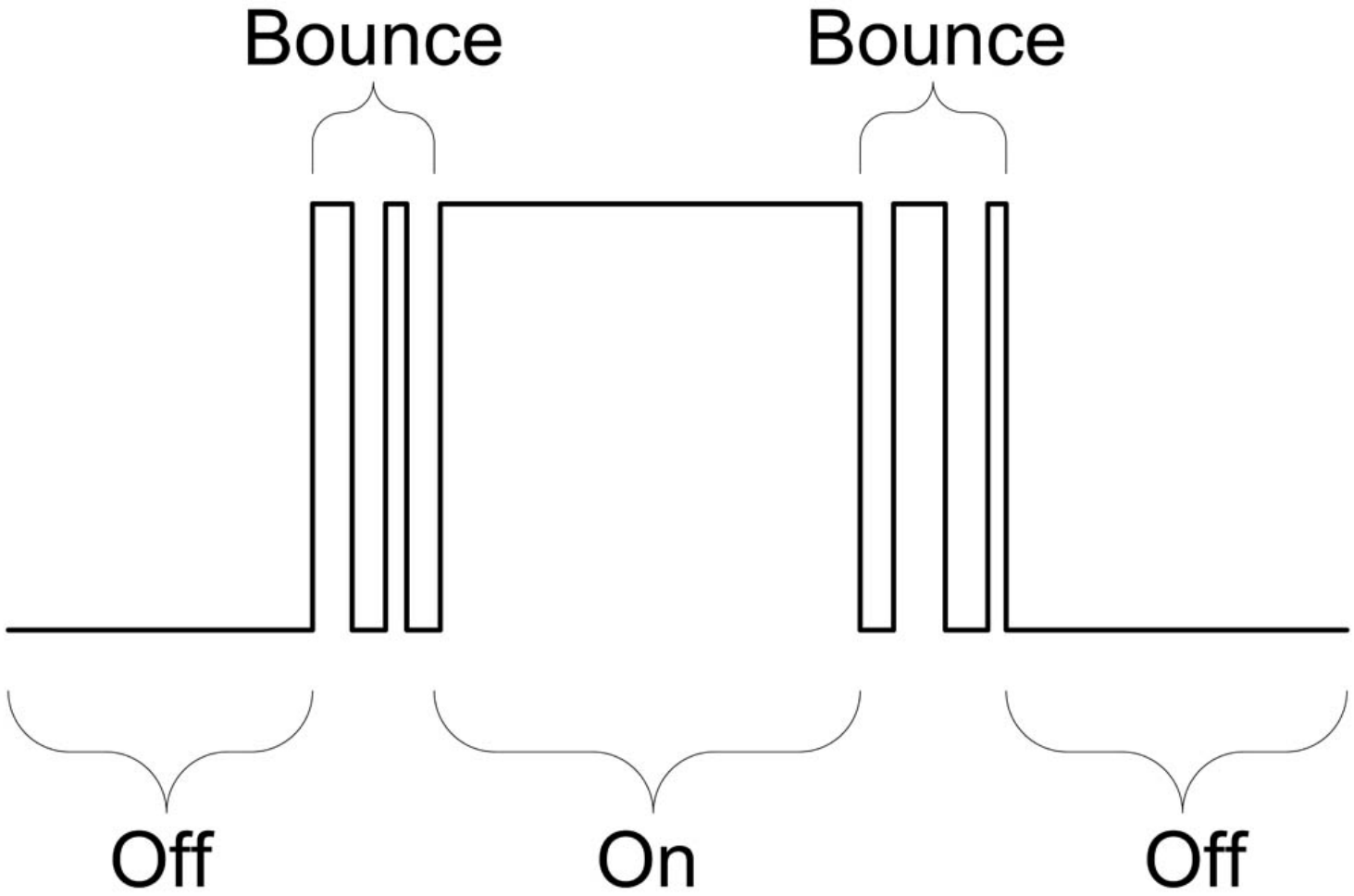
Dender (bounce)

- ook wel glitches genoemd
- bij elke druk op de knop treedt er dender op
- ernst afhankelijk van kwaliteit knop
- bij elke overgang is er dus een onstabiele toestand
- dit kan foutieve output veroorzaken

Zonder dender



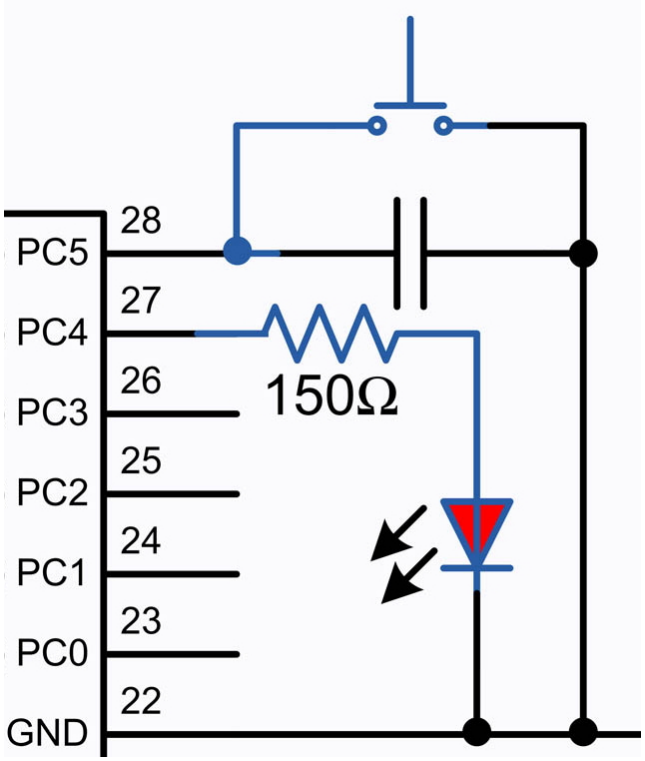
Met dender



Oplossingen

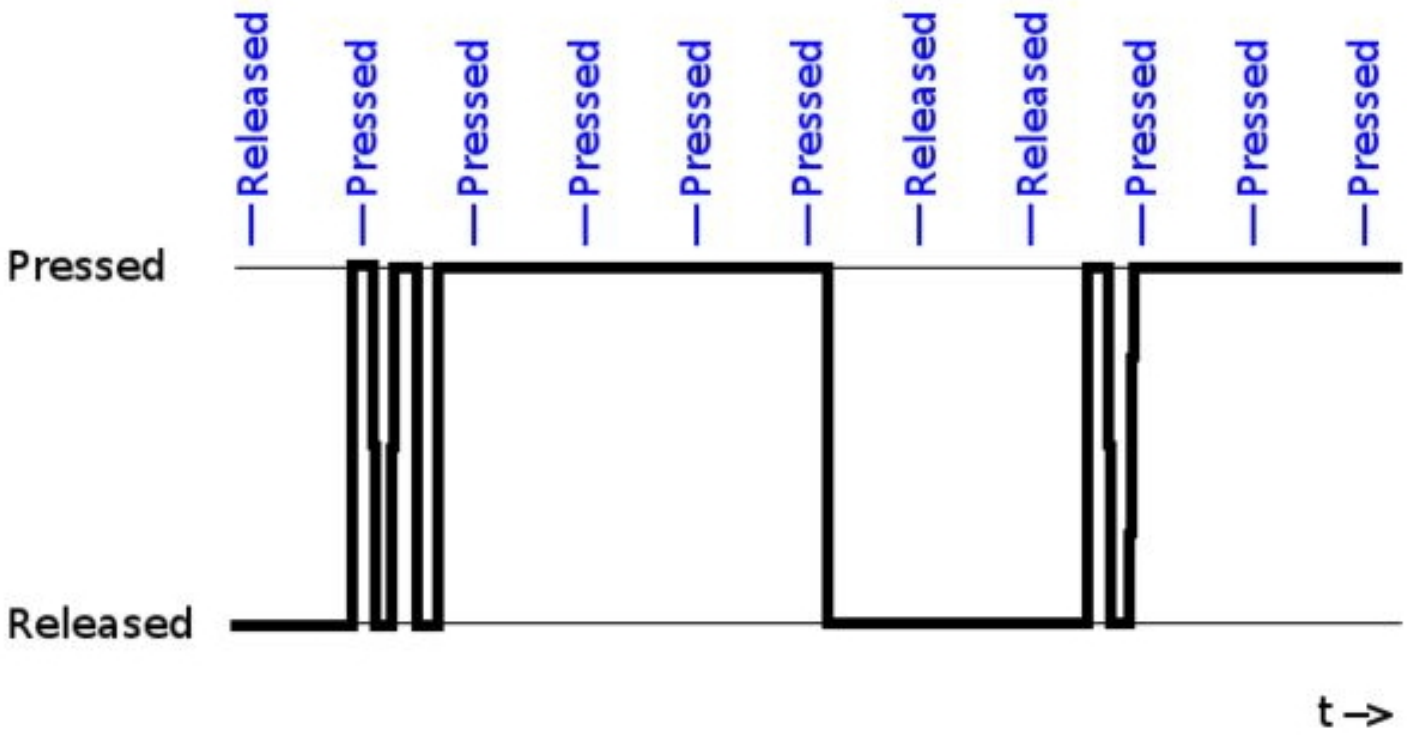
- **hardware**: verander de schakeling zodat geen dender optreedt
- **software**: pas de code op de Arduino aan zodat juist gereageerd wordt

Hardware



- de condensator wordt opgeladen via de pull-up weerstand
- zorgt voor de nodige tijdsvertraging om dender te overbruggen

Software



- meestal wordt een tijdsvertraging ingebouwd
- er wordt langer gewacht dan de dendertijd
- dendertijd is enkele ms, meestal geen extra delay nodig

Verkeerd aantal (1)

```
1 void loop() {
2   if (digitalRead(K1) == LOW) {
3       Serial.println(teller);
4       teller++;
5       while(digitalRead(K1) == LOW);
6    }
7 }
```

- telt meestal correct (soms 2x)
- uitvoeren `Serial.println()` duurt lang genoeg

Verkeerd aantal (1)

```
1 void loop() {
2   if (digitalRead(K1) == LOW) {
3       teller++;
4       while(digitalRead(K1) == LOW);
5    }
6   if (teller==10) {
7     Serial.println("10x ingedrukt?");
8   }
9 }
```

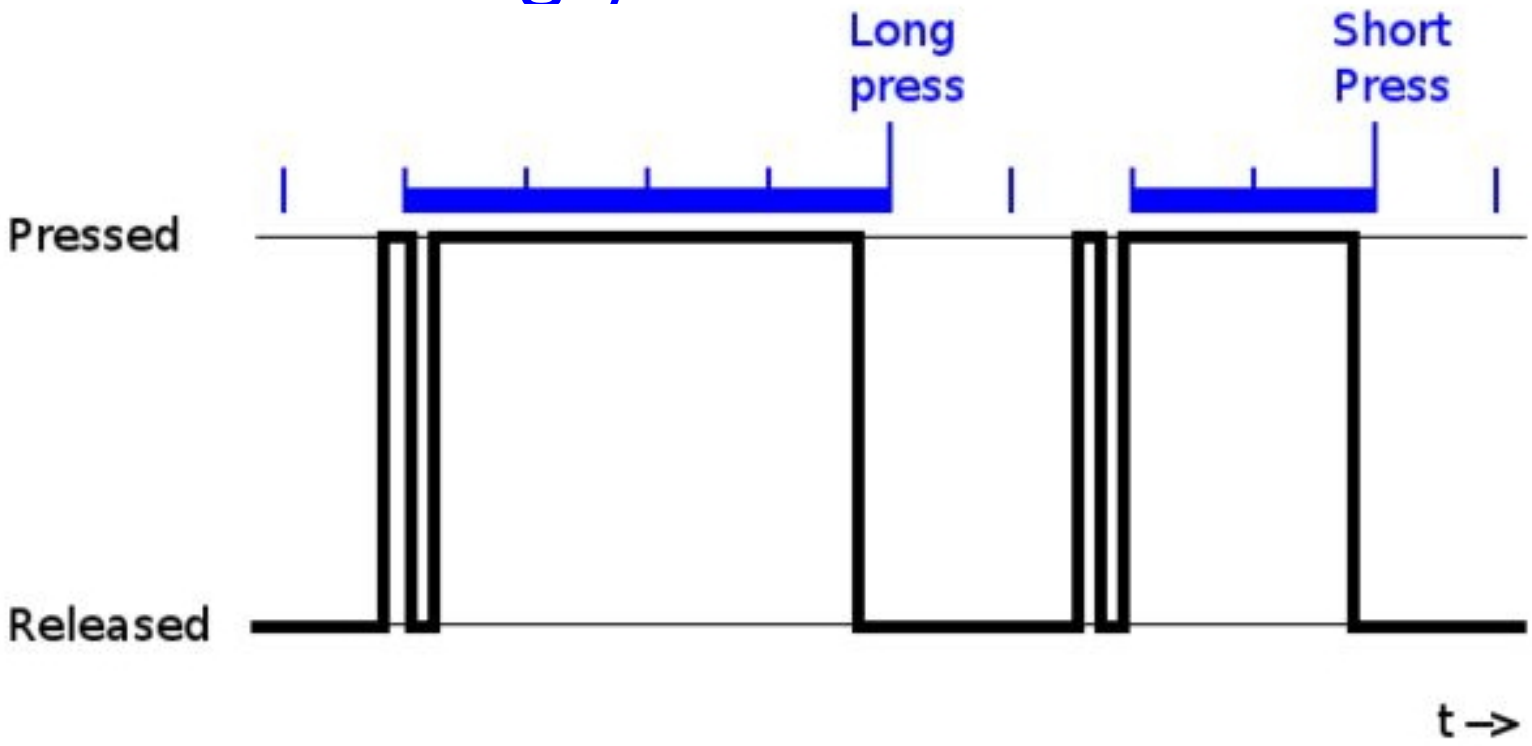
- telcode zo snel mogelijk
- reeds output bij 5 tot 7x indrukken

Correct aantal

```
1  void loop() {
2    if (digitalRead(K1) == LOW) {
3       delay(25);  // denderdelay
4       teller++;
5       while(digitalRead(K1) == LOW);
6       delay(25);  // denderdelay
7     }
8    if (teller==10) {
9      Serial.println("10x ingedrukt");
10   }
11 }
```

- vertraging voorkomt dender
- telt correct, onafhankelijk van andere code

Lange/korte drukken



- verschillende functies voor korte of lange druk
- wordt veel bij uurwerken gebruikt
- na iedere druk kijken hoeveel tijd er gepasseerd is

`millis()`

- toont de verstreken tijd na power up of reset in milliseconden
- datatype = unsigned long
- loopt over na +/- 50 dagen
- handig om tijdsverschillen te meten
- onafhankelijk van tussenliggende code

Oefening `millis()`

```
1  unsigned long time;
2
3  void loop(){
4    Serial.print("Tijd: ");
5    time = millis();
6    // print tijd na programma start
7    Serial.println(time);
8    // om de seconde output
9    delay(1000);
10 }
```

- welke getallen verschijnen er?

Lange/korte drukken

```
1  void loop() {
2    tijd = millis();
3    if (digitalRead(K1) == LOW) {
4       while(digitalRead(K1) == LOW);
5       if ((millis() - tijd) > 200) {
6         Serial.println("lang ingedrukt");
7       }
8       else Serial.println("kort ingedrukt");
9       delay(50);  // denderdelay
10    }
11 }
```

Meerdere knoppen

- geen problemen als knoppen onafhankelijk zijn
- let op totale tijdsvertragingen
- conflicten: wie heeft voorrang?
- noodstop moet bv. steeds voorrang hebben

Meerdere knoppen

```
1 void loop() {
2   if (digitalRead(K1) == LOW) digitalWrite(↘
    →LED1,HIGH);
3   if (digitalRead(K2) == LOW) digitalWrite(↘
    →LED1,LOW);
4 }
```

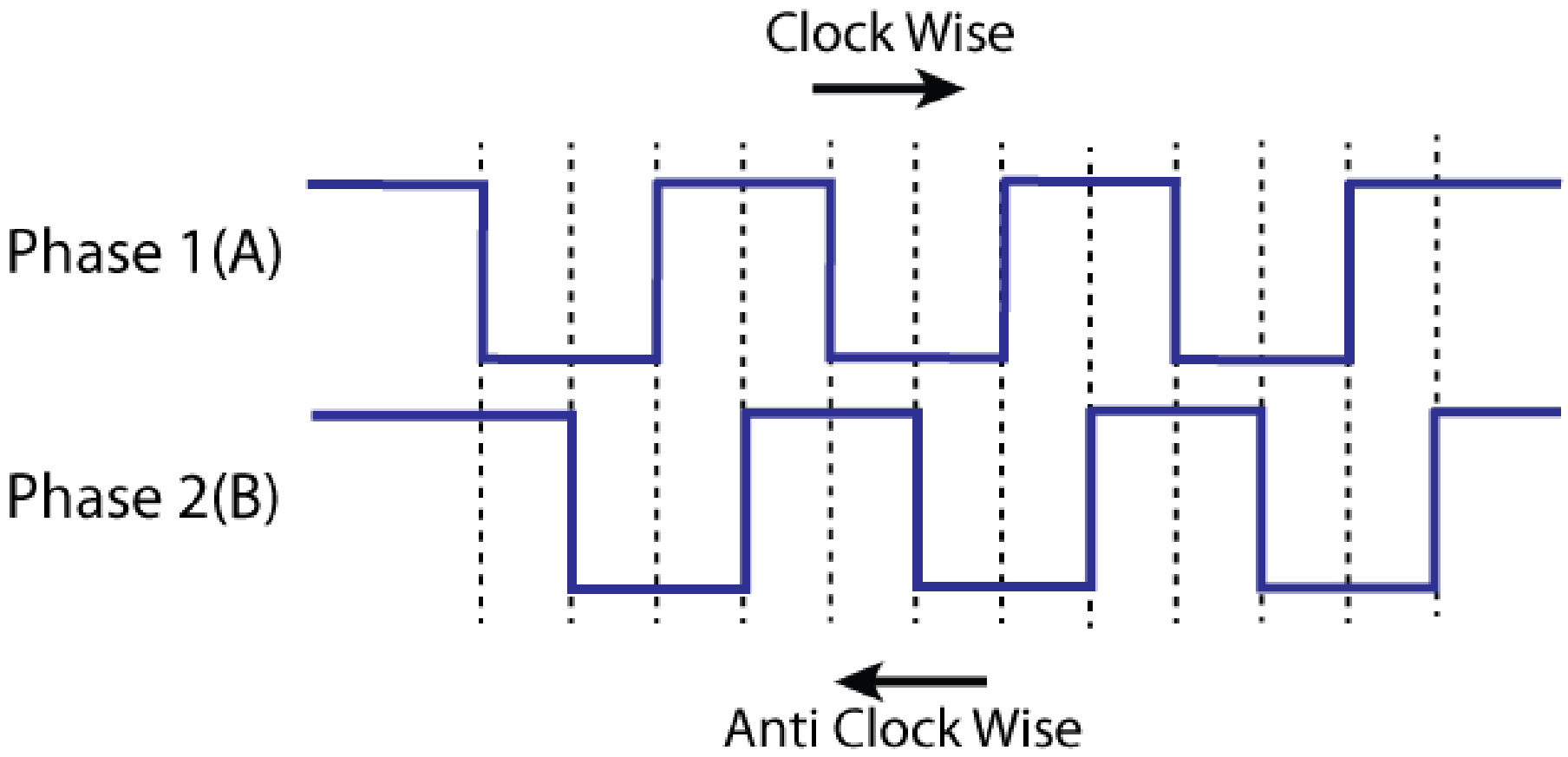
- wat gebeurt er als beide knoppen tegelijk ingedrukt worden?
- is er een voorrangsregeling?

Rotary encoder

- digitaal alternatief voor potentiometer
- wordt bv. gebruikt als volumeknop
- geeft pulsen bij iedere draaibeweging
- draaisnelheid: meten met pulslengte
- volgorde pulsen afhankelijk van draairichting

Rotary encoder

Rotary encoder



Positie bepalen

- als fase A van L naar H gaat en B is laag: optellen
- als fase A van L naar H gaat en B is hoog: aftrekken
- in de `loop()` moet dus enkel A gecontroleerd worden
- indien verandering: stand van B controleren

Positie bepalen

```
1  void loop() {
2    n = digitalRead(encoderPinA);
3    if ((encoderPinALast==LOW)&&(n==HIGH)) {
4      if (digitalRead(encoderPinB) == LOW) {
5        encoderPos++;
6      } else {
7        encoderPos--;
8      }
9      Serial.println(encoderPos);
10   }
11   encoderPinALast = n;
12 }
```

Digitale output

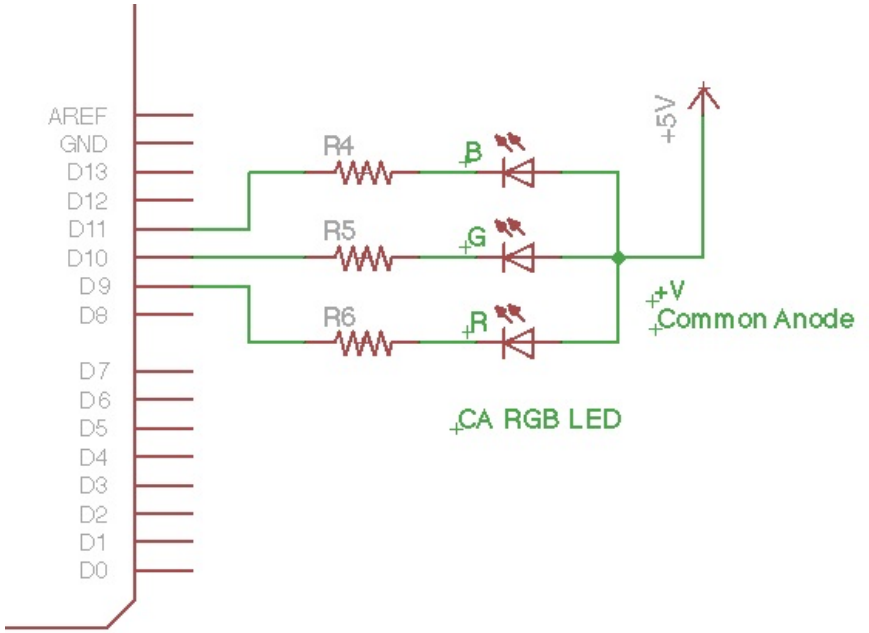
- pinnen niet onbegrenst belasten
- er is een maximum stroom per pin en per poort
- verschil tussen stroom *sourcen* en *sinken*
- stroombeperking voorzien (bv. bij LED's)

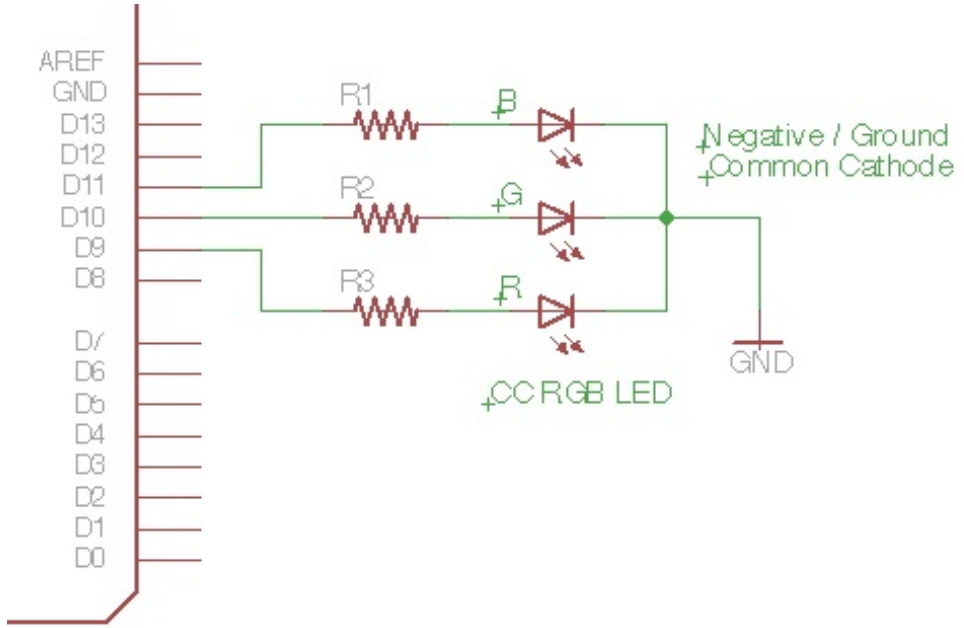
Maximum stromen

- max. 40mA per pin (continu: 20mA)
- max. 150mA per poort (8-bits) leveren (source)
- max. 100mA per poort ontvangen (sink)
- een pin kan dus perfect een LED (20mA) voeden
- let op met meerdere LED's per poort

LED aansluiten

- kan op 2 manieren: sourcen of sinken
- bij RGB-LED of 7-segment rekening houden met CA of CC





Oefening LED clock shield

- bereken de stromen door de LED's
- de spanningsvallen zijn 2V (R en G) en 3,4V (B)