

Odisee - Studiegebied IWT
Opleiding Bachelor Elektronica-ICT

Mario Wyns

IoT MICROCONTROLLERS

Inhoudsopgave

1 Inleiding	11
1.1 De Arduino	11
1.1.1 De UNO	11
1.1.2 Het programmeren	12
1.1.3 De installatie	13
1.2 Open source	13
1.3 Een eerste programma	13
1.3.1 Digitale poorten	13
1.3.2 C++ taal	14
1.3.3 Datatypes	14
1.3.4 Eerste programma: blink	15
1.3.5 setup() en loop()	16
1.3.6 Seriële output	16
2 Overzicht hardware	19
2.1 Soorten bordjes	19
2.1.1 Arduino Mega	19
2.1.2 Arduino Micro	20
2.1.3 Arduino Lilypad	20
2.1.4 Arduino Nano 33 BLE	20
2.1.5 ESP8266	21
2.1.6 ESP32	21
2.2 Shields	22
2.2.1 Motor shield	22
2.2.2 GPRS shield	22
2.2.3 Ethernet shield	23
2.2.4 SD-card shield	23

2.2.5	Display shield	23
2.2.6	Clock shield	23
2.3	Features Arduino Uno en ESP8266	25
3	Datavoorstellingen	27
3.1	De BIT	27
3.2	Talstelsels	27
3.2.1	Decimaal	27
3.2.2	Binair	28
3.2.3	Hexadecimaal	29
3.3	Dataorganisatie	30
3.3.1	Bit	30
3.3.2	Nibble	30
3.3.3	Byte	30
3.3.4	Word	30
3.4	Karaktercodes	30
3.4.1	ASCII	31
3.4.2	Unicode	31
4	Wiskundige en logische functies	35
4.1	Wiskundige functies	35
4.1.1	Optellen	35
4.1.2	Aftrekken	35
4.1.3	Vermenigvuldigen	36
4.1.4	Deling	37
4.2	Logische functies	37
4.2.1	AND	38
4.2.2	OR	38
4.2.3	XOR	39
4.2.4	NOT	40
4.2.5	Shiften	40
4.2.6	U en L notatie	40
4.2.7	Bit manipulaties	41
5	Getalnotatie en strings	43
5.1	Getalnotatie	43
5.1.1	Versturen	43
5.1.2	Omvzetting naar getal	44
5.1.3	Voorbeeld	44
5.1.4	Afronden	45
5.2	Strings	45
5.2.1	Strings vergelijken	45
5.2.2	Substrings	46

	5
5.3 Datatypes	47
5.3.1 Voorbeeld	47
6 Input en output	49
6.1 Interne registers	49
6.1.1 Poorten	49
6.1.2 DDR-register	50
6.1.3 PORT-register	50
6.2 Knoppen	52
6.2.1 Pull-up	53
6.2.2 PIN-register	54
6.2.3 Polling	55
6.3 Dender (bounce)	57
6.3.1 Dender voorkomen	58
6.4 Lange en korte drukken	60
6.4.1 millis()	60
6.4.2 Lang en kort onderscheiden	61
6.5 Meerdere knoppen	61
6.6 Rotary encoder	62
6.6.1 Positie bepalen	63
6.7 Digitale output	63
6.7.1 Maximum stromen	63
6.7.2 LED aansluiten	64
6.8 Interrupts	64
6.8.1 Externe interrupts	66
6.8.2 Pin Change interrupts	68
6.8.3 Volatile	69
6.9 Output signaal versterken	70
6.9.1 Transistor	70
6.9.2 Relais	71
6.9.3 H-brug	71
6.9.4 Optocoupler	72
7 Bibliotheken	75
7.1 Installatie en locatie	75
7.1.1 Installatie	75
7.1.2 Locatie	76
7.2 Soorten bibliotheken	76
7.2.1 Standaard bibliotheken	76
7.2.2 Extra bibliotheken	77

8 Displays	79
8.1 LCD	79
8.1.1 Aansluiting	79
8.1.2 Protocol	80
8.1.3 Custom Characters	81
8.2 TFT	82
8.3 7-segment	82
8.3.1 Opbouw	83
8.3.2 Multiplex	83
9 Timers en tijd	87
9.1 Timers	87
9.1.1 Timer interrupts	87
9.1.2 Timer1	87
9.1.3 Werking	88
9.1.4 Snelheid	88
9.1.5 Prescaler	89
9.1.6 Nauwkeurigheid (16-bit)	89
9.1.7 Invloed	90
9.1.8 Bibliotheek	90
9.1.9 Counter	91
9.2 Interne registers	91
9.2.1 Timer	91
9.2.2 Counter	94
9.3 Tijd	97
9.3.1 time_t	97
9.3.2 Time en timer0	99
9.3.3 Schrikkeljaar en weekdag	99
9.3.4 Externe synchronisatie	99
9.3.5 Hoge nauwkeurigheid	101
9.4 PWM	103
9.4.1 analogWrite()	104
9.4.2 Nauwkeurigheid	104
9.5 pulseIn()	105
10 Analoge signalen	107
10.1 Soorten ADC's	107
10.1.1 De ideale ADC	107
10.1.2 De perfecte ADC	107
10.1.3 ADC's in AVR controllers	107
10.2 Mogelijke fouten	108
10.2.1 Offset error	108
10.2.2 Gain error	109

10.2.3 Non-Linearity	109
10.2.4 Andere fouten	109
10.3 Theorema van Nyquist	109
10.3.1 ADC bandbreedte	110
10.4 Praktisch gebruik van de ADC op de Arduino UNO en ESP8266	111
10.4.1 Arduino UNO	111
10.4.2 ESP8266	111
10.4.3 Analoge sensors	112
10.5 Analoge output	115
10.5.1 Audio player	116
11 Communicatieprotocollen	117
11.1 Seriële poort	117
11.1.1 Protocol	117
11.1.2 Gebruik op de Arduino	119
11.1.3 Code	120
11.1.4 Software serial	122
11.2 I ² C	123
11.2.1 De I ² C-bus	123
11.2.2 Adressering	124
11.2.3 Kloksignaal en timing data	124
11.2.4 Praktische voorbeelden	125
11.3 SPI	134
11.3.1 Werking	134
11.3.2 Arduino ICSP-header	135
11.3.3 Praktische voorbeelden	135
12 Bluetooth	139
12.1 Indelingen	139
12.1.1 Class	139
12.1.2 Versies	139
12.1.3 Profielen	140
12.2 Serial Port Profile	140
12.2.1 Bluetooth Module	140
12.2.2 Smartphone	144
12.2.3 Andere draadloze verbindingen	145
13 Geheugen	147
13.1 Geheugenstructuur	147
13.1.1 Flash	147
13.1.2 EEPROM	147
13.1.3 SRAM	148
13.2 Geheugen optimaliseren	149

13.3 Gebruik EEPROM	150
13.3.1 Bibliotheek	150
13.3.2 Schrijven en lezen	151
13.3.3 Extern geheugen	152
14 Netwerken	153
14.1 Hardware	153
14.1.1 Arduino UNO	153
14.1.2 ESP8266	154
14.2 TCP-verbindingen	155
14.2.1 Voorbeeld	155
14.2.2 TCP-client in hogere programmeertalen	156
14.3 MQTT	158
14.3.1 Architectuur	158
14.3.2 Software	159
14.4 Het web protocol	160
14.4.1 HTTP-request	160
14.4.2 HTTP-response	161
14.4.3 HTTP op de ESP8266	162
14.4.4 De ESP8266 aansturen via het web	165
15 MicroPython	169
15.1 Python	169
15.2 MicroPython op de ESP8266	170
15.2.1 De REPL prompt	170
15.2.2 Mogelijkheden	170
15.2.3 Start up scripts	170
A Geschiedenis van de Computer	173
A.1 Het Mechanisch Tijdperk	173
A.1.1 Abacus	173
A.1.2 Charles Babbage (°1791 †1871)	174
A.2 Het Elektrisch Tijdperk	175
A.2.1 Konrad Zuse (°1910 †1995)	175
A.2.2 Enigma	176
A.2.3 Colossus (1943)	176
A.3 Belangrijke Uitvindingen	177
A.3.1 Transistor	177
A.3.2 Geïntegreerd circuit	177
A.4 Het Microprocessor Tijdperk	178
A.4.1 Intel 4004 en 8008	178
A.4.2 Intel 8086	179
A.4.3 Systemen worden complexer	179

A.4.4 MIPS	181
B Project prototyping	183
B.1 Project benodigdheden	183
B.1.1 Temperatuurmonitor	184
B.2 Benodigde interfaces	184
B.3 Lijst met componenten	185
B.3.1 Temperatuurmonitor	185
B.4 Schema tekenen	185
B.4.1 Breadboard opstelling	186
B.5 De sketch schrijven	186
B.5.1 Temperatuurmonitor	187
B.6 Testen	189

1

Inleiding

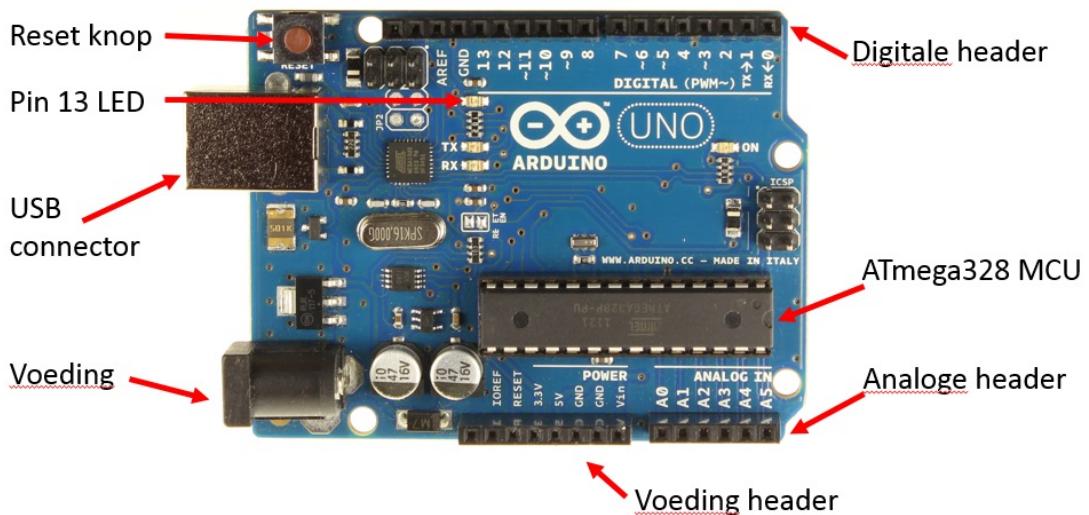
1.1 De Arduino

1.1.1 De UNO

In deze cursus gaan we o.a. gebruik maken van de Arduino UNO.

Dit is een eenvoudig microcontrollerbordje, gebaseerd op Atmega328-processor van Atmel. Het bordje biedt USB-ondersteuning dankzij een extra IC.

De Arduino UNO (figuur 1.1) zelf bevat weinig input en output mogelijkheden zoals knoppen en LED's. Er staan gelukkig heel veel shields ter beschikking met allerlei uitbreidingsmogelijkheden.

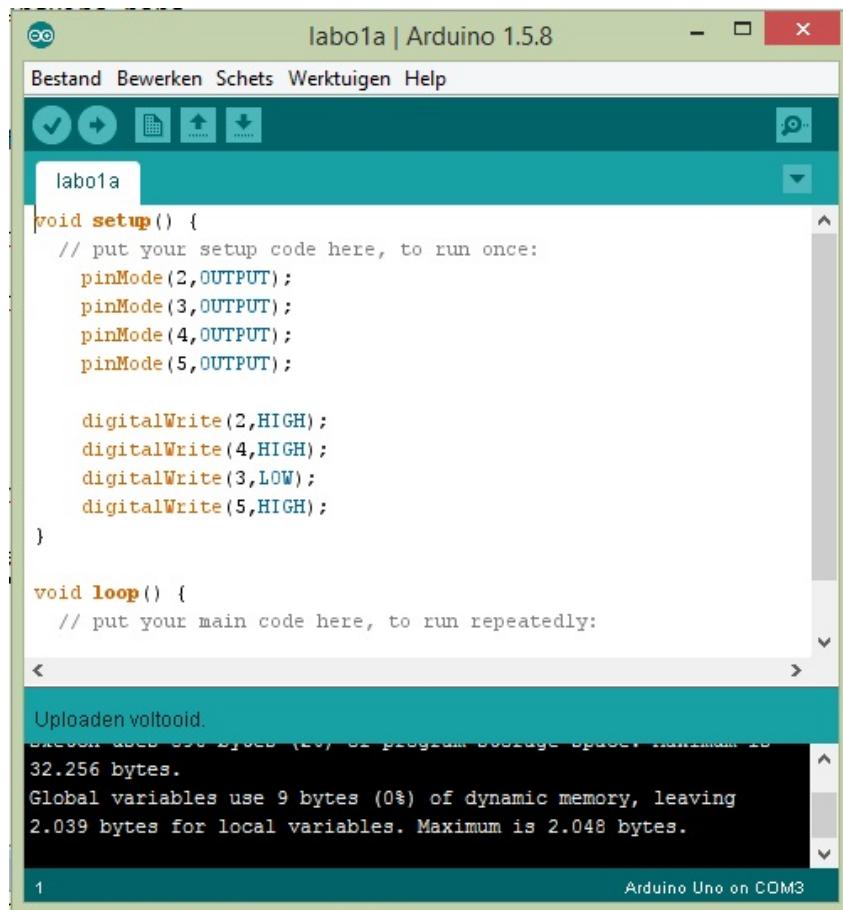


Figuur 1.1: Arduino UNO

1.1.2 Het programmeren

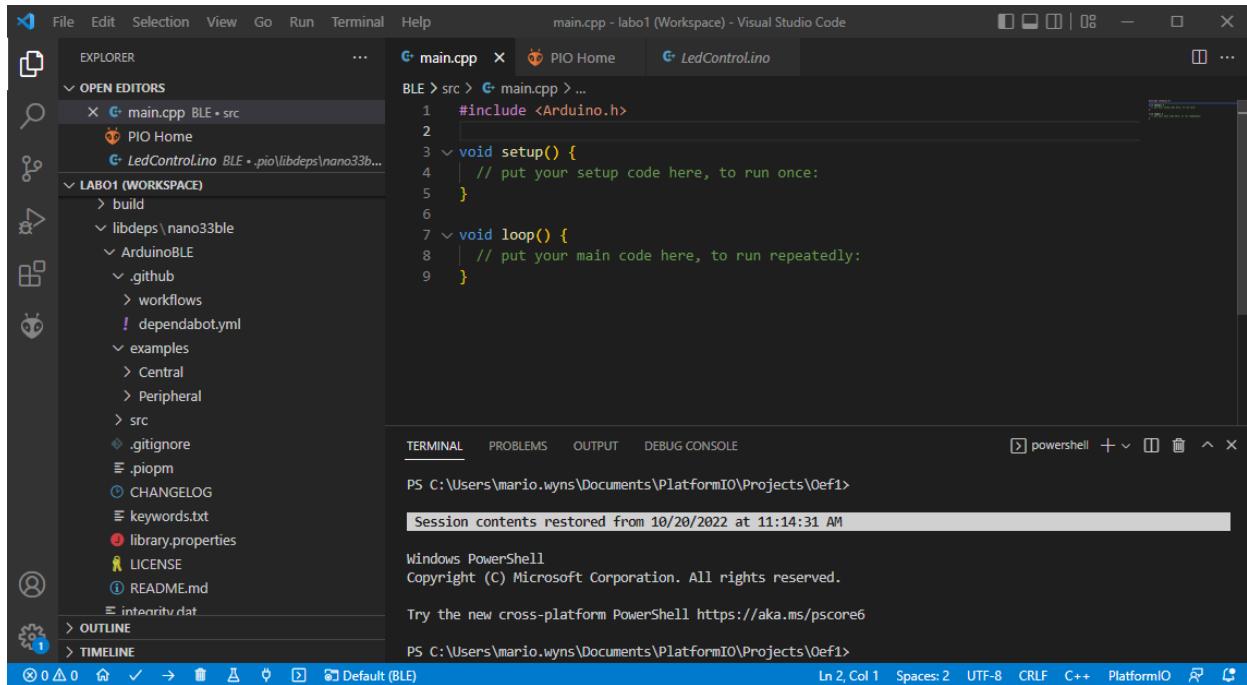
Eén van de voordelen van de Arduino is dat het bordje eenvoudig te programmeren is via een USB-aansluiting. Via dezelfde aansluiting kan het bordje ook van stroom voorzien worden. De voeding kan achteraf, indien het gewenste programma in orde is, ook voorzien worden via een DC-adapter.

Er is een speciale IDE (figuur 1.2) ontwikkeld voor de Arduino waarbij eenvoud centraal staat. Deze programmeeromgeving laat toe om de verschillende Arduino types te programmeren. De eenvoud heeft ook zijn nadelen, zo is er geen debugger aanwezig. Dit kan deels vervangen worden door de seriële monitor.



Figuur 1.2: Arduino IDE

Voor deze cursus gaan we een uitgebreidere omgeving gebruiken, namelijk PlatformIO (figuur 1.3) in combinatie met Visual Studio Code. Deze omgeving heeft o.a. als extra code-completion, debugging en GIT-integratie.



Figuur 1.3: PlatformIO

1.1.3 De installatie

Voor de installatie van PlatformIO en Visual Studio Code kan naar de tekst *Getting Started with PlatformIO* verwezen worden op de website <https://dronebotworkshop.com/platformio/>

1.2 Open source

De hardware van de Arduino is open source, iedereen mag dit dus namaken. Het gevolg is dat het heel populair is geworden en er veel klonen van andere merken bestaan. Deze zijn echter niet altijd zo stabiel als de oorspronkelijke versie, een test is dus steeds noodzakelijk.

1.3 Een eerste programma

1.3.1 Digitale poorten

Een digitale poort is een pin op het bordje waarop je iets kan aansluiten. Hierbij kan de toestand ofwel hoog of laag zijn. Op de UNO is dit respectievelijk 5V en 0V.

De poort kan in twee richtingen werken:

- input: om informatie in te lezen
- output: om iets aan te sturen

De richting wordt ingesteld met de opdracht `pinMode()`. Dit moet éénmalig gebeuren per pin in het begin van het programma. Wanneer de pin ingesteld wordt als input zijn er twee mogelijkheden: met of zonder pull-up weerstand, hierover later meer.

In het codevoorbeeld hieronder wordt pin 13 als output gebruikt, pin 2 en 3 als input. Bij pin 3 wordt er intern een pull-up weerstand geactiveerd. Daarna wordt pin 13 hoog gemaakt, er zal dus bij de UNO 5V op de uitgang staan. Vervolgens wordt de toestand van pin 2 uitgelezen waar dan de gepaste actie kan aan gekoppeld worden.

```

1  pinmode(13, OUTPUT);           // uitgang
2  pinmode(2, INPUT);            // ingang (laag)
3  pinmode(3, INPUT_PULLUP);    // ingang (hoog)
4
5  digitalWrite(13, HIGH);       // schrijven
6  if (digitalRead(2) == LOW) ... // lezen

```

1.3.2 C++ taal

Deze cursus gaat er van uit dat er reeds voldoende programmeervaardigheden gekend zijn.

De Arduino wordt geprogrammeerd in C++, dit is een universele taal en wordt bijna door elke processor ondersteund.

Heel veel code zit reeds in bibliotheken, bv. `digitalWrite()` roept een functie uit een bibliotheek op die het nodige werk achter de schermen doet.

1.3.3 Datatypes

De voornaamste datatypes die gebruikt worden op de Arduino zijn:

boolean : true of false

byte : 0..255

char : -128..127

int : -32768..32767

word : 0..65535

long : -2 miljard .. 2 miljard

float : kommagetallen

Naast de gebruiksvriendelijke datatypes die meestal gebruikt worden op de Arduino bestaan er ook cross-platform datatypes die op elk platform gelijk zijn in grootte. Een word kan bv. op 32-bits systeem een andere grootte hebben.

Een kort overzicht van de cross-platform datatypes:

uint8_t : unsigned integer van 8-bits (gelijk aan byte op de Arduino UNO)

int8_t : signed integer van 8-bits (gelijk aan char op de Arduino UNO)

uint16_t : unsigned integer van 16-bits

int16_t : signed integer van 16-bits (gelijk aan int op de Arduino UNO)

uint32_t : unsigned integer van 32-bits

int32_t : signed integer van 32-bits (gelijk aan long op de Arduino UNO)

1.3.4 Eerste programma: blink

Dit is het eenvoudigste voorbeeld dat meegeleverd wordt met de Arduino IDE. Het toont de basiswerking van het bordje.

Op de Arduino UNO is er één LED aanwezig die kan aangestuurd worden via pin 13. In de code hieronder wordt deze pin als output gedefinieerd. Vervolgens wordt afwisselend de output hoog en laag gemaakt met een tussentijd van één seconde. Hierdoor zal de LED dus afwisselend aan-en uitgaan.

```
1 #include <Arduino.h>
2
3     // wordt eenmalig uitgevoerd
4 void setup() {
5     pinMode(13, OUTPUT);    // pin 13 output
6 }
7
8 // wordt oneindig herhaald
9 void loop() {
10    digitalWrite(13, HIGH);      // LED aan
11    delay(1000);                // wacht 1s
12    digitalWrite(13, LOW);       // LED uit
13    delay(1000);                // wacht 1s
14 }
```

1.3.5 setup() en loop()

In voorgaand voorbeeld kon je reeds zien dat een Arduino programma of sketch opgebouwd is uit twee delen, de functies `setup()` en `loop()`.

`setup()`

Deze functie wordt éénmalig uitgevoerd in het begin van het programma. Dit is ideaal om instellingen juist te zetten of taken uit te voeren die éénmalig moet gebeuren.

`loop()`

Deze functie wordt na `setup()` uitgevoerd en wordt telkens herhaald. De functie wordt onderbroken door interrupts (zie later).

Werking

Intern worden beide functies aangeroepen door een hoofdprogramma, dit is echter niet zichtbaar in de IDE:

```
1 int main()
2 {
3     setup();
4     while(true)
5     {
6         loop();
7         serialEventRun();
8     }
9     return 0;
10 }
```

1.3.6 Seriële output

Om waarden van variabelen tijdens het uitvoeren van een programma te bekijken kunnen we gebruik maken van de seriële monitor, dit is een onderdeel van PlatformIO.

Bij de Arduino is er een virtuele seriële poort aanwezig over USB. Dit laat heel gemakkelijk communicatie toe tussen de PC en de Arduino. Die seriële poort wordt gebruikt om enerzijds een programma up te laden op het bordje en anderzijds voor de communicatie.

De listing toont een voorbeeld van het gebruik van de seriële poort. De exacte werking komt nog uitvoerig aan bod in een later hoofdstuk.

```
1 void setup() {  
2     Serial.begin(9600);      // poort openen  
3     Serial.print("Hallo ");  // schrijven  
4     Serial.println("wereld");  
5 }  
6  
7 void loop() {  
8 }
```

De seriële monitor kan in PlatformIO geopend worden door op het icoontje in de vorm van een stekker links onderaan te klikken. De uitvoer van de seriële monitor komt in het Terminalvenster terecht onderaan. Standaard staat de snelheid op een baudrate op 9600. Dit kan veranderd worden in het platformio.ini-bestand dat bij het project hoort. Zie de listing als voorbeeld.

```
1 [env:uno]  
2 platform = atmelavr  
3 board = uno  
4 framework = arduino  
5 monitor_speed = 19200
```

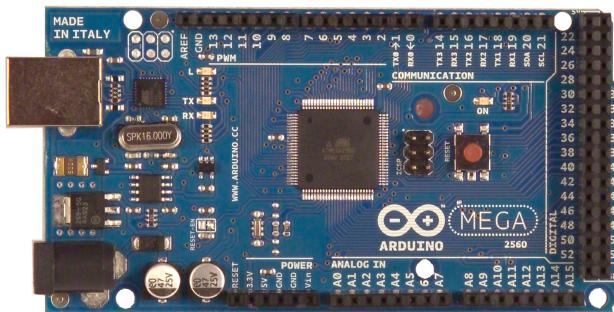
Overzicht hardware

2.1 Soorten bordjes

De Arduino Uno is het basisbordje dat voor veel zaken zal volstaan. Er bestaan heel veel varianten die kunnen variëren in grootte (bv. Micro en Mega), in toepasbaarheid (bv. Lilypad) en een andere processor bevatten (bv. Due). Hieronder volgt een zeer beperkt overzicht. Naast de Arduino's zijn er o.a. ook de ESP gebaseerde bordjes. Deze zijn meestal krachtiger maar ook complexer in gebruik.

2.1.1 Arduino Mega

De Mega (figuur 2.1) bevat veel meer pinnen en meer geheugen dan de UNO. Het bord is dan ook bedoeld voor de grotere projecten. Shiels die passen op de UNO passen ook op de Mega.



Figuur 2.1: Arduino Mega

2.1.2 Arduino Micro

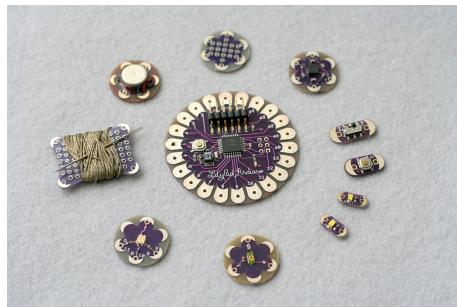
Als de beschikbare plaats om het bordje in te bouwen beperkt is, dan is de Micro (figuur 2.2) de ideale oplossing. Dit bordje is veel kleiner dan de UNO maar bevat ongeveer evenveel pinnen en geheugen als de UNO, er is zelfs een USB-aansluiting aanwezig. Hier passen echter geen shields op.



Figuur 2.2: Arduino Micro

2.1.3 Arduino Lilypad

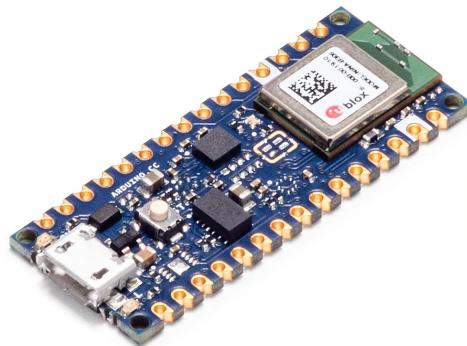
Nog kleiner en vooral bedoeld om in kledij in te bouwen is de Lilypad (figuur 2.3). Draagbaarheid staat hierbij centraal.



Figuur 2.3: Arduino Lilypad

2.1.4 Arduino Nano 33 BLE

Dit bordje bevat een krachtige 32-bit ARM processor en vorm compatibel met de oudere Arduino Nano. De werkspanning is 3,3V en zoals de naam doet vermoeden is er ook Bluetooth BLE ondersteuning (figuur 2.4).



Figuur 2.4: Arduino Nano 33 BLE

2.1.5 ESP8266

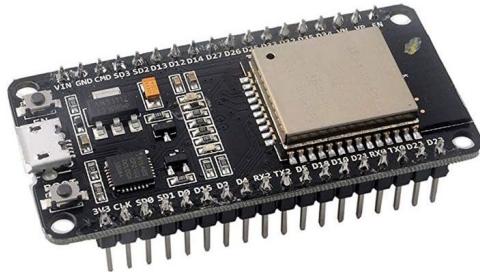
Dit bordje is vooral interessant voor zijn ingebouwde WiFi mogelijkheden. Het wordt gebruikt waarbij een netwerkverbinding cruciaal is. Het bordje is ook krachtig genoeg om te gebruiken met MicroPython. Er bestaan verschillende varianten gebaseerd op de ESP8266 processor, de meest gebruikte variant is de NodeMCU (figuur 2.5).



Figuur 2.5: NodeMCU ESP8266

2.1.6 ESP32

De ESP32 (figuur 2.6) bevat naast WiFi ook Bluetooth mogelijkheden en is nog krachtiger dan de ESP8266. Er bestaan terug verschillende varianten.



Figuur 2.6: ESP32

2.2 Shields

Standaard bevat het Arduino bordje weinig mogelijkheden. Er zijn geen bv. geen knoppen of display aanwezig. Er is ook maar één LED aanwezig die kan gebruikt worden.

Gelukkig kan het bordje zeer eenvoudig uitgebreid worden met een *shield*. Dit is een extra bordje dat bovenop de Arduino geplaatst wordt en op een eenvoudige manier hardware toevoegt.

Door de populariteit van Arduino zijn er heel veel shields te verkrijgen. De shields zijn normaal ontworpen voor de UNO en passen op de meeste andere Arduino's, bij de Due moet wel op de werkspanning gelet worden.

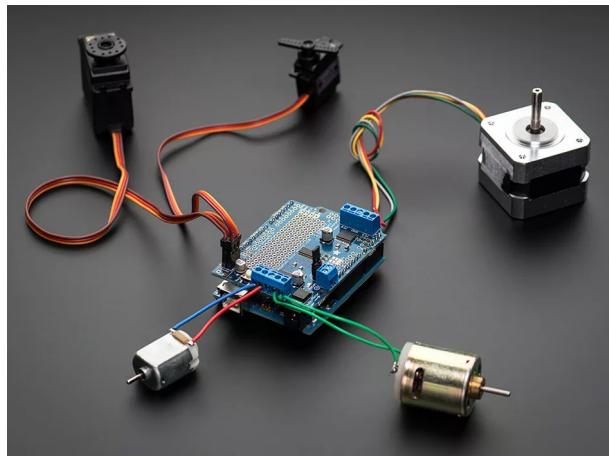
Een aantal grote fabrikanten van shields zijn Arduino, Adafruit, Sparkfun en Seeedstudio. Goedkope alternatieven kunnen ook gevonden worden op de Chinese markt, bv. bij DX en AliExpress. De kwaliteit laat soms te wensen over, goed testen is dus altijd de boodschap om verborgen verbreken te ontdekken.

2.2.1 Motor shield

Het motor shield (figuur 6.16) bevat een aantal H-bruggen die toelaten om de stroom te versterken en de polariteit op de uitgang te wisselen. Dit is ideaal om een DC-motor (of stappenmotor) van stroom te voorzien en de draaizin eenvoudig te kunnen wijzigen. Er zijn ook een aantal aansluitingen voorzien voor servo-motors.

2.2.2 GPRS shield

Het GPRS shield (figuur 2.8) laat toe om gegevens via het GSM-netwerk te verzenden. Dit is dus ideaal om bv. een sensor ten valde uit te lezen en de gegevens naar een centrale server of de cloud door te sturen. Onderaan het shield is er een slot voor een SIM-kaart.



Figuur 2.7: Motor shield

2.2.3 Ethernet shield

Wanneer een ethernet verbinding ter beschikking is kan de Arduino met behulp van een ethernet shield (figuur 2.9) aangesloten worden op een netwerk. Daardoor kan de Arduino bv. fungeren als eenvoudige webserver of gegevens opvragen van het internet. Er is een SD-kaart slot aanwezig om grote hoeveelheden data (zoals HTML-pagina's) te kunnen bewaren.

2.2.4 SD-card shield

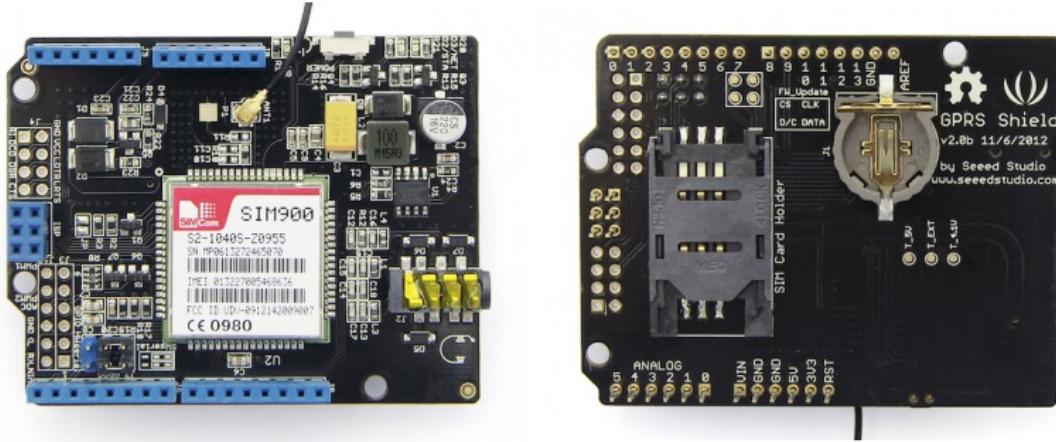
Voor eenvoudige meetstations waarbij het vooral de bedoeling is om één of meerdere sensoren uit te lezen kan dit shield (figuur 2.10) gebruikt worden. Het laat toe om de data op te slaan op een SD-kaart. Verder is er vrije (soldeer) ruimte voorzien om sensoren of andere componenten op het bordje te solderen.

2.2.5 Display shield

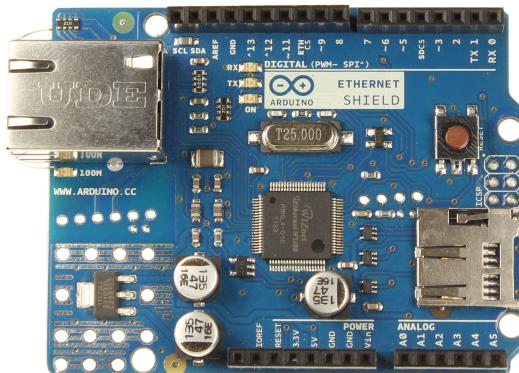
Gegevens visualiseren kan handig en noodzakelijk zijn, denk bv. aan een koffieautomaat. Het display-shield (figuur 2.11) bevat een LCD die twee lijnen tekst kan tonen en energiezuinig is. Daarnaast zijn een aantal knoppen aanwezig om het bordje te kunnen bedienen.

2.2.6 Clock shield

In het labo worden alle basiszaken aangeleerd, daarbij is input (knoppen, sensoren), en output (LED's, display, buzzer) nodig. Het Clock shield (figuur 2.12) bevat al het voorgaande en



Figuur 2.8: GPRS shield

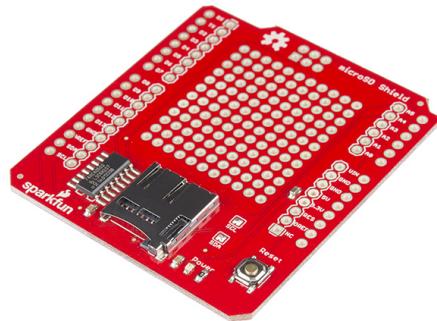


Figuur 2.9: Ethernet shield

bovendien ook een realtime klok (RTC) die de tijd kan bijhouden. Dit shield is oorspronkelijk bedoeld om een klok te maken maar heeft ook een hoge educatieve waarde.

Het Clock shield is ontwikkeld door de firma Seeedstudio en wordt door hun Start shield of Tick Tock shield genoemd. Hun versie is zelf te solderen. De website van Seeedstudio bevat alle info en de links naar de nodige bibliotheken. Het is te verkrijgen bij de meeste elektronica winkels.

Er bestaan ook clonen van dit shield, bv. het Clock shield van o.a. Catalex. Dit is reeds gesoldeerd en kan direct gebruikt worden. Een nadeel is dat het enkel te verkrijgen is via bv. dx.com en de verzendtijd een aantal weken in beslag neemt. Let hierbij ook op voor eventuele douanekosten.

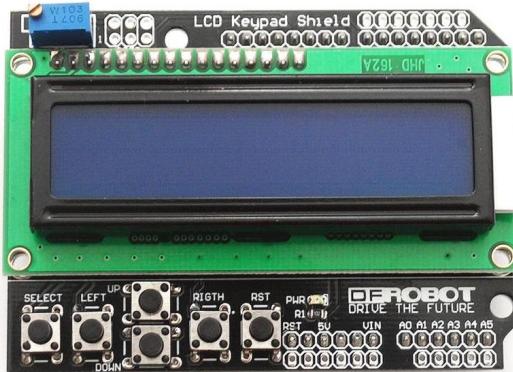


Figuur 2.10: SD-card shield

2.3 Features Arduino Uno en ESP8266

Hieronder volgt nog een overzicht van de mogelijkheden van de Arduino UNO.

Microcontroller	Arduino Uno	ESP8266 NodeMCU
Voedingsspanning	5V	3.3V
Bits	8	32
Ingangsspanning (aanbevolen)	7-12V	5V
Ingangsspanning (uitersten)	6-20V	5-12V
Digitale I/O-pinnen	14 (waarvan 6 PWM)	17
Analoge input pinnen	6	1
DC stroom per I/O-pin	40mA	12mA
Flash geheugen	32kB	16MB
SRAM geheugen	2kB	160kB
EEPROM geheugen	1kB	4kB
Kloksnelheid	16MHz	80MHz



Figuur 2.11: Display shield



Figuur 2.12: Clock shield

3

Datavoorstellingen

Aangezien een computer een digitaal elektronisch systeem is werkt het enkel met enen en nullen (men noemt dit bits). Dit impliceert dat zowel de instructies (code) als de gegevens (data) die zullen verwerkt worden als lange slerten (men spreekt van bitpatronen) nullen en enen worden opgeslagen in hetzelfde geheugen. Dit stelt ons - en zeker de processor - voor een uitdaging: „Hoe kunnen we dan nog het onderscheid maken tussen instructies en gegevens?” en ook „stel dat een bepaald bitpatroon een gegeven voorstelt, hoe kan een processor dan weten of dit een natuurlijk-, geheel-, reëel getal, karakterteken,... betreft?”

3.1 De BIT

Zoals reeds gezegd wordt alle informatie in een computer voorgesteld als patronen van bits. Een BIT (Binary digit) kan enkel de waarde 0 of 1 aannemen. Soms zegt men dat de bit waar/true/on is wanneer de bit de waarde 1 heeft en onwaar/false/off wanneer de bit de waarde 0 heeft. De keuze voor de BIT als elementaire informatiecel heeft te maken met de technologie waarop de computer is gebaseerd. De digitale elektronica is bij uitstek geschikt om te werken met aan/uit logica waarbij een spanning van 0 Volt staat voor een 0, en een spanning van 5 Volt een 1 vertegenwoordigt¹.

3.2 Talstelsels

3.2.1 Decimaal

We gebruiken het decimale talstelsel sinds onze kindertijd zodat dit talstelsel als evident wordt beschouwd. Wanneer je bijvoorbeeld het getal 156,3 ziet, dan hoef je (hopelijk) niet na te

¹tegenwoordig wordt een lagere spanning dan 5 Volt gebruikt om minder vermogen te verbruiken

denken over de „grootte of de waarde” van dit getal. Wanneer je dit getal ontleedt, krijg je:

$$1 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0 + 3 \cdot 10^{-1} = 100 + 50 + 6 + 0,3 \quad (3.1)$$

De positie komt dus overeen met de plaats van het cijfer ten opzichte van de decimale punt. Het eerste cijfer links van de decimale punt staat op positie 0 (men spreekt over de eenheden omdat de hiermee overeenkomende macht van 10 gelijk is aan één). Het volgende cijfer staat op positie 1 (men noemt dit de tientallen omdat $10^1 = 10_d$). Het derde cijfer links van de decimale punt staat op positie 2 (de honderdtallen omdat $10^2 = 100_d$).

3.2.2 Binair

Zoals reeds aangehaald werkt een digitaal elektronisch apparaat met 2 waarden: 0 en 1 (of toestanden). Het ligt dus voor de hand een nummersysteem te ontwikkelen dat hierop is gebaseerd: het binaire talstelsel. Ook in het binaire talstelsel is het gewicht van elk cijfer [0..1] afhankelijk van de positie van dat cijfer. Het binair getal 110010101_b komt overeen met:

$$1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 256 + 128 + 16 + 4 + 1 = 405_d \quad (3.2)$$

De voornaamste veelvouden van twee zijn:

$$\begin{aligned} 100000_b &= 32_d &= 2^5 \\ 10000_b &= 16_d &= 2^4 \\ 1000_b &= 8_d &= 2^3 \\ 100_b &= 4_d &= 2^2 \\ 10_b &= 2_d &= 2^1 \\ 1_b &= 1_d &= 2^0 \end{aligned} \quad (3.3)$$

Om een decimaal getal terug om te zetten naar een binair getal, delen we het getal telkens door twee. De rest, in omgekeerde volgorde, is dan ons binair getal:

$$\begin{aligned}
 405/2 &= 202 \text{ rest } 1 \\
 202/2 &= 101 \text{ rest } 0 \\
 101/2 &= 50 \text{ rest } 1 \\
 50/2 &= 25 \text{ rest } 0 \\
 25/2 &= 12 \text{ rest } 1 \\
 12/2 &= 6 \text{ rest } 0 \\
 6/2 &= 3 \text{ rest } 0 \\
 3/2 &= 1 \text{ rest } 1 \\
 1/2 &= 0 \text{ rest } 1
 \end{aligned} \tag{3.4}$$

Dit vormt dus uiteindelijk 110010101_b .

3.2.3 Hexadecimaal

Omdat de binaire schrijfwijze lastig is in gebruik en we veelvuldig zullen werken met 16-bit adressen en 16-bit data maken we gebruik van een verkorte notatie voor bitpatronen die een veelvoud zijn van 4-bits. Een groepje van 4 bits noemt men een nibble.

Binair	Decimaal	Hexadecimaal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Stel dat je een 16-bit binair getal in hexadecimale notatie moet schrijven, dan verdeel je dit getal eerst in nibbles om vervolgens elke nibble naar zijn overeenkomstig hexadecimaal teken te converteren:

$$1010001111100010_b = 1010\ 0011\ 1110\ 0010_b = A3E2_h \quad (3.5)$$

Van hexadecimaal naar binair kan eenvoudig door cijfer per cijfer in binair formaat te schrijven:

$$3FA_h = 0011\ 1111\ 1010_b = 1111111010_b \quad (3.6)$$

3.3 Dataorganisatie

3.3.1 Bit

De kleinste eenheid van data in een binaire computer is een bit. Een bit kan slechts 2 verschillende waarden (typisch 0 en 1) aannemen, vandaar dat de meeste datatypes bestaan uit meerdere bits. Op deze manier kunnen meerdere waarden worden voorgesteld, bvb met een byte kunnen 256 waarden worden voorgesteld. Afhankelijk van hoeveel bits men samen neemt spreekt men over nibbles (4), bytes (8), words (16), double words (32), ...

3.3.2 Nibble

Een nibble is een verzameling van 4 bits. Hiermee kunnen 16 verschillende waarden worden voorgesteld. Nibbles worden vaak gebruikt bij de conversie tussen een binaire en hexadecimale voorstelling van informatie (instructies, adressen, data).

3.3.3 Byte

De belangrijkste datastructuur in de 80x86 microprocessoren is de byte. Een byte bestaat uit 8 bits en is het kleinst adreseerbaar data element in de 80x86 processorfamilie. Zowel geheugen- als IO-adressen zijn byte adressen, dwz dat met elk adres een bytelocatie overeenstemt. De grootte van opslagmedia wordt uitgedrukt in bytes.

3.3.4 Word

Een word bestaat uit twee bytes of 16 bits. Dit betekent dat een word een bereik heeft van 0 tot 65535.

3.4 Karaktercodes

Elke computer gebruikt een verzameling karakters. Als absoluut minimum bestaat deze verzameling uit 26 hoofdletters, 10 cijfers en enkele leestekens, zoals spatie, punt, komma en carriage

return. In uitgebreidere karakterverzamelingen komen zowel de hoofdletters als de speciale tekens voor, de 10 cijfers, een groot scala aan leestekens, een collecte speciale karakters die nuttig zijn voor wiskundige en zakelijke teksten, en soms zelfs Griekse letters.

Om deze karakters binnen de computer binnen te brengen wordt aan elk karakter een nummer toegekend: $a=1$, $b=2$, ... De afbeelding op gehele getallen wordt een **karaktercode** genoemd. Tegenwoordig gebruiken computers een 6-, 7-, 8- of 9-bits code. In een 6-bits code zijn maar $2^6 = 64$ karakters mogelijk, namelijk 26 letters, 10 cijfers en 28 andere karakters, voornamelijk leestekens en wiskundige symbolen.

3.4.1 ASCII

Voor veel toepassingen zijn 64 karakters niet voldoende, in welk geval een 7- of 8-bits code moet worden gebruikt. Met een 7-bits karaktercode zijn er 128 karakters mogelijk. Een veel gebruikte 7-bits code is ASCII (American Standard Code for Information Interchange), zie de tabel in figuur 3.1.

De meeste computers werken echter met 8-bits codes (bytes). Het 8e bit werd traditioneel gebruikt voor foutdetecterende codes (met name: een pariteitsbit) en andere apparaatspecifieke toepassingen. Omdat in landen buiten de Verenigde Staten behoefte was aan extra tekens (zoals andere letters, letters met accenten, valutasymbolen) werd het aantal mogelijke tekens vergroot door ook het 8e bit te gebruiken (tweemaal zoveel, namelijk $2^8 = 256$). Ook worden veel stuurcodes niet meer voor hun oorspronkelijke doel gebruikt en zijn dus voor extra tekens beschikbaar. Zo ontstonden de extended ASCII-tekenverzamelingen. Hierbij zijn echter verschillen ontstaan tussen de tekenverzamelingen van verschillende talen. Extended ASCII is niet één bepaalde standaard, maar een verzamelnaam voor de verschillende tekenrepresentaties die de 95 ASCII-teken als basis hebben.

3.4.2 Unicode

Unicode is een internationale standaard voor het coderen van tekst en andere grafische tekens, met als doel alle karakters die nodig zijn om elke geschreven mensentaal op te schrijven, eenduidig te ordenen. In tegenstelling tot ASCII, dat alleen de standaard letters uit het Engels zonder accenten kent, en het Latin-1 met accenten, dat veel wordt gebruikt in West-Europa, omvat Unicode meerdere schriften van meerdere, ook wanneer die verschillende talen in één enkel document worden gebruikt. In beginsel wordt voor elk schriftteken een apart nummer gebruikt, maar wordt geen onderscheid gemaakt tussen de manier waarop dat teken wordt geschreven.

De eerste versie van Unicode bood ruimte aan 65.536 ($= 2^{16}$) tekens met een eigen nummer, dus aan even zoveel verschillende lettertekens. Daarvan waren er aanvankelijk zo'n 40.000 bezet. 65536 tekens is echter te weinig om tekens in alle wereldtalen te kunnen indelen. Zo bestaat

het Chinees op zichzelf al uit zo'n 25.000 tekens. Daarom is het aantal mogelijke lettertekens in de Unicode-standaard later uitgebreid naar zo'n één miljoen.

Het is niet mogelijk elk Unicode letterteken te coderen in een enkele byte, of zelfs in twee. De Unicode-standaard lost dit probleem niet op: de zogenaamde codering maakt geen deel uit van de huidige versie van de Unicode-standaard. Wel zijn er standaarden bedacht om de Unicode-karaktersets in series bytes op te kunnen slaan. De meest voor de hand liggende methoden zijn UCS-2 of UCS-4, waarbij twee respectievelijk 4 bytes per teken gebruikt worden. Omdat dit bij teksten in ons welbekende Romeins alfabet ruimteverspillend werkt en bovendien voor iedereen incompatibel met het verleden zijn coderingen bedacht die hier minder last van hebben, dit zijn UTF-8 en UTF-16; in een volgende versie van de Unicode-standaard worden deze coderingen mogelijk opgenomen. Ook bestaat er de zeer stringente UTF-7 codering, die zeker als een losse standaard zal blijven bestaan, maar slechts beperkte toepassingen kent. UTF-8 heeft als groot voordeel dat het 100 procent compatibel is met 7-bit ASCII, alle andere karakters worden anders opgeslagen dan in andere coderingen.

Oefeningen

Oefening 1

Ontleed telkens de volgende decimale getallen:

- 4785
- 86,7
- 0,3654

Oefening 2

Bereken de respectievelijke decimale waarde:

- 11_b
- 100_b
- 1010_b
- 1100_b
- 101010_b
- 11110000_b

Oefening 3

Bereken de respectievelijke binaire waarde:

- 3_d
- 7_d
- 4_d
- 14_d
- 33_d
- 523_d
- 302_d

Oefening 4

Vul onderstaande tabel aan:

Binair	Decimaal	Hexadecimaal
	16	
		FF
11011		
		17
	84	
1010110		
		A3E
	317	
		9F9F

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	..	65	41	A	97	61	a
^B	2	02		STX	34	22	#	66	42	B	98	62	b
^C	3	03		ETX	35	23	\$	67	43	C	99	63	c
^D	4	04		EOT	36	24	%	68	44	D	100	64	d
^E	5	05		ENQ	37	25	&	69	45	E	101	65	e
^F	6	06		ACK	38	26	,	70	46	F	102	66	f
^G	7	07		BEL	39	27	(71	47	G	103	67	g
^H	8	08		BS	40	28)	72	48	H	104	68	h
^I	9	09		HT	41	29	*	73	49	I	105	69	i
^J	10	0A		LF	42	2A	+	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	-	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[27	1B		ESC	59	3B	;	91	5B	[123	7B	{
^`	28	1C		FS	60	3C	<	92	5C	\	124	7C	
^]	29	1D		GS	61	3D	=	93	5D]	125	7D	}
^~	30	1E	▲	RS	62	3E	>	94	5E	~	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	¤	127	7F	*¤

Figuur 3.1: De ASCII-karaktercode

4

Wiskundige en logische functies

4.1 Wiskundige functies

Onder wiskundige functies verstaan we het optellen, aftrekken, vermenigvuldigen en delen van getallen. Bij het optellen en vermenigvuldigen wordt het resultaat groter dan de brongetallen, bij het aftrekken en vermenigvuldigen kleiner. Hiermee moet rekening gehouden worden bij de gebruikte datatypes.

4.1.1 Optellen

Het optellen van twee getallen levert meestal geen problemen op, wel is het resultaat meestal groter dan de brongetallen. Zorg er dus voor dat dit in het datatype kan qua grootte en let ook op het teken (signed en unsigned).

Bij eventuele overflow wordt opnieuw begonnen met tellen, zie de optellingen in het voorbeeld.

```
1 int h = 32767; // signed
2 int i = h + 1; // i = -32768
3
4 byte a = 0x10; // unsigned
5 byte b = 0xFF;
6 byte c = a + b; // c = 0x0F
```

4.1.2 Aftrekken

Als het resultaat positief is is er geen probleem. Bij een negatieve uitkomst wordt gebruik gemaakt van het 2's complement. Dit wordt uitvoerig besproken in de lessen digitaaltechniek.

```

1 byte a = 0x10;
2 byte b = 0x20;
3 byte c = a - b; // c = 0xF0
4 long d = a - b; // d = 0xFFFFFFFF0

```

Defening

Wat komt er op het scherm als onderstaande code uitgevoerd wordt?

```

1 byte a = 0x80
2 byte b = 0x90
3 Serial.println(a+b, HEX);
4 byte c = a-b;
5 Serial.println(a-b, HEX);
6 Serial.println(c, HEX);

```

4.1.3 Vermenigvuldigen

Bij gehele getallen is het resultaat groter dan de brongetallen, zorg er dus terug voor dat dit in het gekozen datatype kan.

```

1 byte a = 0x20;
2 byte b = 0x10;
3 int c = a*b; // c = 0x200;
4 byte d = a*b; // d = 0x00;

```

Defening

Wat komt er op het scherm als onderstaande code uitgevoerd wordt?

```

1 byte a = 12; // DEC
2 byte b = B10; // BIN
3 Serial.println(a*b, HEX);
4
5 int c = 0x1234;
6 int d = 0x100;
7 Serial.println(c*d, HEX);

```

4.1.4 Deling

De uitkomst bij een deling is kleiner dan één van de twee brongetallen. Naast de deling zelf (/) is er ook de rest van de deling (%).

Let op bij een deling van gehele getallen, de uitkomst is in dat geval ook een geheel getal en geen kommagetal. Dit wordt aangetoond in het voorbeeld.

```

1 byte a = 0x20;
2 byte b = 0x10;
3 byte c = a/b; // c = 0x02
4 byte d = b/a; // d = 0x00
5 byte e = b%a; // e = 0x10
6
7 Serial.println(9/2); // uitkomst 4
8
9 float f = 0x10;
10 float g = 0x12;
11 Serial.println(f/g); // uitkomst 0.89d

```

Defening

Wat komt er op het scherm als onderstaande code uitgevoerd wordt?

```

1 int b = 0x4445;
2 int a = 0x100;
3 Serial.println(b/a,HEX);
4
5 Serial.println(float(9/2));
6 Serial.println(9/float(2));

```

4.2 Logische functies

Logische functies zijn functies waarvan de brongetallen bitgewijs verwerkt worden en het resultaat ook uit een aaneenschakeling van bits bestaat. De bits worden positie per positie met elkaar verwerkt en het resultaat komt op dezelfde plaats in het doelgetal. Logische functies werken steeds met 0 en 1 als cijfers.

4.2.1 AND

De logische AND bitwise instructie is (`&`), dit mag niet verward worden met de booleaanse instructie (`&&`) die bv. in een if-structuur gebruikt wordt.

De functie `&` voert dus bitsgewijs de AND-functie uit. De waarheidstabel is hierbij:

```
0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1
```

De AND-functie wordt gebruikt om bits te **clearen** (maskerading).

Voorbeeld

```
1 unsigned int e=0xF0F0; //1111 0000 1111 0000
2 unsigned int f=0x1234; //0001 0010 0011 0100
3 unsigned int g=e & f; //0001 0000 0011 0000 (= 1030h)
4
5 byte h=23;    // 0001 0111b (= 17h)
6 byte i=14;    // 0000 1110b (= 0Eh)
7 byte j= h&i; // 0000 0110b (= 06h)
```

4.2.2 OR

De functie `|` voert bitsgewijs de OR-functie uit. De waarheidstabel is hierbij:

```
0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1
```

De OR-functie wordt gebruikt om bits te **setten**.

Voorbeeld

```
1 unsigned int e=0xF0F0; //1111 0000 1111 0000
2 unsigned int f=0x1234; //0001 0010 0011 0100
3 unsigned int g=e | f; //1111 0010 1111 0100 (= F2F4h)
```

```

4
5 byte h=23;    // 0001 0111b (= 17h)
6 byte i=14;    // 0000 1110b (= 0Eh)
7 byte j= h|i; // 0001 1111b (= 1Fh)

```

4.2.3 XOR

De functie `^` voert bitsgewijs de XOR (eXclusieve OR) functie uit. De waarheidstabel is hierbij:

```

0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
1 xor 1 = 0

```

De XOR-functie wordt gebruikt om bits te **inverteren**.

Voorbeeld

```

1 unsigned int e=0xF0F0; //1111 0000 1111 0000
2 unsigned int f=0x1234; //0001 0010 0011 0100
3 unsigned int g=e ^ f; //1110 0010 1100 0100 (= E2C4h)
4
5 byte h=23;    // 0001 0111b (= 17h)
6 byte i=14;    // 0000 1110b (= 0Eh)
7 byte j= h^i; // 0001 1001b (= 19h)

```

Oefening

Bepaal telkens de hex-waarde na de AND-, OR- en XOR-instructie:

```

1 unsigned int a=0x1234;
2 unsigned int b=0xFFFF;
3 unsigned int c=0x00F0;
4 unsigned int d=0xF00F;
5 unsigned int e=a&b;
6 unsigned int f=e|c;
7 unsigned int g=f^d;

```

4.2.4 NOT

Tenslotte is er nog de NOT-functie, de functie `~` voert bitsgewijs de NOT functie uit. Dit mag niet verward worden met de booleaanse functie `!`.

Hierbij is de waarheidstabel:

```
not 1 = 0
not 0 = 1
```

Merk dus op dat een 0 een 1 wordt en omgekeerd, ze worden dus geïnverteerd.

Voorbeeld

```
1 unsigned int e=0xF0F0; //1111 0000 1111 0000
2 unsigned int g=~e;      //0000 1111 0000 1111 (=0F0Fh)
3
4 byte h=23;    // 0001 0111b (= 17h)
5 byte j=~h;    // 1110 1000b (= E8h)
```

4.2.5 Shiften

Bits kunnen in een getal ook van plaats verschoven worden (naar links of rechts), dit noemt men shiften.

Naar links shiften kan met `<<` en naar rechts met `>>`. Eénmaal shiften naar respectievelijk links of rechts voert een vermenigvuldiging of deling met 2 uit.

```
1 byte g = B00000001;
2 byte h = g << 3; // h=00001000 = 8d
3 byte i = g >> 2; // i=00000000
4
5 byte d = 24;
6 byte e = d >> 3; // e=24/8=3d
7 byte f = d << 1; // f=24*2=48d
```

4.2.6 U en L notatie

In sommige gevallen willen we constanten definiëren zoals een getal. Standaard wordt dit in een integer bewaard, bij manipulatie van het getal kan het dus niet kleiner of groter worden dan respectievelijk -32768 en +32767.

Dit gedrag kan gewijzigd worden zodat de constante een ander datatype toegewezen krijgt. 'U' zorgt voor een unsigned datatype en 'L' voor het datatype *long*. Zo zal `1UL` een constante definiëren met de waarde 1 en dit bewaren in een *unsigned long*.

4.2.7 Bit manipulaties

Bij de Arduino bestaan er ook eenvoudige functies die rechtstreeks de waarde van een bit weergeven of veranderen op een bepaalde positie in het getal.

Bits lezen en schrijven kan met `bitRead()` en `bitWrite()`. Bits setten en clearen kan met `bitSet()` en `bitClear()`.

```

1 byte a = B01010101;
2 Serial.println(bitRead(a,6)); // 1
3 bitWrite(a,6,0);
4 Serial.println(a,BIN); // 00010101
5 bitSet(a,6);
6 Serial.println(a,BIN); // 01010101
7 bitClear(a,6);
8 Serial.println(a,BIN); // 00010101

```

Deze functies worden niet door alle microcontrollers ondersteund, het is dus aan te raden deze niet te gebruiken wanneer je code ontwikkeld voor meerdere types microcontrollers.

De functies kunnen ook als volgt geïmplementeerd worden:

- `bitRead(value,bit)` wordt `((value) >> (bit)) & 0x01`
- `bitSet` wordt `(value) |= (1UL << (bit))`
- `bitClear` wordt `(value) &= ~(1UL << (bit))`
- `bitWrite(value,bit,bitvalue)` wordt
 `bitvalue ? bitSet(value, bit) : bitClear(value, bit)`

Een aantal voorbeeld berekeningen:

We willen bit 6 van byte a lezen (equivalent van `bitRead(a,6)`):

```

byte a          = B11110000
byte b = a >> 6 = B00000011 (bit 6 staat nu helemaal rechts)
byte c = b & 0x01 = B00000001 (enkel bit 0 overhouden)

```

We willen bit 6 van byte a op nul zetten (equivalent van `bitClear(a,6)`):

```
byte a      =          B1111 0000
1UL         = B0000 0000 0000 0000 0000 0000 0001 (unsigned long=32-bits)
1UL << 6    = B0000 0000 0000 0000 0000 0100 0000 (6 plaatsen naar links)
~(1UL << 6) = B1111 1111 1111 1111 1111 1011 1111 (inverse)
a &= ~(1UL << 6) =          B1011 0000 (bit 6 wordt gewist)
```

5

Getalnotatie en strings

5.1 Getalnotatie

Getallen kunnen decimaal, hexadecimaal of binair afgebeeld worden. De waarde van het getal blijft echter steeds gelijk, ongeacht de notatie!

Intern wordt een getal steeds binair bewaard op een microcontroller of een computer (dus een verzameling van enen en nullen).

Er moet worden opgelet met de omzetting van en naar ASCII.

5.1.1 Versturen

Serial.write()

Deze functie stuurt via de seriële poort een getal als bytewaarde door, de cijfers worden dus niet afzonderlijk verstuurd.

```
1 void loop(){
2     Serial.write(65); // stuur een byte met waarde 65 (ascii- ↴
3     ↴waarde van "A")
4     Serial.write(0x0A); // LF
5     Serial.write(0x0D); // CR
6     int bytesSent = Serial.write("hallo"); //stuur de string ↴
7     ↴"hallo"
8 }
```

Het bovenstaand voorbeeld toont in de seriële monitor:

A

```
hallo
```

```
Serial.print()
```

Deze functie daarentegen zet het getal om naar leesbare tekst, het wordt dus gesplitst in cijfers en omgezet naar de respectievelijke ASCII-tekens.

Het splitsen in cijfers kan zowel decimaal, hexadecimaal of binair gebeuren.

```

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

```

Het bovenstaand voorbeeld toont in de seriële monitor:

```
65
65
41
1000001
```

Merk op dat eventuele voorgaande nullen niet getoond worden.

5.1.2 Omzetting naar getal

Bij geavanceerde meettoestellen kunnen de gemeten waarden opgevraagd worden via de computer. Deze waarden worden meestal doorgestuurd als één string die dan op een correcte manier in één of meerdere getallen moeten worden omgezet.

De volgende functies kunnen daarbij helpen:

- `toInt()`: omzetting naar integer
- `toFloat()`: omzetting naar kommagetal

Voorwaarde voor deze functies is dat de string moet starten met een cijfer. De conversie zelf stopt bij het einde van het getal, als het volgende karakter dus niet numeriek is.

5.1.3 Voorbeeld

```

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

```

Het bovenstaand voorbeeld toont in de seriële monitor:

65
65.30

5.1.4 Afronden

Standaard toont de functie `Serial.println()` bij een float twee cijfers na de komma. Hierbij wordt er correct afgerond. Dit gedrag kan gewijzigd worden door een extra parameter mee te geven die aanduidt hoeveel cijfers men wenst.

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

```

Het bovenstaand voorbeeld toont in de seriële monitor:

65.37
65.3691
65.4

5.2 Strings

5.2.1 Strings vergelijken

In een string moet soms gezocht worden naar een patroon, een vergelijking is dan nodig. Dit kan met de functie `==` of met `equals()`.

Alfabetisch vergelijken kan met `<` en `>`, let op spaties en speciale karakters. Er wordt steeds via de waarden in de ASCII-tabel gerangschikt.

Verder zijn er nog de functies != (verschillend van) en equalsIgnoreCase (hoofdletterongevoelig).

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");

```

Het bovenstaand voorbeeld toont in de seriële monitor:

```

niet gelijk
gelijk (hoofdletterongevoelig)
1 alfabetisch groter dan 2

```

5.2.2 Substrings

Wanneer in een uitlezing meer dan één (getal)waarde zit moet er gefilterd worden op een deel van de string. Bij de meeste (meet)toestellen is er een vaste lengte per (getal)waarde die aangehouden wordt.

Hiervoor is de functie `substring()` handig. Let er op dat het eerste karakter van een string op de plaats 0 staat.

Wanneer één parameter wordt meegegeven met de functie dan wordt de rest van de string vanaf die positie behouden. Als men twee parameters meegeeft dan wordt de string behouden vanaf de startpositie tot de stoppositie.

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");

```

Het bovenstaand voorbeeld toont in de seriële monitor:

gelijk aan text

5.3 Datatypes

Let op voor foutieve resultaten bij tussenberekeningen. Het datatype van het brongetal bepaalt bij wiskundige bewerkingen het datatype van de uitkomst. Zo is bij een deling het resultaat een integer als de noemer een integer is, ook als is de uitkomst geen geheel getal.

Bij het shiften daarentegen wordt het resultaat (soms ongewild) aangepast aan de uitkomst. Zo zal bij het shiften naar links het resultaat een 16-bits getal zijn als de bron een 8-bits getal is. Dit kan opgevangen worden met behulp van de juiste *typeconversie*.

5.3.1 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);
```

Het bovenstaand voorbeeld toont in de seriële monitor:

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


6

Input en output

6.1 Interne registers

Een voordeel voor de beginnende gebruiker bij Arduino is dat hij weinig moet afweten van de gebruikte processor op het bordje. Bij de UNO is dit de Atmega328.

Om een pin aan te sturen volstaat om de mode in te stellen met `pinMode()` en de pin uit te lezen of aan te sturen met respectievelijk `digitalRead()` en `digitalWrite()`. Dit is natuurlijk heel handig in gebruik en laat ons ook toe om verschillende processors te gebruiken op verschillende Arduino's.

Om meer te controle te hebben over hetgeen er gebeurt op de achtergrond en ook dit proces te begrijpen is het noodzakelijk de aansturing op processorniveau te kennen.

Deze manier van aansturing kan noodzakelijk zijn in de volgende gevallen:

- snelheid: deze aansturing laat een veel schnellere controle van de pinnen toe, we kunnen een 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

6.1.1 Poorten

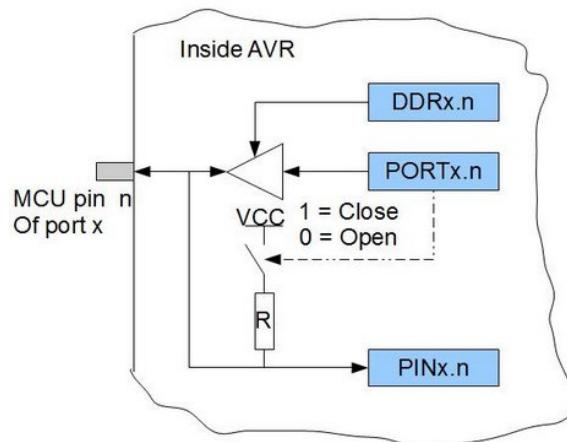
De poorten bij de Atmega328 (en dus de Arduino UNO) zijn onderverdeeld in B, C en D. Elke poort bevat een aantal pinnen:

- B: digitale pinnen 8 tot 13
- C: analoge pinnen
- D: digitale pinnen 0 tot 7

Het Arduino UNO pinout diagramma in bijlage toont de verschillende poorten (PB0-5, PC0-5 en PD0-7). Zo kan je bijvoorbeeld zien dat PB4 overeenkomt met digitale pin 12.

6.1.2 DDR-register

Elke poort bevat drie registers die het mogelijk maken om een poort in stellen (DDR), uit te lezen (PIN) of aan te sturen (PORT) (figuur 6.1).



Figuur 6.1: Pull-up weerstand

De mode (lezen of schrijven) wordt voor alle pinnen van een poort ingesteld in het DDR-register. Dit wil dus zeggen dat we alle pinnen tegelijk moeten instellen. Met de logische OR- en AND-functies kunnen we echter ook één bit manipuleren en dus de mode van één pin veranderen.

6.1.3 PORT-register

Als de pin in schrijfmodus staat kan het aansturen gebeuren via het PORT-register. Door een bit hoog te maken in een PORT-register wordt de daarmee verbonden pin hoog gemaakt. Ook hier stuurt een PORT-register een volledige poort aan, we moeten dus terug met de OR- en AND-functie werken om één pin aan te sturen.

Een poort aansturen

Het voorbeeld toont hoe poort D rechtstreeks gebruikt wordt via de interne registers.

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

```

3   DDRD = B11111111; // zet PORTD (digital 7-0) als outputs
4 }
5
6 void loop()
7 {
8   PORTD = B11110000; // digital 7-4 HIGH, digital 3-0 LOW
9   delay(1000);
10  PORTD = B00001111; // digital 7-4 LOW, digital 3-0 HIGH
11  delay(1000);
12 }
```

Als we de beschikbare Arduino-functies gebruiken dan moeten we elke pin individueel aansturen wat meer codelijnen en tijd vereist:

```

1 void setup()
2 {
3   for (int i = 0; i<=7; i++) pinMode(i, OUTPUT);
4 }
5
6 void loop()
7 {
8   for (int i = 0; i<=3; i++) digitalWrite(i, LOW);
9   for (int i = 4; i<=7; i++) digitalWrite(i, HIGH);
10  delay(1000);
11  for (int i = 0; i<=3; i++) digitalWrite(i, HIGH);
12  for (int i = 4; i<=7; i++) digitalWrite(i, LOW);
13  delay(1000);
14 }
```

Een pin aansturen

Het volgende voorbeeld toont hoe een LED (en dus één pin) kan aangestuurd worden door rechtstreeks registers aan te spreken in de processor.

In de functie `setup()` wordt om de 5-de pin in te stellen in het DDR-register een 1 5x naar links geshift. Daarna wordt de 1 op de juiste positie d.m.v. de OR-functie aan de volledige byte toegevoegd.

Dezelfde methode wordt daarna gebruikt in `loop()` om de led afwisselend hoog en laag te maken.

```

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 }
```

De berekening hieronder toont de verschillende stappen:

```

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
```

Als we de beschikbare Arduino-functies gebruiken dan is de voorgaande code equivalent met:

```

1 #define LED_NUMMER 13
2
3 void setup() {
4     pinMode(LED_NUMMER, OUTPUT);
5 }
6
7 void loop() {
8     digitalWrite(LED_NUMMER, HIGH);
9     delay(500);
10    digitalWrite(LED_NUMMER, LOW);
11    delay(500);
12 }
```

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

6.2 Knoppen

In de meeste toepassingen is input noodzakelijk, de Arduino moet dus gegevens kunnen ontvangen. Dit kan gaan over het inlezen van knoppen, het uitlezen van sensoren, gegevens ontvangen

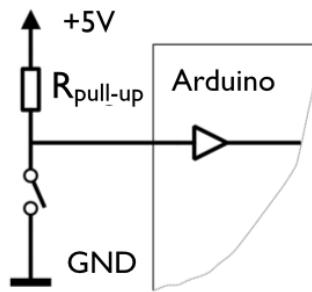
via de seriële poort, enz.

Knoppen worden ook wel drukschakelaars genoemd. Ze maken (of verbreken) een verbinding tussen twee contacten als de knop ingeduwd wordt. In de meeste gevallen wordt terug gegaan naar de vorige toestand als de knop wordt losgelaten.

De toestand van een knop kan ingelezen worden via een digitale input. We definiëren hiervoor een pin als input met de functie `pinMode()` of door het aanpassen van het DDR-register. De toestand zelf kan uitgelezen worden met de functie `digitalRead()` of door het PIN-register te raadplegen.

6.2.1 Pull-up

De input knoppen van het Clock Shield zijn verbonden met de GND en een input pin. Wanneer deze niet ingedrukt zijn dan is de input pin in een zwevende toestand. Om dit te verkomen wordt een pull-up weerstand (figuur 6.2) gebruikt die verbonden is tussen voedingsspanning en de input pin, deze maakt de toestand van de input pin hoog als de knop niet ingedrukt wordt.



Figuur 6.2: Pull-up weerstand

Interne pull-up

Het Clock Shield bevat zelf geen pull-up weerstanden. Deze zijn niet nodig aangezien er intern in de Arduino zijn ingebouwd. De pull-up weerstanden in de Arduino kunnen geactiveerd worden met de functie `pinMode()`.

Er zijn twee mogelijkheden om een pin als INPUT te definiëren (met en zonder interne pull-up weerstand): `INPUT` en `INPUT_PULLUP`.

Wanneer we een pin in leesmodus plaatsen dan is het bijhorende PORT-bit losgekoppeld van de pin zelf. Wanneer het PORT-bit in dat geval hoog gemaakt wordt dan wordt de interne pull-up weerstand geactiveerd, in het andere geval is hij gedeactiveerd (figuur 6.1).

In het voorbeeld hieronder wordt knop K1 verbonden met pin 9 van de Arduino. De pinmode ervan (met pull-up weerstand) wordt ingesteld in de functie `setup()`. Daarna wordt in de

loop() telkens de waarde van K1 gecontroleerd met een digitalRead() en wordt indien de knop ingedrukt is, de LED aangeschakeld.

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 uit het voorbeeld met digitalRead() aan zodat de LED ook kan uitgezet worden. Gebruik hiervoor knop K2.

6.2.2 PIN-register

Het uitlezen van een INPUT-pin gebeurt in het PIN-register. Ook hier wordt een volledige byte gelezen, de gewenste bits moeten er nog worden uitgefilterd. Eerst wordt de gebruikte pin in leesmodus geplaatst door de bijhorende bit in het DDRB-register laag te maken. Daarna wordt in de loop() het PINB-register uitgelezen waaruit de gebruikte pin uitgefilterd wordt.

```

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
11 void loop() {
```

```

12     if((PINB & (1<<K1)) == 0) PORTD |= (1<< LED1);
13 }
```

Defening

Pas de code uit het voorbeeld met het PIN-register aan zodat de LED ook kan uitgezet worden. Gebruik hiervoor knop K2.

6.2.3 Polling

De methode die gebruikt wordt in het vorige voorbeeld noemt men *polling*. In de `loop()`-functie wordt continu de toestand van de knop(pen) uitgelezen. Dit belast de CPU vrij intensief, deze kan dan ondertussen weinig andere zaken doen of verbruikt meer stroom dan nodig.

Als er nog veel andere taken zijn worden de knoppen niet continu uitgelezen, dit geeft een trage reactie als gevolg. De oplossing voor deze problemen is het gebruik van *interrupts* (zie later).

Tellen

Een volgend probleem is om een telling uit te voeren die gekoppeld is aan elke druk op de knop. De teller moet voldoende snel reageren (menselijke reactie is ongeveer 100ms) en moet correct tellen: één druk is één tel.

Bij iedere druk op de knop wordt dus een teller verhoogd. Zonder extra wachttijd (delay) zal één druk ongeveer 40x geteld worden. Dit komt doordat de software sneller is dan het indrukken van de knop door een persoon.

Als we in elke uitleescyclus een wachttijd van ongeveer 100ms invoeren dan wordt er vrij correct geteld. Een gevolg is dat bij het blijven indrukken van de knop er om de 100ms wordt geteld. In de praktijk is dit meestal geen probleem.

Het voorbeeld hieronder zal dus om de 100ms de stand van de knop inlezen en de teller indien ingedrukt verhogen.

```

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

Aantallen tellen

Als één druk als één moet geteld worden en dus niet om de 100ms dan moet de code worden uitgebreid. De telling wordt dan onafhankelijk van de indruktijd.

Om dit probleem op te lossen kan er gewacht worden totdat de knop terug losgelaten wordt.

In het voorbeeld hieronder is een while-lus toegevoegd dat de code laat wachten totdat de knop terug losgelaten wordt. De delay is ook iets verkleind zodat de code sneller is dan de menselijke reactie.

```

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

Het voorgaande voorbeeld telt correct (op eventuele dender na) en reageert quasi onmiddellijk. Het gevolg is wel dat eventuele andere taken die moeten uitgevoerd worden geblokkeerd worden door de while-lus.

In het voorbeeld is het togglen van een LED een andere taak die los staat van de teller. Deze zal niet togglen als de knop ingedrukt gehouden wordt. Een oplossing is de code vrijgeven tijdens het indrukken van de knop en werken met een variabele die getoggled wordt.

```

1 boolean statusK1;      // statusvariabele
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; // toestand = ingedrukt
9     }
10    digitalWrite(LED1,!digitalRead(LED1));
11    delay(50);
12 }
```

In het voorbeeld hierboven is de boolean statusK1 normaal gezien laag. Tijdens het indrukken van de knop wordt deze tijdelijk hoog gemaakt, er wordt dan niet geteld. Als de knop terug

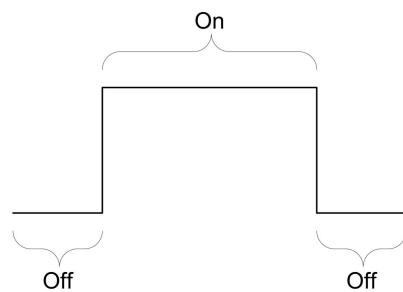
losgelaten wordt ($K_1 = \text{hoog}$) dan wordt de boolean terug laag gemaakt om een volgende druk op de knop te kunnen opvangen.

6.3 Dender (bounce)

Dender (in het Engels bounce) wordt ook wel glitches genoemd. Dit treedt op bij elke druk op de knop. Er wordt bij het indrukken van de knop dan meer dan éénmaal contact gemaakt, dit is een ongewenst gedrag. De ernst hiervan is afhankelijk van de kwaliteit van de knop.

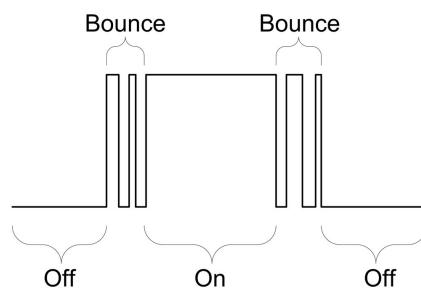
Bij elke overgang is er dus een onstabiele toestand wat een foutieve output kan veroorzaken in de code.

Figuur 6.3 geeft een correcte werking weer zonder dender.



Figuur 6.3: Signaal zonder dender

Figuur 6.4 toont een werking die meestal in de praktijk optreedt, er wordt meer dan éénmaal contact gemaakt.



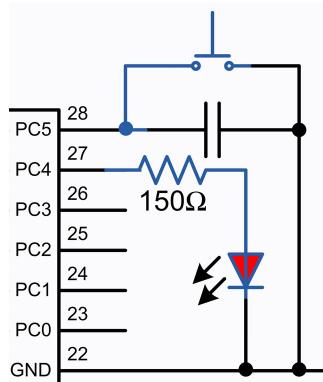
Figuur 6.4: Signaal met dender

6.3.1 Dender voorkomen

Dender voorkomen kan op twee manieren: hardware-gebaseerd of software-gebaseerd.

Hardware

Men kan de schakeling aanpassen zodat geen dender optreedt (figuur 6.5), er wordt een tijdsvertragend element (zoals een condensator) toegevoegd die de pieken opvangt en de dender dus ongedaan maakt.



Figuur 6.5: Ontdendering met condensator

De condensator wordt opgeladen via de pull-up weerstand, dit zorgt voor de nodige tijdsvertraging om de dender te overbruggen. Deze oplossing is duurder dan de software-oplossing omdat er meer componenten nodig zijn, ze wordt dan ook maar enkel in speciale gevallen gebruikt.

Software

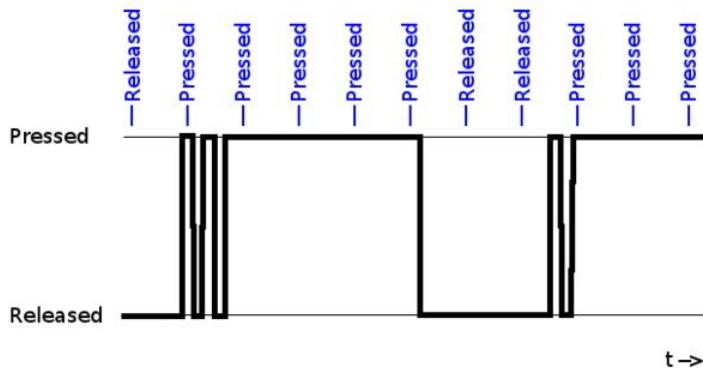
Hier wordt de code op de Arduino aangepast zodat juist gereageerd wordt, de software zorgt er hier voor dat de stoorpulsen niet in rekening gebracht worden (figuur 6.6).

Meestal wordt een tijdsvertraging ingebouwd die iets langer is dan de praktische dendertijd, dit is enkele milliseconden. Als er ook nog andere taken uitgevoerd worden is er meestal geen extra delay nodig.

De code hieronder houdt geen rekening met dender en zal dus af en toe verkeerd tellen in geval dat er dender optreedt. Dat er meestal toch correct wordt geteld komt door de functie `Serial.println()`, deze duurt enkele ms en dat is meestal voldoende om de dender te overbruggen. Bij een slechte knop wordt toch soms dubbel geteld.

```

1 void loop() {
2     if (digitalRead(K1) == LOW) {
```



Figuur 6.6: Software ontdegendering

```

3     Serial.println(teller);
4     teller++;
5     while(digitalRead(K1) == LOW);
6 }
7 }
```

In het volgende voorbeeld wordt de functie `Serial.println()` buiten de tellus geplaatst. Ze kan dus niet helpen om de dender te overbruggen. Het gevolg is dat in de praktijk de teller reeds na 5 tot 7x indrukken van de knop de waarde 10 bereikt. Deze code kan dus ook enkel en alleen gebruikt worden bij een goede hardware-ontdendering.

```

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 }
```

Als er in de tellus geen andere (tijdsconsumerende) taken zijn kunnen we zelf een korte delay toevoegen. Een 50-tal ms in totaal volstaat meestal. De code hieronder telt correct en is onafhankelijk van eventuele andere code.

```

1 void loop() {
2     if (digitalRead(K1) == LOW) {
3         delay(25); // denderdelay
4         teller++;
5         while(digitalRead(K1) == LOW);
```

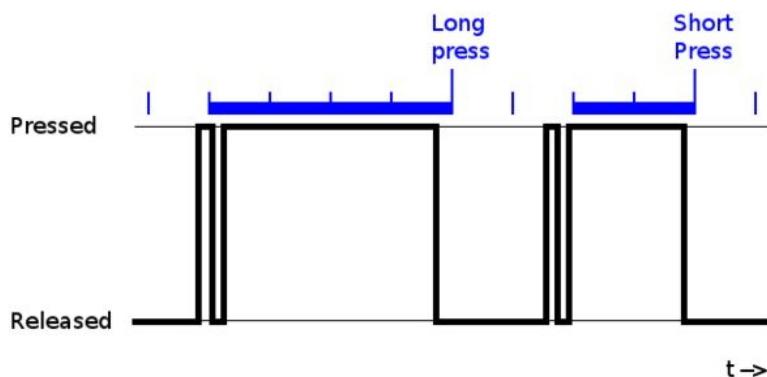
```

6     delay(25); // denderdelay
7 }
8 if (teller==10) {
9   Serial.println("10x ingedrukt");
10 }
11 }
```

6.4 Lange en korte drukken

In sommige toepassingen is het handig om verschillende functies te koppelen aan een korte of lange druk. Men kan dan met één knop meerdere functies uitvoeren. Dit wordt o.a. veel bij uurwerken gebruikt waarbij een korte druk de functie wisselt en een lange druk gebruikt wordt om de tijd correct in te stellen.

Om dit toe te passen moet na iedere druk gekeken worden hoeveel tijd er gepasseerd is (figuur 6.7).



Figuur 6.7: Lange en korte druk

6.4.1 millis()

De functie `millis()` toont de verstreken tijd na een power-up of reset van het Arduino-bordje, dit in milliseconden. Het datatype is een *unsigned long* aangezien de tijden kunnen oplopen. De tijd loopt over na ongeveer 50 dagen.

Deze functie is heel handig om tijdsverschillen te meten, ze is namelijk onafhankelijk van de andere uitgevoerde code.

Oefening

Welke getallen verschijnen er bij het uitvoeren van onderstaande code?

```

1 unsigned long time;
2
3 void loop(){
4     Serial.print("Tijd: ");
5     time = millis();           // tijd opvragen
6     Serial.println(time);
7     delay(1000);             // een seconde wachten
8 }
```

6.4.2 Lang en kort onderscheiden

In onderstaande code wordt bij een druk op de knop gewacht totdat deze weer losgelaten wordt. Als blijkt dat de tijd erna 200ms hoger ligt dan de oorspronkelijke tijd dan was het een lange druk. In het andere geval is het een korte druk.

```

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 }
```

6.5 Meerdere knoppen

Tot nu toe werd telkens maar één knop bediend. Meerder knoppen vormen geen probleem als de knoppen onafhankelijk van elkaar werken. Let wel op totale tijdsvertragingen, deze kunnen oplopen naargelang het aantal knoppen toeneemt.

Hou ook rekening met conflicten, als K1 ingedrukt is zal bv. niet gereageerd worden op K2. Dit kan problemen opleveren. Een noodstop moet bv. steeds voorrang hebben.

De listing hieronder toont een eenvoudig voorbeeld die twee knoppen bedient. Als oefening kan je je afvragen wat er gebeurt als beide knoppen tegelijk ingedrukt worden? Is er ook een voorrangsregeling?

```

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

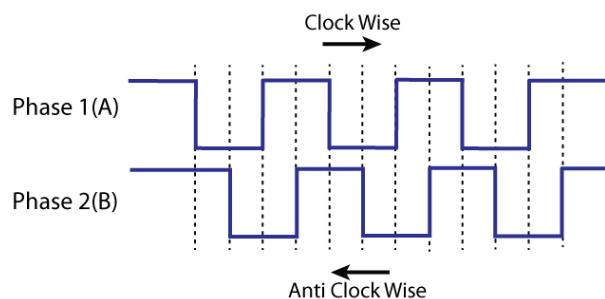
6.6 Rotary encoder

Een rotary encoder (figuur 6.8) is een digitaal alternatief voor een analoge potentiometer waarvan de weerstandswaarde kan gewijzigd worden. Dit wordt bv. gebruikt als volumeknop.



Figuur 6.8: Rotary encoder

Bij iedere draaibeweging produceert de rotary encoder pulsen op verschillende pinnen (figuur 6.9). De draaisnelheid bepaalt het aantal pulsen per seconde, de draairichting is afhankelijk van de volgorde van de pulsen (A t.o.v. B).



Figuur 6.9: Pulsentrein

6.6.1 Positie bepalen

Het is bij een rotary encoder noodzakelijk om te kunnen bepalen in welke positie hij zich bevindt t.o.v. de oorspronkelijke toestand. Men zet bijvoorbeeld een teller op een standaard (veilige) waarde bij het starten van het toestel en dan kan men de teller verhogen of verlagen door een draai aan de knop.

Het verhogen en verlagen van de teller gaat als volgt:

- 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

Merk op dat in de code dus enkel A gecontroleerd wordt, enkel bij een verandering van A wordt ook naar B gekeken.

```
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 }
```

6.7 Digitale output

Naast input is er ook output nodig, onder digitale output verstaan we pinnen die ofwel hoog of laag gemaakt worden, er zal dus 5V of 0V op gemeten worden.

De pinnen kunnen niet onbegrenst belast worden, er is een maximum stroom per pin en per poort. Er is ook verschil tussen stroom *sourcen* (leveren) en *sinken* (opnemen).

Voor sommige componenten (zoals LED's) moet een stroombeperking voorzien worden zodat de stroom beperkt wordt tot een veilige waarde.

6.7.1 Maximum stromen

Bij de Arduino UNO zijn de stromen als volgt beperkt:

- max. 40mA per pin (continu: 20mA)
- max. 150mA per poort (8-bits) leveren (source)
- max. 100mA per poort opnemen (sink)

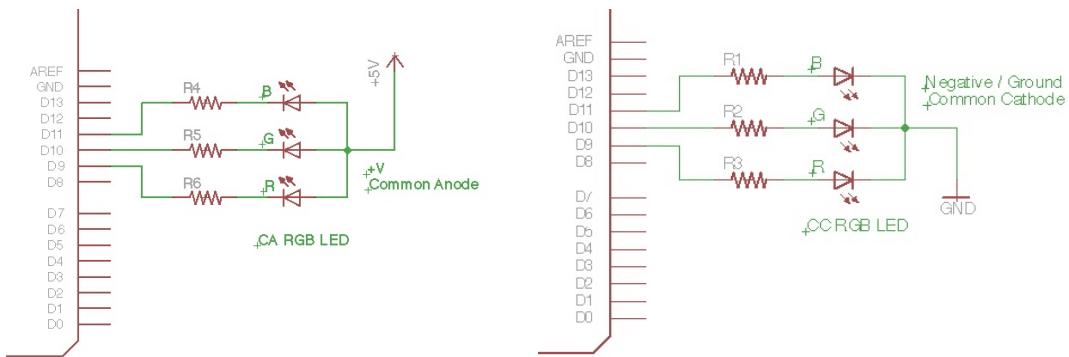
Een pin kan dus perfect een LED (20mA) voeden, let op met meerdere LED's per poort. De totale stroom per poort mag niet boven de toegestane waarde komen.

6.7.2 LED aansluiten

Een gewone LED kan op twee manieren aangesloten worden waarbij de stroom in de pin zal sourcen of sinken.

Bij een RGB-LED of 7-segment display waarbij een aantal pinnen gemeenschappelijk zijn moet men rekening houden met CA (common anode) of CC (common cathode).

In figuur 6.7.2 heeft de linkse RGB-LED een CA-aansluiting. Pinnen 9, 10 en 11 zullen dus stroom opnemen. De rechtse RGB-LED met een CC-aansluiting zorgt ervoor dat de pinnen stroom leveren.



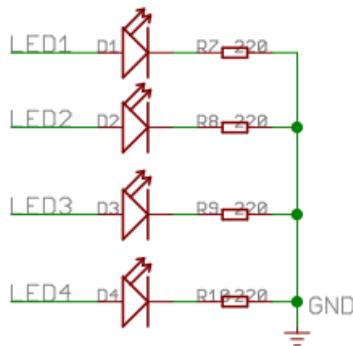
Oefening LED clock shield

Het clock shield bevat ook een aantal LED's (figuur 6.10).

Bereken de stromen door de LED's, de spanningsvallen over de LED's zijn respectievelijk 2V (rood en groen) en 3,4V (blauw).

6.8 Interrupts

Bij polling wordt de hardware continu gecontroleerd op veranderingen door de software op de Arduino. Dit veroorzaakt veel tijdsverlies voor de CPU en geeft tegelijkertijd een trage reactie van de software als er nog (veel) andere taken moeten uitgevoerd worden.

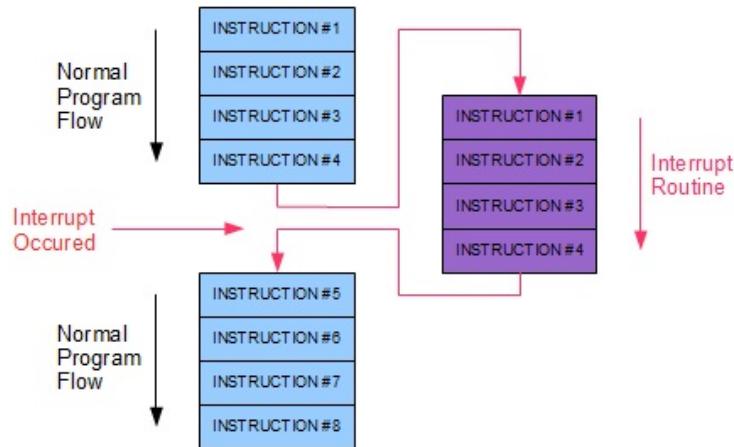


Figuur 6.10: LED aansluiting

Door het gebruik van interrupts wordt de werkwijze aangepast. De hardware genereert een signaal bij een verandering zodat er niet continu moet nagegaan worden of er iets veranderd is. De CPU is ondertussen vrij om andere taken te doen, er zal dus sneller gereageerd worden en andere taken worden vlotter uitgevoerd. Het nadeel aan interrupts is dat de hardware complexer is om de interrupt te kunnen melden.

Ook de microcontroller op het Arduino-bordje ondersteunt interrupts. Een aantal typische voorbeelden zijn externe interrupts, pin change interrupts en timer interrupts.

Afhandeling interrupt



Figuur 6.11: Interrupt afhandeling

Als de hardware een interrupt genereert dan onderneemt de CPU de volgende stappen:

- de interrupt-vlag wordt nul gemaakt, om invloed van andere interrupts te beperken

- de huidige programmapositie wordt tijdelijk bewaard
- het programma springt naar de code voor de interrupt
- na het uitvoeren van de interruptcode wordt verder gegaan met de gewone code

Een interrupt is dus in staat om de huidige werking te onderbreken en de interruptcode op te roepen (figuur 6.11).

6.8.1 Externe interrupts

Dit type van interrupt wordt getriggerd door een digitale ingang. De hardware (en dus niet de software) monitort continu deze pinnen en geeft bij een verandering een interrupt-signal door aan de CPU.

Externe interrupts zijn zeer snel maar om hardwareredenen beperkt tot specifieke pinnen. Bij de Arduino Uno is dit INT0 op pin 2 en INT1 op pin 3.

Gebruik externe interrupts

Om een externe interrupts in te schakelen kan de functie `attachInterrupt(interrupt, isr, mode);` gebruikt worden.

Hierbij is `interrupt` het nummer van de interrupt (0 of 1), `isr` de naam van de interrupt routine die aangeroepen wordt bij een interrupt en `mode` RISING, FALLING, CHANGE of LOW, dit is de gewenste verandering die moet gedetecteerd worden.

Snelheid

Om de snelheid van je code te optimaliseren hou je de interrupt routine best zo kort mogelijk. Tijdens de uitvoer ervan wordt de rest van het programma gestaakt. Ook andere interrupts worden ondertussen genegeerd.

Gebruik verder geen seriële functies of een `delay()`-functie in de interrupt routine. Deze maken namelijk ook gebruik van interrupts en zullen daardoor niet correct functioneren in een interrupt routine.

Men kan eventueel zelf een vertraging genereren door het uitvoeren van nutteloze code, de assembler-instructie `nop` voert niks uit maar duurt wel enkele klok pulsen:

```

1 void wacht() {
2     // 200ms = +/- 450000
3     for (unsigned long i=0; i<450000; i++) {
4         __asm__("nop\n\t");

```

```
5 }  
6 }
```

Interrupt uitschakelen

Soms is het noodzakelijk om een interrupt uit te schakelen tijdens de werking van een programma. Dit kan bijvoorbeeld noodzakelijk zijn bij tijdskritische zaken die in geen geval mogen onderbroken worden.

Het uitschakelen van een interrupt kan met de functie `detachInterrupt(interrupt)`, daarna is het mogelijk om de interrupt terug in te schakelen.

Voorbeeld

De code hieronder toont een voorbeeld van de werking van een externe interrupt. Er is een knop verbonden met digitale pin 2 en dus gekoppeld aan interrupt 0. Met de functie `attachInterrupt()` wordt de interrupt ingeschakeld en zal hij in werking treden bij een overgang van hoog naar laag, bij het indrukken van de knop dus.

```
1 #define LED 4 // LED op pin 4  
2  
3 void knopISR() {  
4     digitalWrite(LED, HIGH); // LED aan  
5 }  
6  
7 void setup() {  
8     pinMode(LED, OUTPUT); // LED uitgang  
9     pinMode(2, INPUT_PULLUP); // pin 2 input  
10    attachInterrupt(0, knopISR, FALLING);  
11    digitalWrite(LED, LOW); // LED uit  
12 }  
13  
14 void loop() {}
```

Oefening

Pas de code uit het voorbeeld aan zodat de LED ook kan uitgezet worden. Gebruik hiervoor de interrupt op poort 3 (INT1).

6.8.2 Pin Change interrupts

De externe interrupts kunnen maar op 2 pinnen gebruikt worden die ook nog eens vast gedefinieerd zijn. Om andere pinnen te laten reageren op een interrupt bestaan er *Pin Change Interrupts*.

Deze zijn softwaregebaseerd en bijgevolg dus trager. Verder reageren ze op elke verandering (*change*), dus zowel bij een stijgende (*rise*) als dalende puls (*fall*).

De *Pin Change Interrupts* werken per poort, er zullen dus meerdere pinnen dezelfde interrupt genereren. Een overzicht zie je in het lijstje hieronder:

- PCINT0: pin D8 tot D13
- PCINT1: pin A0 tot A5
- PCINT2: pin D0 tot D7

Gebruik

De Pin Change Interrupts kunnen ingeschakeld worden met de functie `pciSetup()`. Deze wordt in `setup()` opgeroepen voor **elke** gewenste pin, ook al behoren ze tot dezelfde poort en dus dezelfde interrupt. De code die in deze functie staat stelt enkele registers correct in en moet exact zijn zoals vermeld in het voorbeeld.

Daarnaast moet een interrupt routine (ISR) voorzien worden per poort, als de gebruikte pinnen tot dezelfde poort behoren dan is maar één ISR nodig. Let op de correcte schrijfwijze ervan en de juiste parameters.

Aangezien er maar één ISR is voor soms meerdere pinnen moet er in de ISR nog verder gecontroleerd worden welke pin de interrupt heeft veroorzaakt.

Als voorbeeld gebruiken we het Clock Shield waarbij K1, K2 en K3 respectievelijk verbonden zijn met pin 9, 10 en 11. Deze behoren alle drie tot PCINT0 en gebruiken dezelfde ISR. De code zal via een interrupt op K1 LED3 inschakelen en via knop K2 uitschakelen.

```

1 #define K1 9 // K1 op pin 9
2 #define K2 10 // K2 op pin 10
3 #define LED3 4 // LED3 op pin 4
4
5 void pciSetup(byte pin)
6 {
7     *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin)); //enable pin
8     PCIFR |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt

```

```

9   PCICR |= bit (digitalPinToPCICRbit(pin)); // enable ↓
10  →interrupt for the group
11 }
12 void setup() {
13   pinMode(LED3, OUTPUT);
14   pinMode(K1, INPUT_PULLUP);
15   pinMode(K2, INPUT_PULLUP);
16   pciSetup(K1); // interrupt op K1
17   pciSetup(K2); // interrupt op K2
18 }
19
20 void loop() {}
21
22 ISR (PCINT0_vect) // afhandeling interrupt pin D8 tot D13
23 {
24   if (digitalRead(K1) == LOW) digitalWrite(LED3, HIGH); // ↓
25   →controle op pin-niveau
26   if (digitalRead(K2) == LOW) digitalWrite(LED3, LOW);
27 }
```

6.8.3 Volatile

Let op met variabelen die ook in een interrupt routine gebruikt worden. Standaard worden variabelen bewaard in een tijdelijk geheugenregister. In bepaalde omstandigheden kunnen de variabelen onbedoeld gewijzigd worden, bij de Arduino UNO kan dit gebeuren bij het aanroepen van een interrupt.

De oplossing bestaat erin om de variabele te declareren als *volatile*, op die manier wordt de variabele bewaard in het RAM geheugen en niet in een geheugenregister.

Het voorbeeld hieronder toont hoe je een variabele op een veilige manier kan gebruiken in een interrupt routine.

```

1 #define LED 4
2 volatile bool toestand = LOW;
3
4 void setup() {
5   pinMode(LED, OUTPUT);
6   attachInterrupt(0, knopISR, FALLING);
7 }
8
9 void loop() {
```

```

10   digitalWrite(LED, toestand);
11 }
12
13 void knopISR() {
14     toestand = !toestand;
15 }
```

6.9 Output signaal versterken

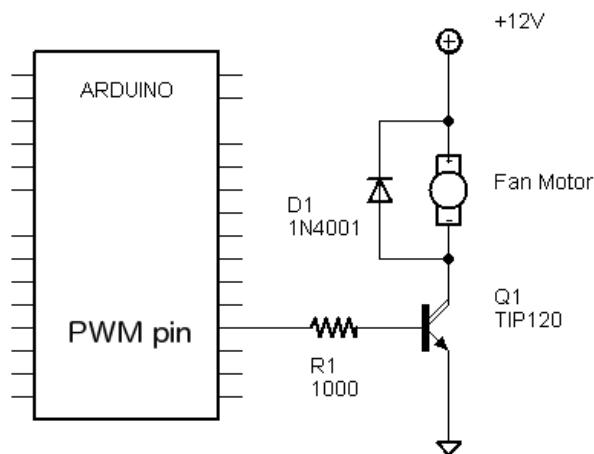
Een digitale output kan gebruikt worden om bv. een LED aan te sturen, hiervoor levert de microcontroller voldoende stroom. In veel toepassingen zal de geleverde stroom echter niet volstaan en moet het signaal versterkt worden.

Naargelang de toepassing en de eisen ervan (zoals stroomsterkte, galvanische scheiding, ompoling en snelheid) kan voor verschillende opties gekozen worden.

6.9.1 Transistor

Een transistor kan prima als schakelaar fungeren (figuur 6.12). Het stroomverbruik voor de sturing van de transistor zelf is daarbij minimaal.

Een transistor kan naargelang het type grote stromen aan, is snel en laat PWM toe. Met één transistor kan geen ompoling gerealiseerd worden.



Figuur 6.12: Transistor als schakelaar

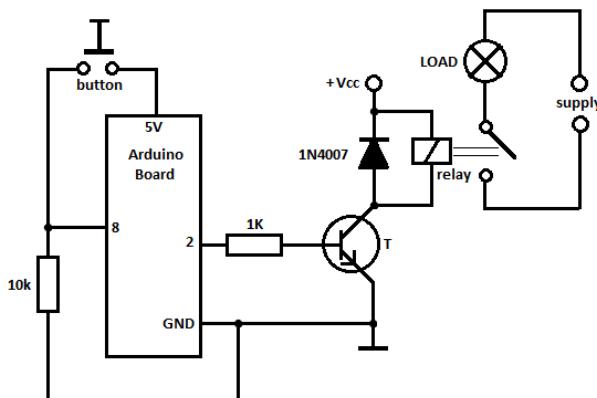
6.9.2 Relais

Een relais (figuur 6.13) heeft als voordeel dat hij een galvanische scheiding voorziet tussen de microcontroller en de belasting. Er is dus geen rechtstreeks elektrisch contact tussen beide.

Verder kan een relais (zeer) grote stromen aan. Het is wel traag en bijgevolg is er geen PWM mogelijk. Een relais laat geen ompolting toe.

Meestal is er ook een transistor nodig om de spoel van het relais aan te sturen, die vraagt meestal meer stroom dan dat de microcontroller kan leveren.

Een relais wordt meestal gebruikt om netspanning te schakelen via de microcontroller.



Figuur 6.13: Schakeling met relais

Het *relais* shield zoals in figuur 6.14 bevat een aantal relais.

6.9.3 H-brug

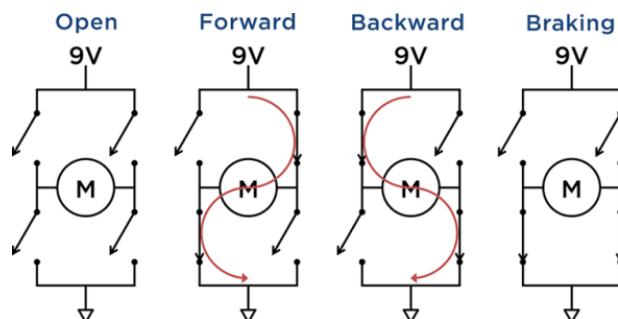
Wanneer we niet enkel de stroom wensen te versterken maar die ook willen kunnen ompolen dan komt de H-brug (figuur 6.15) van pas. Deze is meestal uit transistoren of MOSFET's opgebouwd en kan dus even grote stromen aan. Een H-brug opgebouwd uit transistoren of MOSFET's is snel en laat dus PWM toe.

Een typische toepassing is een DC-motor, daarvan kunnen we de draaizin wijzigen door de stroom om te polen. Er zijn speciale IC's beschikbaar die reeds één of meerdere complete H-bruggen bevatten (bv. L293D).

Een H-brug wordt toegepast in het *motorshield* (figuur 6.16) voor de Arduino.



Figuur 6.14: Relais shield



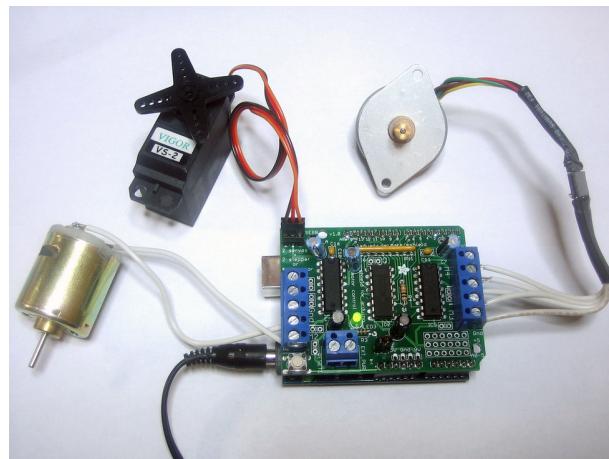
Figuur 6.15: Werking H-brug

6.9.4 Optocoupler

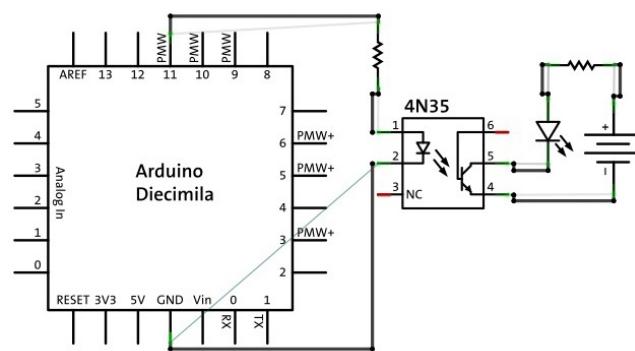
Ten slotte kan men in een schakeling met relatief kleine stromen die galvanisch gescheiden moeten zijn een optocoupler (figuur 6.17) gebruiken. Deze werkt op basis van licht, aan de kant van de microcontroller wordt het signaal omgezet in een lichtstraal en langs de kant van de belasting wordt deze lichtstraal terug omgezet in een elektrisch signaal.

Door de gebruikte technologie is een optocoupler snel en laat dus PWM toe. Net zoals een H-brug is een optocoupler ook beschikbaar in IC-vorm.

Het DMX-protocol dat o.a. gebruikt wordt in de theaterwereld wordt aangewend om licht en geluid te sturen. Doordat de installaties doorgaans complex zijn is het aan te raden om de sturing galvanisch te scheiden van de belasting. Het DMX-shield (figuur 6.18) bevat een aantal optocouplers die deze taak op zich nemen.



Figuur 6.16: Motorshield



Figuur 6.17: Schakeling met optocoupler



Figuur 6.18: DMX-shield

Bibliotheken

De Arduino is heel populair geworden door o.a. het enorme codeaanbod op internet. Het meeste van deze code wordt aangeboden in de vorm van een bibliotheek. Een bibliotheek bevat doorgaans een aantal functies rond een bepaald item, dit kan gaan over een sensor, een timer, wiskunde functies, ...

Door het enorme aanbod kan het gebeuren dat er voor bv. een bepaalde sensor meer dan één bibliotheek bestaat. Het is dan aan de gebruiker om hieruit de meest geschikte bibliotheek te vinden die aan zijn verwachtingen voldoet. Meestal zal de bibliotheek waarnaar de fabrikant van de sensor verwijst de meeste functies bevatten en het best functioneren.

7.1 Installatie en locatie

7.1.1 Installatie

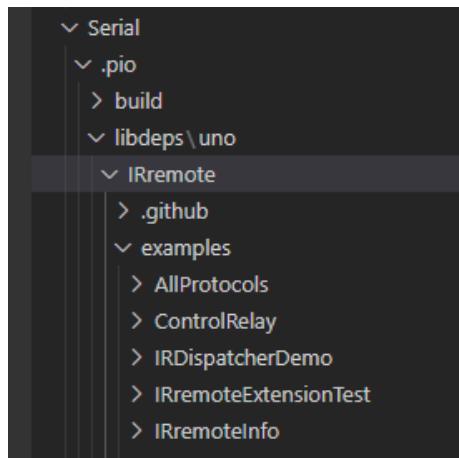
Bibliotheken worden in PlatformIO beheerd via het tabblad *Libraries* in het venster van *PlatformIO Home*. Je kan zoeken op een naam van een bibliotheek en deze toevoegen aan je project.

Let op: in tegenstelling tot de Arduino IDE wordt een bibliotheek in Platform IO gekoppeld aan een project. Dit heeft als voordeel dat je voor verschillende projecten verschillende versies van bibliotheken kan gebruiken. Bij de Arduino IDE wordt een bibliotheek geïnstalleerd voor alle projecten, het kan dus gebeuren dat er een nieuwe versie van een bibliotheek geïnstalleerd wordt en dat een oud project daardoor niet meer zal werken.

Na het koppelen van een bibliotheek aan een project, zal in het bestand `platformio.ini` van het project een extra lijn verschijnen met de vermelding `lib_deps` en de naam van de bibliotheek.

7.1.2 Locatie

De bibliotheken worden in PlatformIO bewaard bij het project onder de map `.pio\libdeps\`. Iedere map bevat ook een map `examples` met voorbeelden die de bibliotheek gebruikt (figuur 7.1).



Figuur 7.1: Map bibliotheken

7.2 Soorten bibliotheken

Er zijn verschillende soorten bibliotheken: de bibliotheken die reeds standaard aanwezig zijn in de IDE, extra bibliotheken die gedownload worden via internet of persoonlijke bibliotheken die men zelf ontwikkelt.

7.2.1 Standaard bibliotheken

Deze bibliotheken worden meegeleverd met PlatformIO, er is dus geen extra installatie vereist. De complete lijst met meegeleverde bibliotheken vind je in PlatformIO onder het tabblad *Built-in bij Libraries*.

Een overzicht van een aantal ingebouwde bibliotheken:

- EEPROM: om het intern EEPROM-geheugen op de Arduino te gebruiken (1024 bytes op de UNO)
- Ethernet en Wifi: netwerktoegang, bedraad of draadloos

- GSM: gebruik van het GSM-netwerk om gegevens door te sturen
- LiquidCrystal en TFT: veel gebruikte displays
- SD: toegang tot een SD-kaart
- Servo en Stepper: deze types van motoren aansturen
- SPI en Wire: SPI en I²C protocollen gebruiken om o.a. sensoren uit te lezen of in te stellen

Gebruik

Een standaard bibliotheek kan geïmporteerd worden door de benodigde lijn code toe te voegen:
`#include <libraryheader>`.

Na het importeren kunnen alle functies uit de bibliotheek gebruikt worden.

In het codevoorbeeld worden de bibliotheken SPI en Audio geladen, daarna wordt van beide een functie gebruikt. Om een functie uit een bibliotheek te gebruiken wordt de naam van de functie voorafgegaan door de naam van de bibliotheek.

```
1 #include <SPI.h>
2 #include <Audio.h>
3
4 void setup()
5 {
6     SPI.setClockDivider(4);
7     Audio.begin(88200, 100);
8 }
```

7.2.2 Extra bibliotheken

Naast de standaard bibliotheken die reeds aanwezig zijn, zijn er ook nog extra bibliotheken ontwikkeld door de gemeenschap. Deze worden niet standaard meegeleverd en moeten dus handmatig worden geïnstalleerd. Dit werd reeds besproken in een voorgaande paragraaf.

Voorbeeld IR-ontvanger

Als voorbeeld willen we een schakeling kunnen besturen via een infrarood afstandsbediening.

De eerste stap is om de benodigde hardware te zoeken: een afstandsbediening (zowat elk exemplaar van een TV of radio is OK) en een IR-ontvanger (bv. een IR-fotodiode).

Daarna kan er gezocht worden naar de juiste bibliotheek: bv. *IRremote*. Er zijn nog andere bibliotheken beschikbaar, hier moet soms eerst getest worden vooraleer men de gewenste bibliotheek kan gekozen worden.

Eenmaal de bibliotheek geïnstalleerd is kan men een voorbeeld openen en een vlugge test uitvoeren. Let op de pinnummers, die zullen in jouw schakeling meestal anders zijn en moeten dus aangepast worden.

In het getoonde voorbeeld wordt de juiste pin gedefinieerd waar de foto-diode aan gekoppeld is en wordt de ontvanger gestart. Daarna worden alle ontvangen signalen getoond via de seriële monitor. Normaal heeft iedere knop op een afstandsbediening een unieke code.

```
1 #include <Arduino.h>
2
3 int RCV_PIN = 11;
4
5 IRrecv irrecv(RCV_PIN);      // IR-fotodiode
6
7 decode_results results;
8
9 void setup()
10 {
11     Serial.begin(9600);
12     irrecv.enableIRIn(); // Start the receiver
13 }
14
15 void loop() {
16     if (irrecv.decode(&results)) {
17         Serial.println(results.value, HEX);
18         irrecv.resume(); // Receive the next value
19     }
20 }
```

8

Displays

Onder displays verstaan we elke output die dient om een status of instelling zichtbaar te maken. Een interface met de gebruiker dus.

Er zijn verschillende soorten displays, gaande van de meest eenvoudige LED tot een volledig aanstuurbare aanraakgevoelig kleurenscherm:

- LED: wordt gebruikt om een status van een toestand te tonen
- LCD: toont meerdere alfanumerieke karakters en kan dus variabelen of meetwaarden weergeven
- 7-segment: kan één getal weergeven met een beperkt aantal cijfers, bv. een tijds- of een temperatuurswaarde
- TFT: een volledig aanstuurbare grafisch scherm dat uit afzonderlijke pixels bestaat, meestal in kleur, kan ook afbeeldingen en menu's weergeven

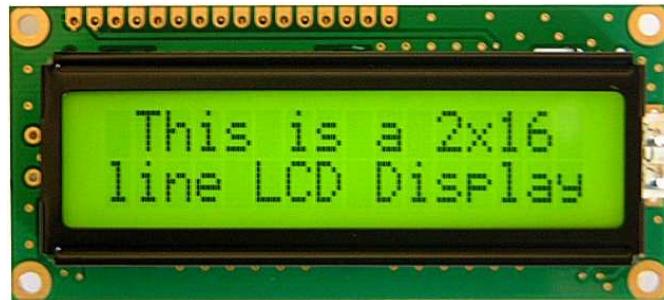
8.1 LCD

Het LCD (figuur 8.1), voluit Liquid Crystal Display bestaat standaard uit 2 lijnen van 16 alfanumerieke karakters. Grottere of kleinere afmetingen zijn mogelijk.

Aangezien het display is opgebouwd uit vloeibare kristallen verbruikt het zeer weinig stroom (zonder achtergrondverlichting). Het is dan ook heel geschikt voor batterij gevoede apparaten.

8.1.1 Aansluiting

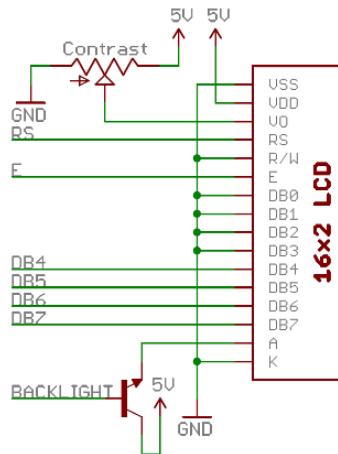
Een typisch LCD heeft 14 tot 16 pinnen om aan te sluiten. Er kan gekozen worden uit 4 of 8 datapinnen. De eerste optie heeft minder verbindingen nodig maar is trager, meestal wordt



Figuur 8.1: Standaard LCD

voor deze optie gekozen.

De aansluiting met een Arduino kan je in figuur 8.2 zien.



Figuur 8.2: Aansluiting LCD

8.1.2 Protocol

Vooraleer karakters kunnen getoond worden dient het LCD geïnitialiseerd te worden. Voor de initialisatie wordt er gewacht totdat de voedingsspanning gestabiliseerd is (ongeveer 15ms).

Na het wachten kunnen vervolgens de initialisatiebytes verstuurd worden, waarbij telkens gewacht wordt tussen de verschillende stappen.

Daarna kunnen de eigenlijke karakters verstuurd worden die moeten verschijnen op het display, deze bestaan telkens uit een datagedeelte en een puls op de enable pin.

Voor de Arduino is het volledig protocol geïmplementeerd in een bibliotheek: *LiquidCrystal*.

De voornaamste functies zijn: initialisatie, scrollen, wissen en schrijven. De cursor kan op zijn beurt aangepast worden naar wens: blink, underscore, ...

Verder is het mogelijk om een beperkt aantal eigen karakters aan te maken in het geheugen van het LCD en die te gebruiken.

Oefening: wat komt er op het display?

Analyseer het onderstaande codevoorbeeld en zoek uit wat er op het display zal verschijnen.

```
1 #include <LiquidCrystal.h>
2
3 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
4     // RS, enable, D4..D7
5
6 void setup() {
7     lcd.begin(16, 2);
8     lcd.print("hello, world!");
9 }
10
11 void loop() {
12     lcd.setCursor(0, 1); // 2de lijn
13     lcd.print(millis() / 1000);
14 }
```

8.1.3 Custom Characters

Een LCD laat toe om acht eigen karakters te definiëren. Deze worden eerst in een byte-array gedefinieerd. Daarna kan het karakter in het LCD aangemaakt worden met de functie `lcd.createChar()`. Het karakter tonen kan met de functie `lcd.write((byte))`.

In de volgende listing zie je een voorbeeld om een karakter te definiëren.

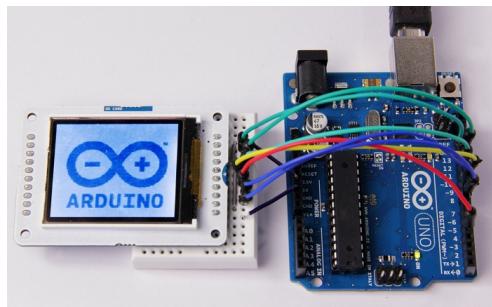
```
1 byte smiley[8] = {
2     B00000,
3     B10001,
4     B00000,
5     B00000,
6     B10001,
7     B01110,
8     B00000,
9 };
```

Daarna wordt het karakter effectief aangemaakt en getoond op het display.

```
1 void setup() {  
2     lcd.createChar(0, smiley);  
3     lcd.begin(16, 2);  
4     lcd.write(byte(0));  
5 }
```

8.2 TFT

Dit is een grafisch display dat meestal kleuren ondersteunt (figuur 8.3). Aangezien er veel pixels zijn die moeten aangestuurd worden zijn er ook veel pinnen. Deze worden aangestuurd via een matrix of een serieel protocol, bij de laatste optie is een ingebouwde microcontroller in het display noodzakelijk.



Figuur 8.3: TFT

8.3 7-segment

Deze display's (figuur 8.4) worden typische gebruikt om (kleine) getallen weer te geven, zoals bij weegschalen, klokradio's en thermometers. Het display bestaat uit 7 afzonderlijke segmenten waarmee elk decimaal cijfer kan gevormd worden. Per segment is een individuele LED ingebouwd om het segment te doen oplichten.

7-segment display's zijn betrouwbaar en relatief zuinig aangezien ze zelf voor hun verlichting instaan.

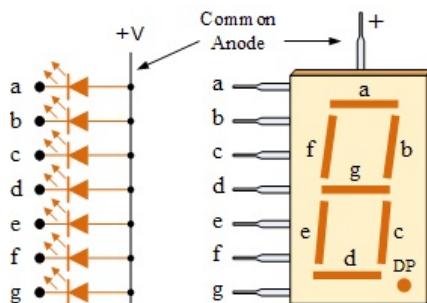


Figuur 8.4: 7-segment display

8.3.1 Opbouw

Ieder segment (en het decimale punt) heeft een afzonderlijke aansluiting. De andere kant van de ingebouwde LED is dan gemeenschappelijk met de andere segmenten (figuur 8.5). Daarbij kan men kiezen tussen een gemeenschappelijk anode (CA) of cathode (CC).

Het display is opgebouwd uit LED's, er is dus stroombegrenzing nodig per segment d.m.v. een weerstand.

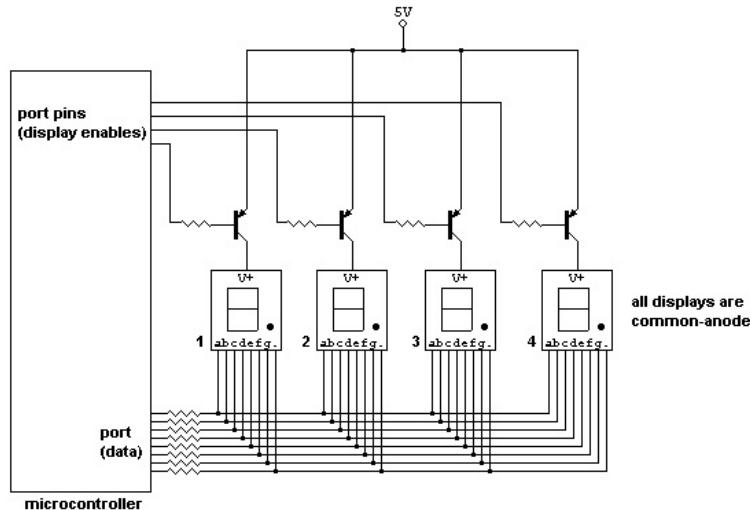


Figuur 8.5: Gemeenschappelijke anode aansluiting

8.3.2 Multiplex

Wanneer een display uit bv. 4 cijfers bestaat dan worden alle gelijke segmenten verbonden met elkaar. De gemeenschappelijke aansluiting zal men afwisselend aansturen in de tijd die snel genoeg is zodat het oog niks opmerkt. De stroom zal ook hoger liggen aangezien die maar een deel van de tijd aan de afzonderlijke LED's aangeboden wordt (figuur 8.6).

De methode van multiplexing vereist nog steeds redelijk veel pinnen en is rekenintensief voor de microcontroller aangezien er snel in de tijd van signaal moet gewisseld worden. Er bestaan dan ook speciale IC's, display drivers genoemd die de taak van het multiplexen op zich nemen. De drivers worden meestal serieel aangestuurd door een microcontroller die op die manier ontlast wordt van het multiplexen.



Figuur 8.6: Multiplexen van 7-segment display

Keypad

Bij het multiplexen kunnen we ook het keypad bekijken (figuur 8.7), dit is een mini toetsenbord voor input van getallen. Dit wordt bijvoorbeeld gebruikt bij een drankautomaat.

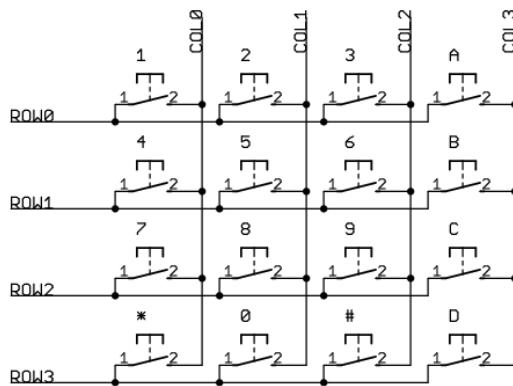


Figuur 8.7: Keypad

Het keypad is opgebouwd uit schakelaars die gemultiplexed worden in rijen en kolommen. Zodoende bekomt men een matrixstructuur (figuur 8.8).

Het uitlezen van een keypad is zeer analoog aan het aansturen van een 7-segment display. De pinnen verbonden met de rijen worden bv. als input gedefinieerd (met pullup weerstanden). De kolommen worden op hun beurt als output gedefinieerd.

Voor het uitlezen zal men de kolommen 1 voor 1 laag maken en telkens elke rij inlezen. Elke



Figuur 8.8: Multiplexen van keypad

combinatie van kolom en rij geeft een toetswaarde weer. Opnieuw is hier de methode rekenintensief (aansturen + uitlezen) en zijn er nog een behoorlijk aantal pinnen nodig.

TM1636 display driver

Dit IC, dat gebruikt wordt in het Clock Shield, is ontwikkeld om een 7-segment display en/of een keypad aan te sturen. Het IC vermindert het aantal benodigde pinnen tot 2 (data + clk) en kan de LED's in een display rechtstreeks aansturen zonder weerstand waarbij ook de helderheid kan geregeld worden.

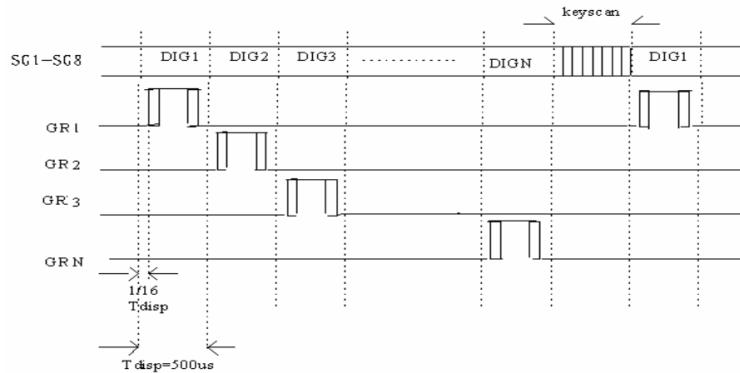
Het IC beschikt over een matrix uitgang (max. 4 x 7-segment) dat zowel een display of keypad kan aansturen of uitlezen.

Het IC wordt aangestuurd via een 2-draads protocol, dat een dialect is van I²C (zie later). Bij elk kloksignaal kan een bit doorgestuurd worden (figuur 8.9). Een 7-segment display en keypad kunnen eventueel samen gebruikt worden.

De Arduino bibliotheek TM1636 bevat verschillende functies om een 7-segment display aan te sturen:

- `init()` : initialiseren van een display (helderheid)
- `display()` : een array van 4 bytes tonen op het display
- `point()` : de vlag voor het dubbele punt instellen
- `clearDisplay()`: het display wissen

De voorbeeldcode hieronder initialiseert het display met standaard helderheid. Daarna wordt het getal 1234 op het display getoont.



Figuur 8.9: Protocol TM1636

```

1 #include <TM1636.h>
2 TM1636 tm1636(7,8); // CLK, DATA
3 int8_t disp[4];
4
5 void setup() {
6     tm1636.init();
7 }
8
9 void loop() {
10    disp[0]=1; disp[1]=2;
11    disp[2]=3; disp[3]=4;
12    tm1636.display(disp);
13    delay(500);
14 }
```

Het driver IC laat ook toe om eigen karakters te definiëren en te tonen, bv. het graden symbool voor een thermometer. Het definiëren gebeurt in het TM1636.cpp bestand waarbij de array TubeTab[] kan uitgebreid worden. De hexadecimale getallen stellen de bytewaarde voor van de bits uit een 7-segment cijfer.

```

1 static int8_t TubeTab[] =
2     {0x3f,0x06,0x5b,0x4f,
3      0x66,0x6d,0x7d,0x07,
4      0x7f,0x6f,0x77,0x7c,
5      0x39,0x5e,0x79,0x71,
6      0x40,0x00}; //0~9,A,b,C,d,E,F,"-", " "
```

9

Timers en tijd

9.1 Timers

Wanneer men een tijdsvertraging wenst te genereren, dan kan dit eenvoudig met de functie `delay()`, dit is echter CPU-belastend en houdt andere taken tegen. Ook het gebruik van de functie `millis()` is vrij belastend, in dat geval moet de klok steeds gecontroleerd worden om een tijdsverschil te kunnen meten.

Met het gebruik van de ingebouwde hardware timers kan op het gewenste ogenblik een interrupt gegenereerd worden. Vergelijk het met een kookwekker die biept na een bepaalde tijd, je moet dan zelf niet continu op de klok zitten te kijken.

Een (hardware) timer ontlast dus de CPU die zich ondertussen met andere taken kan bezighouden.

9.1.1 Timer interrupts

Timers werken aan de hand van interrupts, deze zijn niet gebaseerd op een input verandering maar zullen reageren op een (herhalende) tijdsgebeurtenis.

Een voorbeeld is om de vijf minuten een signaal genereren om een sensor uit te lezen. Dat zou ook kunnen met een `delay` of het gebruik van `millis()` maar dat zal de CPU veel meer beladen.

9.1.2 Timer1

Alle Arduino's beschikken over een aantal timers. De Arduino UNO bevat twee 8-bit timers en één 16-bit timer:

- timer0: 8-bit
- timer1: 16-bit
- timer2: 8-bit

De bitrate geeft een mate van nauwkeurigheid weer, zo zal een 8-bit timer van 0 tot 255 kunnen tellen en dan weer herbeginnen. Een 16-bit timer telt van 0 tot 65535 vooraleer hij overloopt. Bij dezelfde frequentie kan je met een 16-bit timer dus langer tellen of als beide tellers op hetzelfde moment overlopen zal de 16-bit teller dus sneller tellen en nauwkeuriger zijn.

9.1.3 Werking

Elke timer heeft een klok die voortdurend telt. Bij elke tik van de klok wordt de teller met één verhoogd. Als de teller een ingestelde vergelijkingswaarde bereikt zal er een interrupt optreden.

Daarna wordt de teller terug op 0 gezet en kan de werking herbeginnen.

We kunnen dit terug vergelijken met een kookwekker: als we die instellen op 10 minuten dan zal hij na die 10 minuten een signaal geven. Een digitale kookwekker kan meestal per seconde ingesteld worden en is dus vrij nauwkeurig. Een analoge kookwekker kan veel minder precies ingesteld worden en is dus minder nauwkeurig. Je zou dit ook kunnen vergelijken met een 16-en 8-bit timer.

9.1.4 Snelheid

De klok van de Arduino heeft een frequentie van 16MHz. Er is dus een tik van de klok om de 62,5ns. Voor de meeste toepassingen is dit (veel) te snel. We kunnen een vertraging genereren met behulp van een *prescaler*.

Als we geen prescaler instellen wordt de maximum tijd die we kunnen bereiken vooraleer de teller overloopt:

- 8-bit timer: $256 * 62,5\text{ns} = 16\mu\text{s}$
- 16-bit timer: $65536 * 62,5\text{ns} = 4\text{ms}$

De bekomen waarden maken duidelijk dat dit (veel) te kort is voor vertragingen van bijvoorbeeld 1s. Een prescaler die de klok vertraagt is dan nodig.

9.1.5 Prescaler

De snelheid van de timer (in Hz) is de Arduino kloksnelheid (16MHz) gedeeld door de waarde van de prescaler. Voorbeeld: een prescaler van 256 verlaagt de klok dus tot: $16\text{MHz} / 256 = 62,5\text{kHz}$.

De mogelijk instelbare prescalers zijn: 1, 8, 64, 256, en 1024. De laagste timersnelheid (grootste prescaler) is dus $16\text{MHz} / 1024 = 15,625 \text{ kHz}$.

Wanneer we bij deze prescaler een vertraging van 1s wensen zijn dus 15625 tellen nodig. Een 8-bit timer kan zo ver niet tellen, een 16-bit timer is dus nodig.

Maximale tijden

Een timer is dus niet eindeloos, de maximum instelbare tijden zijn:

- bij 8-bit $256/15625\text{Hz} = 16,4\text{ms}$
- bij 16-bit: $65536/15625\text{Hz} = 4,19\text{s}$

Door de timer in PWM-mode te gebruiken kunnen de tijden verdubbeld worden door enkel te reageren op een dalende of stijgende flank. De bibliotheek TimerOne.h maakt gebruik van deze mode, de maximum tijden zijn dan:

- bij 8-bit $256/15625\text{Hz} * 2 = 32,8\text{ms}$
- bij 16-bit: $65536/15625\text{Hz} * 2 = 8,39\text{s}$

Indien langere tijden gewenst zijn kan men de timer meerdere malen laten overlopen en zelf een extra teller bijhouden. Bijvoorbeeld voor 1min laten we een timer van 1s 60x overlopen.

9.1.6 Nauwkeurigheid (16-bit)

De gebruikte prescaler bepaalt ook hoeveel tijd er tussen elke tik verloopt en hoe nauwkeurig een 16-bit timer dus is. De tabel hieronder toont een overzicht. De max. periodes gelden voor PWM-mode.

Prescaler	Tijd per tik	Max periode
1	$0,0625\mu\text{s}$	$8,192\mu\text{s}$
8	$0,5\mu\text{s}$	$65,536\text{ms}$
64	$4\mu\text{s}$	$524,288\text{ms}$
256	$16\mu\text{s}$	$2097,152\text{ms}$
1024	$64\mu\text{s}$	$8388,608\text{ms}$

9.1.7 Invloed

De timers kunnen we zelf gebruiken voor onze toepassingen maar worden ook intern gebruikt door de Arduino voor allerlei tijdsgerelateerde zaken. Hieronder zie je een overzicht:

- **timer0** wordt gebruikt voor o.a.: `delay()`, `millis()` en `micros()`.
- **timer1** wordt gebruikt voor servo-functies
- **timer2** wordt gebruikt voor o.a.: `tone()`

Deze timers zelf instellen beïnvloed dus deze functies waardoor hun werking niet meer correct verloopt.

9.1.8 Bibliotheek

De bibliotheek `TimerOne.h` vereenvoudigt de code sterk, deze gebruikt zoals de naam doet vermoeden `timer1`. De benodigde prescaler en vergelijkingswaarde worden automatisch ingesteld. We moeten dus zelf geen berekeningen uitvoeren. Het is wel noodzakelijk de interne werking te begrijpen zodat we de maximum tijden en nauwkeurigheid kennen.

Er zijn in de bibliotheek twee belangrijke functies:

- `Timer1.initialize()` stelt de tijd in in μs
- `Timer1.attachInterrupt()` koppelt een ISR (interrupt subroutine) aan de timer

Door het gebruik van een interrupt kan de Arduino in tussentijd andere taken afwerken.

De code hieronder toont een voorbeeld van gebruik van de bibliotheek. De timer wordt geïnitialiseerd op 0,5s en gekoppeld aan de functie `timerIsr()`.

Daardoor wordt om de halve seconde deze functie opgeroepen zodat de LED telkens van toestand verandert.

```

1 #include <TimerOne.h>
2 #define LED1 2
3
4 void setup() {
5   pinMode(LED1, OUTPUT);
6   Timer1.initialize(500000); // 0.5s
7   Timer1.attachInterrupt( timerIsr ); // timerIsr aankoppelen
8 }
9 void loop(){}
10
11 void timerIsr() {
12   digitalWrite(LED1, digitalRead(LED1) ^ 1); // LED togglen

```

9.1.9 Counter

We kunnen een timer ook gebruiken om te tellen (counter). Hiervoor definiëren we een variabele die bij elke interrupt van timer1 geupdated wordt. Dit principe wordt bijvoorbeeld gebruikt om een klok te maken.

De voorbeeldcode hieronder zal een teller elke seconde verhogen en deze waarde via de seriële monitor zichtbaar maken. Merk op dat er een seriële functie gebruikt wordt in de interruptroutine ondanks dat dit wordt afgeraden. Voor maximaal een tweetal karakters levert dit in de praktijk geen problemen op.

```

1 #include <TimerOne.h>
2 int teller =0;
3
4 void setup() {
5     Timer1.initialize(1000000); // 1s
6     Timer1.attachInterrupt( timerIsr );
7     Serial.begin(9600);
8 }
9 void loop() {}
10
11 void timerIsr() {
12     teller++; // teller updaten
13     Serial.println(teller);
14 }
```

9.2 Interne registers

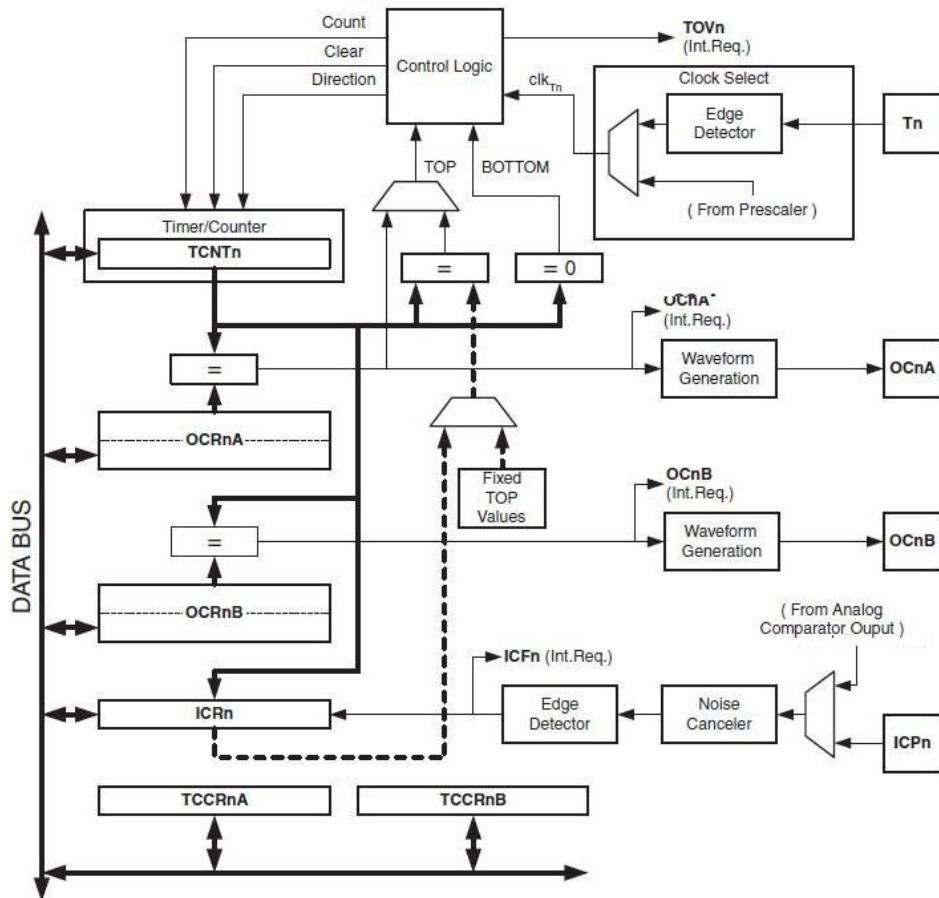
9.2.1 Timer

We kunnen een timer ook zelf instellen in de interne registers, dit laat meer mogelijkheden toe dan dat we de Arduino functies (zoals Timer1.initialize()) gebruiken.

De timer kan in twee modes werken, normale mode en CTC mode. In normale mode loopt de timer over als hij zijn maximumwaarde bereikt (bij timer1 is dit 65535). We kunnen in die mode de tijd enkel aanpassen met de prescalerwaarde en dus niet exact instellen.

In CTC mode (Clear Timer on Compare) bij timer1 wordt bij elke puls de teller TCNT1 met één verhoogd. Wanneer de teller de overloopwaarde OCR1A bereikt dan wordt een interrupt gegenereert en wordt de teller gereset. Door de overloopwaarde aan te passen kunnen we de timer vrij exact instellen (figuur 9.1).

Timer1 bezit twee CTC modes (4 en 12), we gaan in deze cursus er maar één toelichten. De gebruikte methodes zijn voor de andere timers (0 en 2) analoog.



Figuur 9.1: Atmega328 Timer1

Timer1 heeft twee controle registers, TCCR1A (figuur 9.2) en TCCR1B (figuur 9.3). Hierin kunnen we de CTC mode en prescalerwaarde instellen.

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10

Figuur 9.2: TCCR1A controle register

Om dus timer1 te gebruiken die telkens na een gewenste tijd een interrupt genereert moeten we de volgende stappen uitvoeren:

- interne klok gebruiken (CTC mode 4)
- prescaler berekenen en instellen (CS-bits)

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Figuur 9.3: TCCR1B controle register

- overloopwaarde berekenen en instellen (OCR-register)
- interrupts inschakelen

De overloopwaarde (voor timer1) wordt als volgt berekend:

$$OCR1A = \left(\frac{kloksnelheid}{prescalerwaarde} \right) \cdot tijd - 1 \quad (9.1)$$

We willen bijvoorbeeld een tijd van 0,9s. Als we een prescalerwaarde van 1024 kiezen dan bekomen we de volgende waarde voor OCR1A:

$$OCR1A = \left(\frac{16MHz}{1024} \right) \cdot 0,9s - 1 = 14061,5 \quad (9.2)$$

We kunnen enkel gehele getallen gebruiken en moeten dus afronden. Dit geeft als gevolg dat de tijd niet nauwkeurig zal zijn.

Aangezien OCR1A een 16-bits register is mag de maximale waarde 65535 bedragen. We kunnen dus nauwkeuriger werken door een lagere prescaler te gebruiken.

We doen de berekening opnieuw met een prescalerwaarde van 256:

$$OCR1A = \left(\frac{16MHz}{256} \right) \cdot 0,9s - 1 = 56249 \quad (9.3)$$

Nu komen we een geheel getal uit en moeten we niet afronden. De timer zal dus met deze waarden exact werken. Probeer dus steeds de prescalerwaarde zo laag mogelijk te kiezen en zorg dat OCR1A onder de 65535 blijft.

De prescalerwaarde kan ingesteld worden met de CS-bits (figuur 9.4).

CTC mode 4 kan ingesteld worden met de WGM-bits (WGM13:WGM10 = 0100).

```

1 #define LED1 PB5 // pin 13
2
3 void setup() {
4   DDRB |= (1 << LED1); // output
5
6   cli(); // disable interrupts
7   TCCR1A = 0; // register wissen

```

CS12	CS11	CS10	DESCRIPTION
0	0	0	Timer/Counter1 Disabled
0	0	1	No Prescaling
0	1	0	Clock / 8
0	1	1	Clock / 64
1	0	0	Clock / 256
1	0	1	Clock / 1024
1	1	0	External clock source on T1 pin, Clock on Falling edge
1	1	1	External clock source on T1 pin, Clock on rising edge

Figuur 9.4: CS bits

```

8   TCCR1B = 0;                      // register wissen
9   TCNT1  = 0;                      // teller op 0
10  OCR1A = 56249;                  // overloopwaarde (0,9s)
11  TCCR1B |= (1 << WGM12);        // CTC mode 4
12  TCCR1B |= (1 << CS12);         // prescaler 256
13  TIMSK1 |= (1 << OCIE1A);       // interrupt on compare
14  sei();                          // enable interrupts
15
16 void loop(){
17   // leeg
18 }
19
20 ISR (TIMER1_COMPA_vect)          // timer1 overflow interrupt
21 {
22   PORTB ^= (1 << LED1);         // toggle elke 0,9s
23 }
```

9.2.2 Counter

Naast het zelf bijhouden van een teller kunnen de ingebouwde (hardware) timers ook rechtstreeks als counter ingesteld worden. Het voordeel is dat de CPU dan veel minder belast wordt aangezien alles in de hardware gebeurt.

In externe klokmode wordt het aantal pulsen op een bepaalde pin (T0 of T1) geteld.

Om een counter te activeren moeten we het timer register configureren zodat deze als counter werkt en op pulsen reageert. Men kan daarbij kiezen of men telt bij een dalende of stijgende flank.

Als voorbeeld tonen we hoe we het aantal drukken op knop K1 bijhouden in een teller-register, we gebruiken timer0.

```

1 #define K1 PD4 // pin 4, T0
2
3 void setup() {
4   Serial.begin(9600);
5   DDRD &= ~(1<< K1); // K1 input
6   PORTD |= (1<< K1); // pull-up activeren
7
8   TCCR0B |= B00000111; // count op T0, stijgende flank
9 }
10
11 void loop(){
12   Serial.println(TCNT0); // counter register
13 }
```

Het TCCR0B register

Dit is het Timer/Counter 0 Control Register B. De drie LSB (rechtste) bits van dit register laten ons toe om de timer als counter in te stellen (figuur 9.5).

TCCR0A – Timer/Counter0 Control Register A								
Bit	7	6	5	4	3	2	1	0
0x24	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

TCCR0B – Timer/Counter0 Control Register B								
Bit	7	6	5	4	3	2	1	0
0x25	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

Figuur 9.5: Het TCCR0A en TCCR0B register

Om de counter te activeren kunnen we CS02, CS01 en CS00 als volgt instellen:

- 110: externe klok op T0-pin, met count op dalende flank
- 111: externe klok op T0-pin, met count op stijgende flank

Het TCNT0 register

De telling van T0 wordt bijgehouden in het TCNT0 register (figuur 9.6). Dit is een 8-bits register. Wanneer de waarde 255 bereikt wordt dan loopt het register over en wordt de telling opnieuw gestart vanaf 0.

Bit	7	6	5	4	3	2	1	0	TCNT0
	TCNT0[7:0]								
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figuur 9.6: Het TCNT0 register

Bovengrens instellen

We kunnen het moment waarop de teller overloopt aanpassen. Stel dat we na 99 i.p.v. 255 willen herbeginnen dan kunnen we dit instellen. Hiervoor moeten we de *Waveform Generation Mode* bits (WGM02, WGM01 en WGM00) instellen in de juiste registers. Een waarde van 010 in deze bits stelt de comparator-mode in. In het register OCR0A stellen we de overloopwaarde in. De teller zal dan herbeginnen na deze waarde.

```

1 #define K1  PD4    // pin 4, T0
2 #define overloop 99
3
4 void setup() {
5   Serial.begin(9600);
6   DDRD &= ~(1<< K1); // K1 input
7   PORTD |= (1<< K1); // pull-up activeren
8
9   TCCR0B |= B00000111; // count op T0, stijgende flank
10
11  TCCR0B &= ~(1<< WGM02); // WGM02 op 0
12  TCCR0A |= (1<< WGM01); // WGM01 op 1
13  TCCR0A &= ~(1<< WGM00); // WGM00 op 0
14
15  OCR0A = overloop;
16 }
17
18 void loop(){
19   Serial.println(TCNT0); // counter register
20 }
```

9.3 Tijd

De inwendige timers kunnen gebruikt worden om de tijd bij te houden op een Arduino, dit kan gemakkelijk via de bibliotheek *Time*.

De tijd is gebaseerd op het c-datatype: `time_t`. Deze integer houdt normaal het aantal seconden sedert 1/1/1970 bij. De bibliotheek *Time* voorziet zowel functies voor datum en tijd. De synchronisatie kan zowel intern als extern gebeuren.

9.3.1 time_t

Dit datatype is een integer type van 32-bit. Historisch stelt dit het aantal seconden voor sedert 00h00, Jan 1, 1970 UTC, bij sommige toepassingen kan dit echter afwijken. Op computersystemen is dit ook gekend als de UNIX timestamp.

Het `time_t` datatype laat gemakkelijk berekeningen toe (verstreken tijd) aangezien alles in seconden uitgedrukt wordt.

De verstreken tijd sedert 1970 wordt dus gewoon in seconden uitgedrukt:

- 03/05/2022 @ 23:45:00 = 1651614300
- 04/05/2022 @ 00:45:00 = 1651617900

Het verschil van beide = $1651617900 - 1651614300 = 3600\text{s} = 1\text{u}$. Zonder het `time_t` datatype zou je zelf alles moeten omrekenen.

Het voorbeeld hieronder toont hoe je de huidige tijd kunt instellen. In de praktijk is dit voorbeeld moeilijk werkbaar aangezien de tijd geen constante is en voortdurend oploopt.

Na het instellen van de tijd in `setup()` kan in de `loop()` de tijd indien nodig opgevraagd worden. De Arduino houdt die intern bij.

```

1 #include <TimeLib.h>
2
3 time_t tijd = 1651617900;
4
5 void setup() {
6     setTime(tijd);
7     Serial.begin(9600);
8 }
```

De code hieronder toont het opvragen van de tijd elke seconde en toont die via de seriële monitor. De functie `printDigits()` zorgt ervoor dat waarden kleiner dan 10 een voorgaande nul krijgen.

```

1 void loop(){
2     digitalClockDisplay();
3     delay(1000);
4 }
5
6 void digitalClockDisplay(){
7     Serial.print(hour());
8     printDigits(minute());
9     printDigits(second());
10    Serial.print(" ");
11    Serial.print(day());
12    Serial.print(" ");
13    Serial.print(month());
14    Serial.print(" ");
15    Serial.print(year());
16    Serial.println();
17 }
18
19 void printDigits(int digits){
20     Serial.print(":");
21     if(digits < 10)
22         Serial.print("0");
23     Serial.print(digits);
24 }
```

Defening

In de code wordt telkens een ander tijdsverschil berekend. Bepaal wat er zal getoond worden in de seriële monitor.

```

1 time_t tijd1 = 1651614300;
2 time_t tijd2 = 1651618020; // 1u 2min later
3
4 void setup() {
5     setTime(tijd2);
6     Serial.begin(9600);
7     delay(1001);
8     Serial.println(hour(now() - tijd1));
9     Serial.println(minute(now() - tijd1));
10    Serial.println(second(now() - tijd1));
11    Serial.println((now() - tijd1) / 60);
```

¹² } }

9.3.2 Time en timer0

De *Time*-bibliotheek gebruikt de functie `millis()` voor zijn inwendige telling. De functie `millis()` gebruikt op zijn beurt *timer0*. Wanneer we zelf deze timer instellen kan dit dus de tijd beïnvloeden.

9.3.3 Schrikkeljaar en weekdag

De *Time*-bibliotheek berekent automatisch schrikkeljaren, als we in de interne code kijken van de bibliotheek dan zien we het volgende:

```
1 #define LEAP_YEAR(Y) ((((1970+Y)>0) && !((1970+Y)%4) && ( →  
→((1970+Y)%100) || !((1970+Y)%400) ) )
```

Er wordt onder andere nagegaan of een jaar deelbaar is door 4, door 100 en door 400. De juiste combinatie van de rest van deze delingen bepaalt al dan niet of een jaar een schrikkeljaar is.

Ook de weekdag en de maand worden correct getoond (hier uitgevoerd op een woensdag in april):

```
1 Serial.println(weekday()); // 4  
2 Serial.println(dayStr(weekday())); // Wednesday  
3 Serial.println(monthStr(month())); // April
```

Deze strings kunnen aangepast worden in het bestand `DateStrings.cpp` van de bibliotheek en bijvoorbeeld vertaald worden naar het Nederlands.

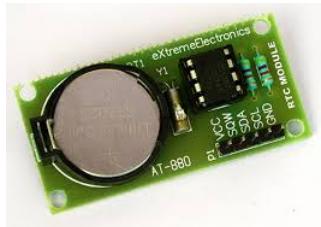
9.3.4 Externe synchronisatie

De klok van een Arduino loopt ondanks het gebruik van een kristal toch nog enkele seconden verkeerd per maand. Het is ook een belastende taak om de interne klok van een Arduino te gebruiken om de tijd bij te houden, er blijft dan nog weinig rekenkracht over om andere taken uit te voeren.

RTC

Het is dus nauwkeuriger en minder belastend als men een externe klok gebruikt. De tijd kan bijvoorbeeld bijgehouden worden in een RTC (Real Time Clock), dit is een speciaal IC die de tijd bijhoudt en dat kan uitgelezen worden wanneer het nodig is. De module (figuur 9.7) is

meestal ook voorzien van een batterij zodat de tijd correct blijft verder lopen bij een eventuele stroomuitval.



Figuur 9.7: RTC module

Het codevoorbeeld *TimeRTC* synchroniseert met de RTC klok op het clock shield. Om het RTC-IC te kunnen aanspreken is een extra bibliotheek nodig, namelijk *DS1307RTC*.

Deze bibliotheek voorziet I²C-communicatie met het IC en gebruikt de standaard I²C-poort van de Arduino, er moeten dus geen pinnummers insteld worden vooraleer het voorbeeld kan gebruikt worden.

```

1 #include <TimeLib.h>
2 #include <Wire.h>
3 #include <DS1307RTC.h>
4
5 void setup() {
6     Serial.begin(9600);
7     setSyncProvider(RTC.get); // get time from RTC
8     if(timeStatus()!= timeSet)
9         Serial.println("Geen sync met RTC");
10    else
11        Serial.println("Sync met RTC OK");
12 }
```

De tijd instellen kan door de rechtstreeks de juiste waarde in `time_t` te plaatsen, maar het is handiger als dit automatisch gebeurd door de compiler. Bij het compileren wordt de tijd opgevraagd van de computer en ingesteld op de Arduino.

Het voorbeeld *SetTime* uit de bibliotheek *DS1307RTC* stelt de tijd (eenmalig) gelijk aan computertijd.

```

1 // get the date and time the compiler was run
2 if (getDate(__DATE__) && getTime(__TIME__)) {
3     parse = true;
4     // and configure the RTC with this info
5     if (RTC.write(tm)) {
6         config = true;
```

```
7     }  
8 }
```

9.3.5 Hoge nauwkeurigheid

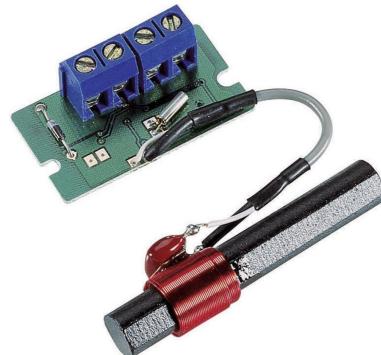
Ook een RTC maakt gebruik van een kristal en blijft dus niet eeuwig correct lopen. Om toch de exacte tijd te kennen kunnen we een externe bron gebruiken die de RTC af en toe van een tijdscorrectie voorziet.

Een aantal mogelijke externe bronnen zijn: DCF77, GPS, RDS en NTP.

DCF77

Dit is een atoomklok in Frankfurt die de tijd via een radiosignaal uitzendt. De DCF-ontvanger (figuur 9.8) genereert een signaal dat nog moet gedecodeerd worden door de Arduino. Elke minuut wordt de tijd aan de hand van een pulsentrein doorgestuurd.

De pulsen moeten dus gedetecteerd worden en omgezet worden naar de juiste tijd, dit verbruikt vrij veel resources op de Arduino.



Figuur 9.8: DCF-ontvanger

GPS

GPS wordt gebruikt om de positie van voornamelijk voertuigen te bepalen. Het GPS-signaal stuurt naast de locatie ook de tijd door.

Een GPS-ontvanger (figuur 9.9) levert typisch zijn informatie via een seriële kanaal, de Arduino hoeft enkel de tijdsinformatie eruit te filteren en wordt dus niet zwaar belast. Een nadeel is dat het enkel buiten werkt.

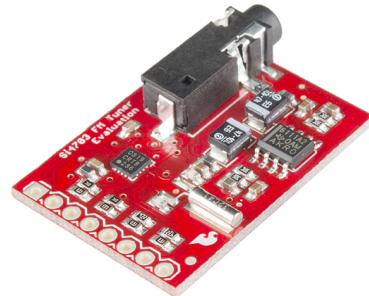


Figuur 9.9: GPS-ontvanger

RDS

De meeste radiozenders op de FM-band sturen extra informatie door a.d.h.v. een RDS-signaal wat staat voor Radio Data System.

De RDS-ontvanger is een onderdeel van een (digitale) FM-ontvanger (figuur 9.10) en levert zijn informatie meestal via een I²C-poort.

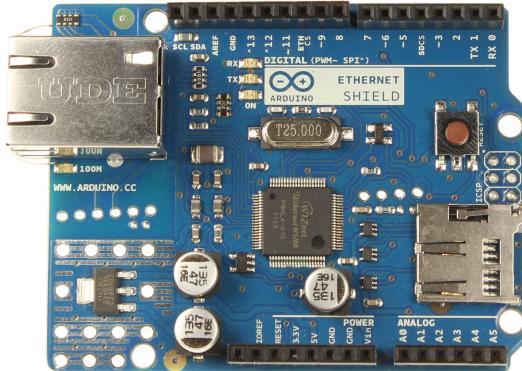


Figuur 9.10: FM-ontvanger

NTP

NTP staat voor Network Time Protocol. De tijd wordt dus door dit protocol aangeleverd via het netwerk en wordt opgevraagd aan een NTP-server. Dit kan via een ethernet- of wifi-verbinding.

Een Arduino kan het netwerk benaderen via een ethernet shield (figuur 9.11) of een wifi shield.



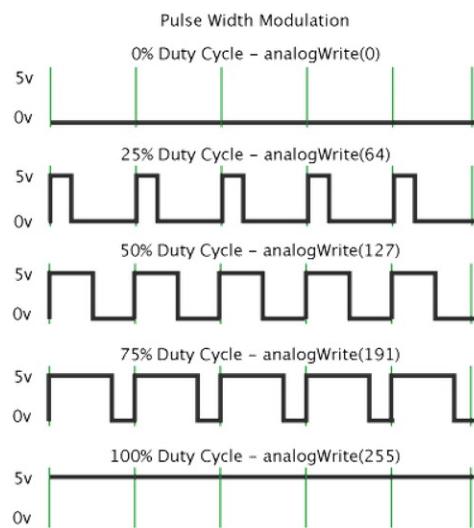
Figuur 9.11: Ethernet shield

9.4 PWM

Met `digitalWrite()` werden tot nu toe pinnen hoog of laag gemaakt. De spanning is dus ofwel hoog of laag. Voor sommige toepassingen willen we echter de spanning kunnen instellen tussen hoog en laag.

Op bepaalde pinnen kunnen we de uitgangsspanning variëren, dit kan met behulp van *Pulse Width Modulation (PWM)*. Bij deze techniek wordt de gebruikte pin zeer snel in de tijd aan en uit gezet.

Door de breedte van de pulsen te variëren (figuur 9.12) zal de gemiddelde spanning variëren, PWM kan bv. gebruikt worden om een LED te dimmen of de snelheid van een motor te regelen.



Figuur 9.12: PWM dutycycles

9.4.1 analogWrite()

De functie `analogWrite()` laat toe om PWM te gebruiken op een Arduino. De mogelijke pinnen die PWM toelaten op een Arduino UNO zijn 3,5,6,9,10 en 11. Je kan deze herkennen aan het golfje (~) dat er bij staat.

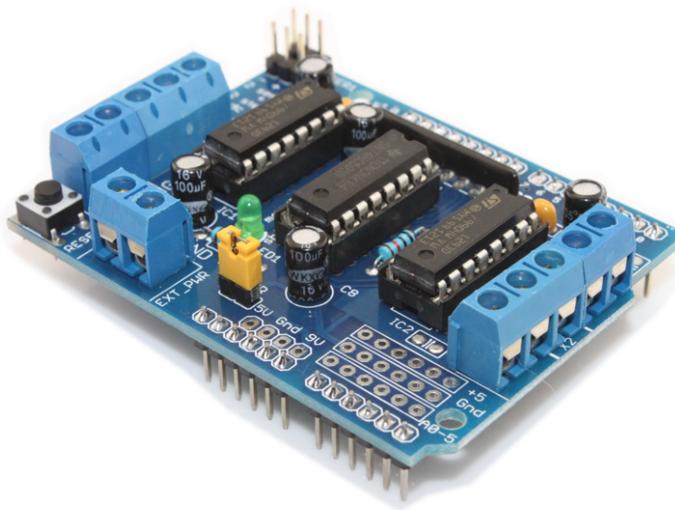
De PWM-frequenties zijn 490Hz of 980Hz, dit is meer dan voldoende voor een LED (het menselijk oog is veel trager) en een DC-motor.

De timers die door PWM gebruikt worden zijn de volgende:

- timer0: pin 5 en 6
- timer1: pin 9 en 10
- timer2: pin 3 en 11

Als op bepaalde pinnen PWM ingeschakeld wordt, dan kunnen de bijhorende timers niet meer voor andere doeleinden gebruikt worden.

9.4.2 Nauwkeurigheid



Figuur 9.13: Motorshield

De resolutie die `analogWrite()` gebruikt is 8-bit, onafhankelijk van de gebruikte timer. De duty-cycle kan dus ingesteld worden tussen 0 en 255.

Voor de meeste toepassingen is dit voldoende. De resolutie kan eventueel aangepast worden door de nodige bit-instellingen uit te voeren.

Een shield dat gebruik maakt van PWM is het motorshield (figuur 9.13). Dit shield laat toe om 4 DC-motoren aan te sturen en de snelheid ervan te regelen. Aangezien alle timers gebruikt worden is het quasi onmogelijk om andere tijdsgerelateerde zaken uit te voeren zoals bijvoorbeeld het toevoegen van een IR-ontvanger.

9.5 pulseIn()

Deze functie meet de lengte van inkomende pulsen op een pin. Dit kan zowel voor hoge als lage toestand bepaald worden. De functie geeft de tijd terug in microseconden.

Het gebruik van pulseIn() is handig om sensoren en signalen uit te meten die op basis van pulslengetes werken. Dit kan bijvoorbeeld een DCF77-ontvanger of draadloos weerstation zijn.

```
int pin = 7;
unsigned long duration;

void setup() {
    pinMode(pin, INPUT);
}

void loop() {
    duration = pulseIn(pin, HIGH);
}
```


10

Analoge signalen

De Arduino en ESP8266 kunnen ook analoge signalen inlezen, deze moeten echter omgezet worden naar digitale waarden. Dit gebeurt a.d.h.v. een ADC (analoog-digitaal-convertisor).

Voor de gebruiker is er een bibliotheek ter beschikking. Het is echter belangrijk om de werking van de ADC te begrijpen om tekortkomingen te kennen.

10.1 Soorten ADC's

10.1.1 De ideale ADC

Hierbij wordt iedere mogelijke analoge ingangswaarde naar een unieke digitale waarde vertaald. De ideale ADC (figuur 10.1) heeft een oneindige resolutie en is enkel theoretisch mogelijk.

10.1.2 De perfecte ADC

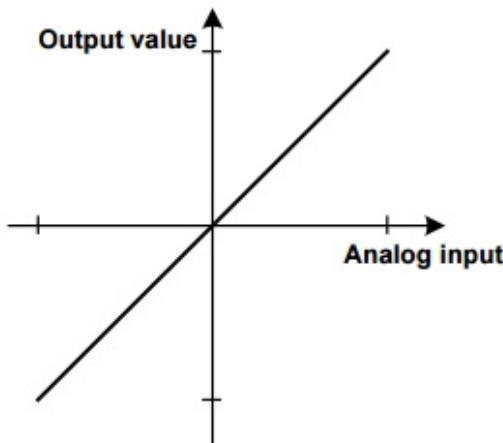
Bij de perfecte ADC (figuur 10.2) is digitaal niet elke waarde mogelijk. Het digitale bereik wordt verdeeld in stappen. De analoge waarde wordt gefit in een trap.

Bij een 10-bit resolutie (zoals bv. bij de Arduino UNO) zijn er 1024 mogelijke trappen. De waarde wordt telkens afgerond naar de dichtsbijzijnde digitale waarde.

De maximale fout die kan optreden is $1/2$ LSB, dit noemt men ook de quantization error. In het midden van elke trap is de perfecte ADC gelijk aan de ideale ADC. Bij echte ADC's treden echter nog andere fouten op. Dit wordt verder besproken.

10.1.3 ADC's in AVR controllers

Er zijn twee mogelijkheden: single-ended en differential (niet bij de Uno) ADC's.



Figuur 10.1: Ideale ADC

Bij een single-ended ADC wordt de ingangsspanning vergeleken met GND. De digitale output ligt tussen 0 en 1023 (10-bit).

Bij een differential ADC worden 2 ingangswaarden van elkaar afgetrokken, de digitale output ligt tussen -512 en 511 (negatieve waarden zijn dus mogelijk). De 2 ingangsspanningen zelf moeten positief zijn t.o.v. de GND.

Referentiespanning

De ingangsspanning wordt vergeleken met A_{REF} , op de Arduino kan men dit instellen met `analogReference()`.

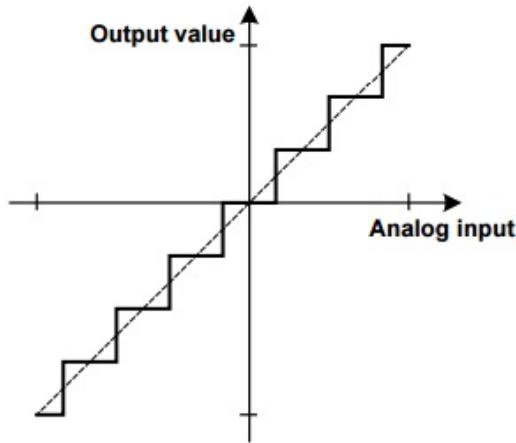
Dit zijn de mogelijke instellingen:

- DEFAULT: 5V
- INTERNAL: 1,1V op Uno
- EXTERNAL: spanning op AREF pin (ts. 0 en 5V)

10.2 Mogelijke fouten

10.2.1 Offset error

De offset error (figuur 10.3) is de afwijking van de transferfunctie t.o.v. ideale lijn. Dit kan gemeten worden door gekende analoge waarden te testen.



Figuur 10.2: Perfecte ADC

10.2.2 Gain error

Dit is de afwijking van de maximum waarde na offset compensatie en kan terug uitgemeten worden (figuur 10.4).

10.2.3 Non-Linearity

Na een offset en gain-compensatie zou de transferfunctie perfect moeten zijn. Er zijn echter nog steeds afwijkingen door de niet lineariteit van de ADC (figuur 10.5). Deze fout kan moeilijk uitgemeten en gecompenseerd worden.

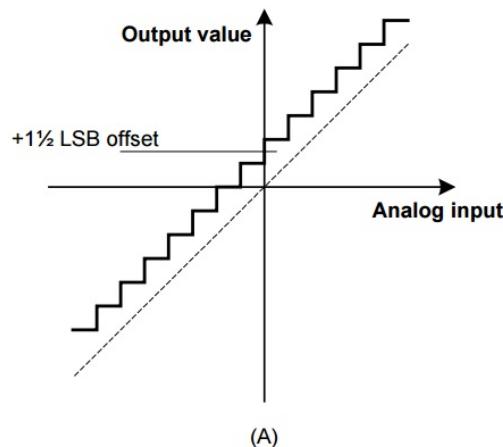
10.2.4 Andere fouten

De referentiespanning moet stabiel en exact zijn. De nauwkeurigheid is ook afhankelijk van de snelheid van omzetting: hoe sneller, hoe onnauwkeuriger de ADC zal zijn.

10.3 Theorema van Nyquist

Bij het omzetten van een analoog signaal naar een digitale waarde moet de bemonsteringsfrequentie volgens Nyquist tweemaal zo hoog zijn dan de hoogste frequentie in het signaal. Dit om het origineel zonder fouten te kunnen reproduceren.

Bij een sinussignaal van 1kHz moet men dus minimaal samplen met 2kHz. Een blokgolf (die ook harmonischen bevat) van 1kHz moet minimaal met 10kHz gesampled worden.



Figuur 10.3: Offset error

10.3.1 ADC bandbreedte

De snelheid van conversie wordt beperkt door de ADC klok. Bij een Arduino UNO neemt één conversie 13 klokpulsen in beslag.

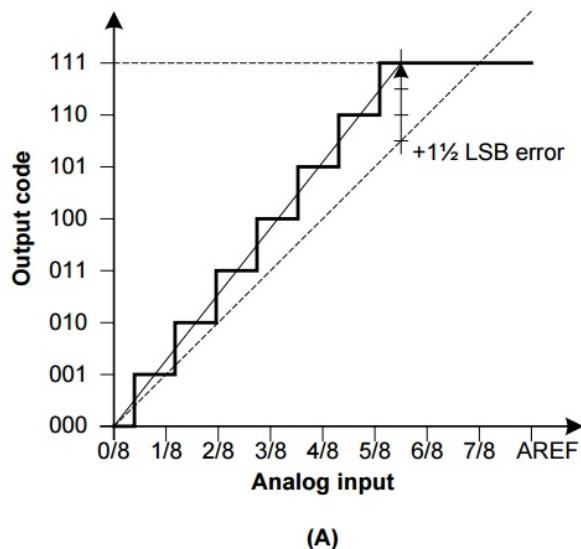
De maximale ADC klok bedraagt 1MHz wat dus 77000 samples/seconde betekent. Volgens Nyquist is de maximale frequentie van een sinussignaal bij deze ADC dus 38,5 kHz. Dit is dus een heel stuk lager dan de oorspronkelijke ADC klok.

Arduino UNO ADC

De standaard prescaler van de ADC bedraagt 128. Bij een processorfrequentie van 16Mhz bedraagt de ADC klok dan 125kHz ($16\text{MHz}/128$). Een ADC klok van 125kHz laat 9600 samples/seconde toe ($125\text{kHz} / 13 \text{ klokpulsen}$).

Dit geeft als gevolg dat de standaard maximale ingangs frequentie bij een sinus 4800Hz bedraagt.

Een ander gevolg is dat de functie `analogRead()` $\pm 104 \mu\text{s}$ duurt. De snelheid kan verhoogd worden ten koste van de nauwkeurigheid.



Figuur 10.4: Gain error

10.4 Praktisch gebruik van de ADC op de Arduino UNO en ESP8266

10.4.1 Arduino UNO

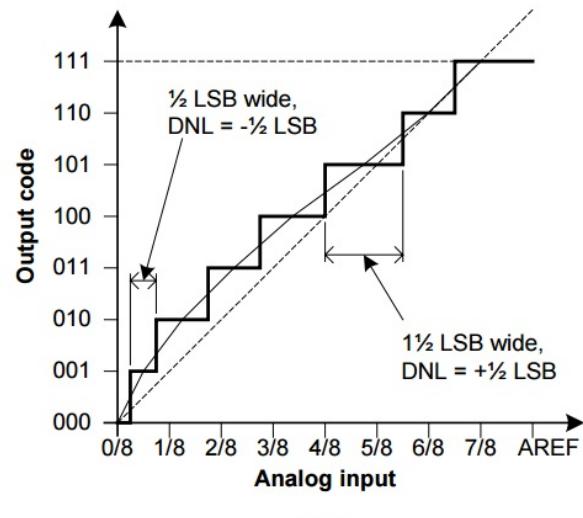
Er zijn zes analoge ingangen op de UNO (pinnen A0 t.e.m. A5) die een maximale spanning van 5V aankunnen. Deze kunnen worden uitlezen met de functie `analogRead()`. Dit geeft een digitale waarde terug tussen 0 (0v) en 1023 (5V). Hieronder zie je een voorbeeld.

```

1 int analogPin = A0;
2 int input;
3 input = analogRead(analogPin);
4 Serial.println(input);
```

10.4.2 ESP8266

De ESP8266 chip heeft één analoge ingang van 0 tot 1V. De meeste ontwikkelbordjes (zoals de NodeMCU) hebben echter een spanningsdeler aan boord zodat de maximale spanning op de analoge ingang 3,3V bedraagt. In figuur 10.6 zie je duidelijk de weerstanden van de spanningsdeler naast de A0 pin. De resolutie is net zoals bij de Arduino UNO 10-bit.



(A)

Figuur 10.5: Non-Linearity

10.4.3 Analoge sensors

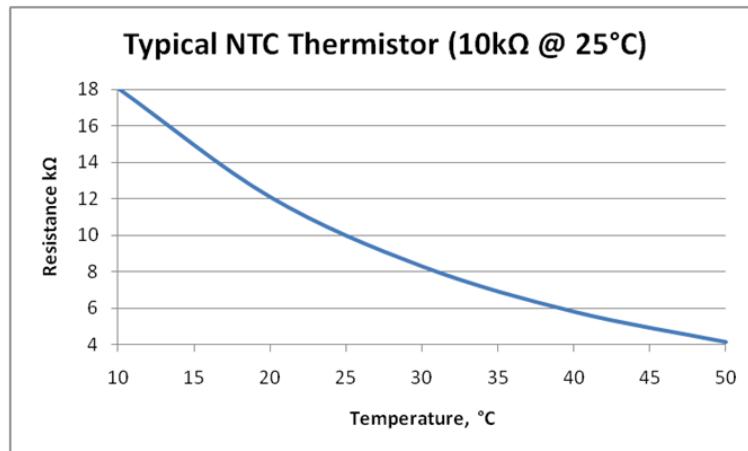
Deze sensors zetten temperatuur, licht, ... om in een analoge waarde. Ze zijn meestal eenvoudig en goedkoop. Daarentegen zijn ze meestal niet lineair en enkel geschikt voor minder nauwkeurige zaken.



Figuur 10.6: Analoge ingang NodeMCU

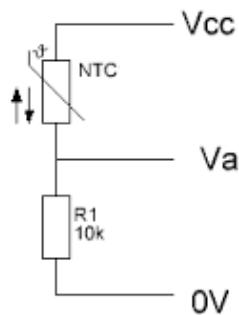
Temperatuursensor

Als analoge sensor wordt voor temperatuursmetingen meestal een NTC weerstand gebruikt. De weerstand van een NTC daalt bij een toenemende temperatuur (figuur 10.7). Het verband tussen weerstand en temperatuur is logaritmisch.



Figuur 10.7: NTC karakteristiek

De uitgangswaarde van een NTC weerstand in serie met een gewone weerstand van $10\text{k}\Omega$ (R_1) kan men berekenen via een weerstandsdeling (figuur 10.8).



Figuur 10.8: Weerstandsdeling

De gemeten temperatuur is dan als volgt:

$$R_{NTC} = \frac{(1023 - a) * 10\text{k}\Omega}{a} \quad (10.1)$$

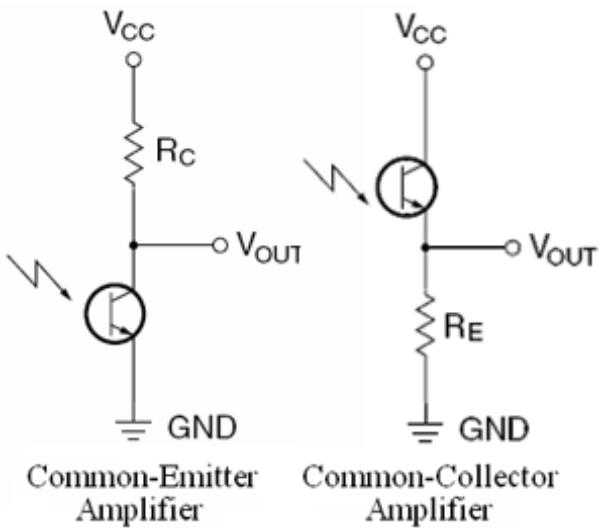
$$T(^{\circ}C) = \frac{1}{\log\left(\frac{R_{NTC}}{10k\Omega}\right) + \frac{1}{3975}} - 273,15 \quad (10.2)$$

Oefening

- gegeven een temperatuur van $40^{\circ}C$
- bepaal uit voorgaande grafiek de weerstandswaarde van de NTC
- de sensor wordt aangesloten zoals op voorgaand schema
- bereken de digitale waarde die zal bekomen worden na de ADC-conversie

Lichtsensor

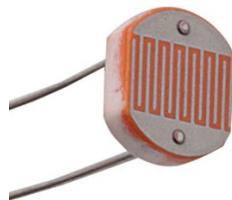
Om licht te meten met een analoge sensor zijn er verschillende mogelijkheden. Meestal wordt een fototransistor (figuur 10.9) of LDR (lichtgevoelige weerstand) gebruikt.



Figuur 10.9: Fototransistor

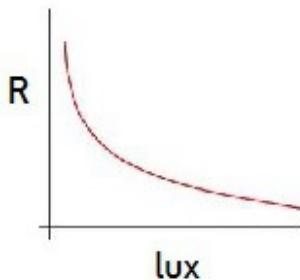
De LDR (figuur 10.10) of licht afhankelijke weerstand is minder nauwkeurig dan een fototransistor en is ideaal voor minder nauwkeurige zaken. Bijvoorbeeld het bedienen van rolluiken of het dimmen van lichten.

De weerstandswaarde van een LDR is een logaritmische functie zoals bij de NTC (figuur 10.11). Er zal een (veel) hogere weerstand zijn bij minder licht (bv. $200k\Omega$). Bij volle lichtintensiteit



Figuur 10.10: LDR

is er een lage waarde (bv. $1k\Omega$). De exacte waarde is minder belangrijk, meestal bepaalt men enkel de grenzen.



Figuur 10.11: LDR karakteristiek

De formule om de weerstand te berekenen in het schema zoals in figuur 10.12 is als volgt:

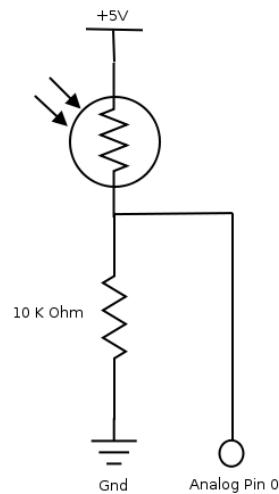
$$R_{LDR} = \frac{(1023 - a) * 10k\Omega}{a} \quad (10.3)$$

10.5 Analog output

De meeste Arduino's beschikken enkel over PWM functionaliteit en niet over een DAC (digitaal analoog convertor).

De Arduino DUE bevat naast PWM ook 2 DAC's. Deze pinnen kunnen analoge signalen genereren. De maximale resolutie van de DAC is 12-bit (6mA, 0-2,8V). Er kunnen dus 4096 mogelijke spanningsniveau's gegenereerd worden.

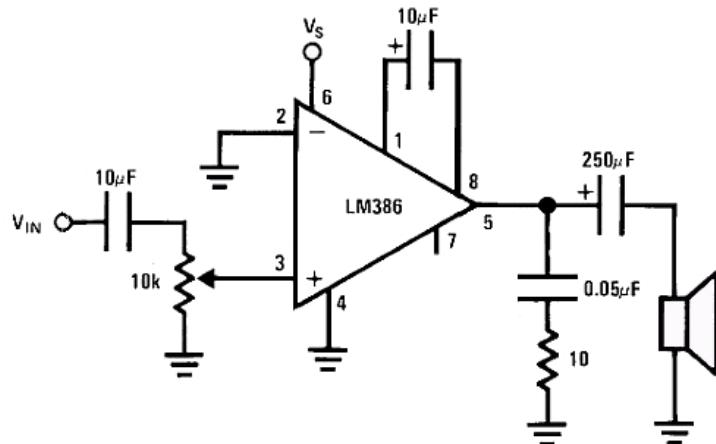
De DAC's van de Arduino DUE worden ook aangestuurd met `analogWrite()`, deze functie wordt ook gebruikt om PWM-pinnen aan te sturen.



Figuur 10.12: Weerstandsdeling

10.5.1 Audio player

Een DAC geneert een signaal met een DC offset, dit kan men wegfilteren met een condensator. Vervolgens kan men het signaal versterken om een luidspreker aan te sturen (figuur 10.13).



Figuur 10.13: Audioplayer

11

Communicatieprotocollen

11.1 Seriële poort

Iedere Arduino en ESP bevat minstens één (hardware) seriële poort. Bij de UNO is deze verbonden met pin 0 (RX) en pin 1 (TX). Deze pinnen kunnen niet als digitale pin gebruikt worden als de seriële poort gebruikt wordt. De grotere Arduino MEGA bevat vier seriële poorten.

Het voordeel van de seriële poort is dat er minder I/O-pinnen nodig zijn dan bij een parallele poort. De kabels en connectoren zijn hierdoor eenvoudiger en goedkoper aangezien er minder geleiders nodig zijn.

Het nadeel is dan weer dat de seriële poort trager is bij gelijke hardware.

11.1.1 Protocol

De seriële poort gebruikt een asynchrone transmissie en heeft dus geen apart kloksignaal. Dit heeft als voordeel dat er een lijn minder nodig is en als nadeel dat zender en ontvanger dezelfde instellingen moeten hebben.

Door de asynchrone werking moet ook het start- en stoptijdstip achterhaald worden (d.m.v. start- en stopbits).

RS232

RS232 is de naam van de aansluiting op een PC (figuur 11.1), oorspronkelijk bedoeld voor terminals en modems. Deze aansluiting bevat naast RX en TX nog andere signalen, zoals RTS (request to send).

De RS232 aansluiting gebruikte oorspronkelijk spanningen tussen -25V en +25V. Je mag dit dus niet rechtstreeks verbinden met een microcontroller.

De aansluiting is zo goed als uitgestorven in deze vorm, maar het seriële protocol wordt wel nog veel gebruikt bij bijvoorbeeld USB, bluetooth en sensoren.



Figuur 11.1: RS232

USART

De USART is een stukje extra hardware in de microcontroller die toelaat om seriëel te communiceren. Het voorziet het protocol (timing), een buffer, enz.

Door de extra hardware wordt de eigenlijke CPU ontlast. In een PC is er een apart IC naast de CPU voor de USART functionaliteit.

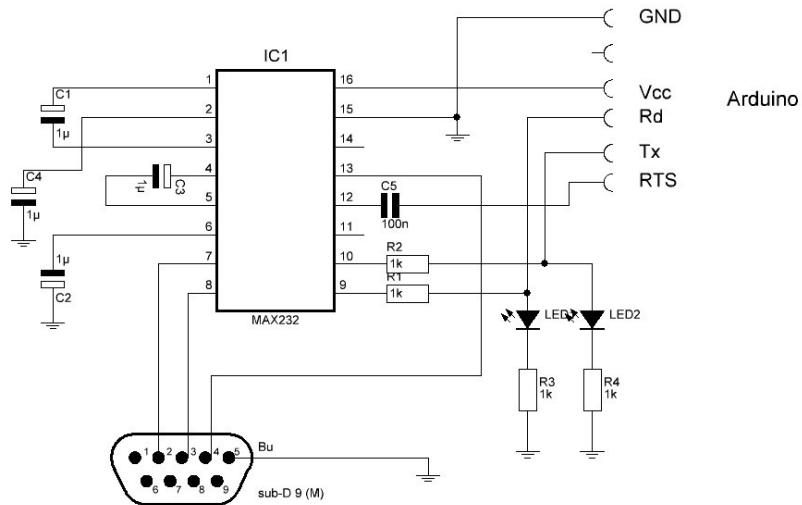
Spanningsniveau's

Een RS232 aansluiting gebruikt spanningen tussen -25V en +25V waarbij een waarde tussen -3V en +3V ongedefinieerd is. De USART daarentegen gebruikt spanningen van 0V en 5V of zelfs 0V en 3,3V. Een conversie is dus nodig tussen beide, men mag dus nooit beide rechtstreeks verbinden.

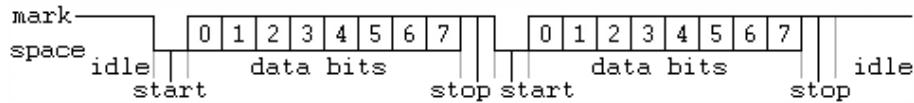
Een oplossing is het MAX232 IC, dit IC converteert de spanningen in beide richtingen tussen RS232 en USART (figuur 11.2).

Bits

Naast de databits worden er ook start- en stopbits verstuurd. Daarnaast is er ook al of niet een parity bit (even of oneven). Het aantal databits kunnen er vijf tot negen zijn, meestal zijn het er acht (figuur 11.3).



Figuur 11.2: MAX232



Figuur 11.3: Bits seriële poort

Baudrate

De gebruikte snelheid is belangrijk bij een asynchrone communicatie, de zender en ontvanger moeten dus gelijk zijn ingesteld en nauwkeurig zijn.

De standaardinstellingen van een seriële poort zijn als volgt:

- baudrate: 9600
- databits: 8
- stopbits: 1
- parity: geen

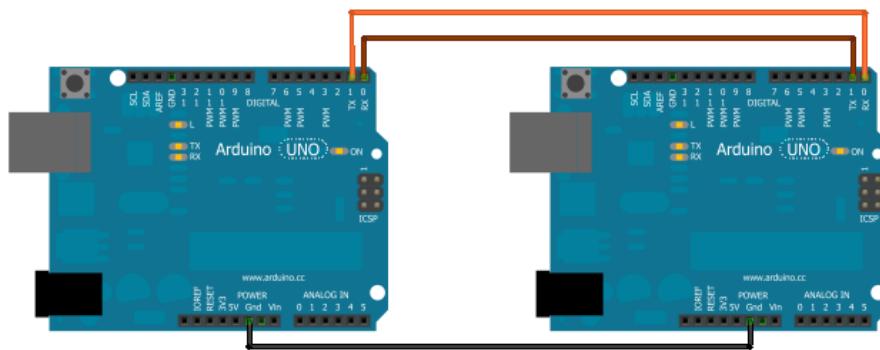
11.1.2 Gebruik op de Arduino

De Arduino is verbonden via een USB-poort met de PC en ondersteunt seriële communicatie. De USART van de microcontroller is namelijk verbonden met een extra IC (Atmega16U2). Dit IC is geprogrammeerd als een seriël naar USB omzetter.

Sommige Arduino's bevatten dit IC niet (bv. de Arduino Ethernet) en hebben een extra module nodig voor seriële communicatie en dus ook voor het laden van code. Nieuwere Arduino modellen hebben een microcontroller met ingebouwde USB-mogelijkheden.

Twee arduino's verbinden

Twee arduino's kunnen gemakkelijk met elkaar communiceren via de seriële poort, het volstaat om de verbinding te kruisen (RX met TX en TX met RX verbinden, zie figuur 11.4).



Figuur 11.4: Arduino's serieel verbonden

Verbinden met een GPS

Om een sensor zoals een GPS-ontvanger aan te sluiten moet men de voedingsspanningen (let op 3,3V en 5V) en de datalijnen RX en TX verbinden (figuur 11.5).

De baudrate van de GPS en eventuele andere instellingen kan men opzoeken in de datasheet.

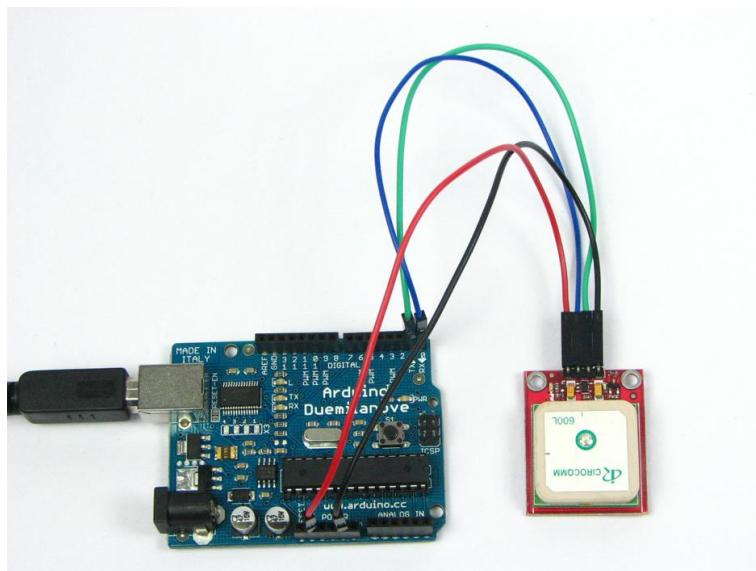
Na een correcte aansluiting kan men op de Arduino de karakters ontvangen en ontleden.

11.1.3 Code

Initialisatie

De functie `Serial.begin()` laat toe om op een Arduino een poort te openen en de instellingen te kiezen. Zo zal `Serial.begin(9600)` de standaardinstellingen gebruiken.

Wil je de instellingen aanpassen, dan kan er een tweede parameter meegegeven worden, zo zal `Serial.begin(115200,SERIAL_7E2)` een hogere snelheid, 7 databits, even pariteit en 2 stopbits gebruiken. Dit is soms nodig voor exotische sensors.



Figuur 11.5: GPS verbinding

Karakters ontvangen

Om karakters te ontvangen op een Arduino moeten er twee taken vervuld worden. Als eerste moet men wachten op de te ontvangen karakters, pas als er karakters ontvangen worden kan men deze inlezen.

De gebruikte functies daarvoor zijn: `Serial.available()` en `Serial.read()`. Er is tevens een buffer ter beschikking zodat geen karakters verloren gaan.

Hieronder vind je een eenvoudig voorbeeld:

```
1 void setup() {
2     Serial.begin(9600); // open poort
3 }
4
5 void loop() {
6     if (Serial.available() > 0) { // bytes beschikbaar?
7         incomingByte = Serial.read(); // lezen
8         Serial.print("I received: ");
9         Serial.println(incomingByte, DEC);
10    }
11 }
```

Getallen ontvangen

Er zijn een aantal functies beschikbaar die het inlezen van getallen eenvoudiger maken.

Met `Serial.parseInt()` kan je getallen op de Arduino ontvangen, het ontvangen van karakters gaat door zolang er cijfers verstuurd worden en stopt bij een niet numeriek karakter.

Daarna worden de cijfers omgezet volgens de juiste machten van tien, de functie bezit een timeout van ongeveer één seconde.

Verder zijn er nog de functies `parseFloat()` en `readBytesUntil()`.

Inkomende buffer wissen

Wanneer men veel karakters ontvangt en niet alle karakters nodig zijn, is het soms nodig om de seriële buffer te ledigen. Met een eenvoudige while-lus kan dit uitgevoerd worden:

```
1 while(Serial.available()) {  
2     Serial.read();  
3 }
```

De lus loopt zolang er bytes beschikbaar zijn.

11.1.4 Software serial

Bij de Arduino Uno is er slechts één hardware seriële poort aanwezig. Soms zijn er meerdere poorten nodig of wenst men een poort op andere pinnen dan pin 0 en 1.

De oplossing is om softwarematig het seriële protocol te implementeren. Dit kan met de bibliotheek `SoftwareSerial`.

Beperkingen

Een software seriële poort verbruikt resources van de CPU, wanneer meerdere softwarepoorten gebruikt worden kan slechts één tegelijk zenden of ontvangen. De hardware- en een software-poort kunnen wel door elkaar gebruikt worden.

Bij sommige Arduino's kan de software seriële poort niet gebruikt worden op alle I/O-pinnen.

Code voorbeeld

```
1 // bibliotheek importeren  
2 #include <SoftwareSerial.h>  
3
```

```

4 // pinnen definieren
5 SoftwareSerial mySerial(10, 11); // RX, TX
6
7 void setup() {
8 // zelfde werking als hardware poort
9   mySerial.begin(4800);
10  mySerial.println("Hello world");
11 }

```

11.2 I²C

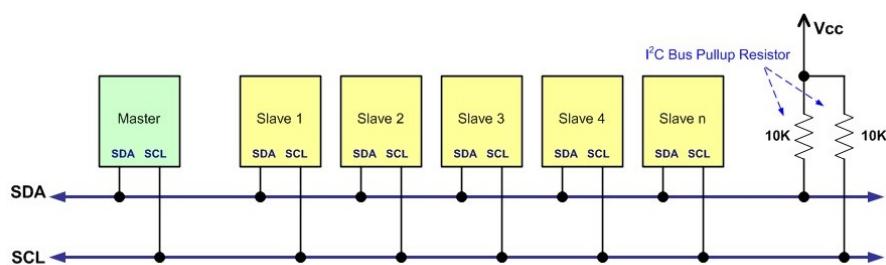
Dit is een protocol voor communicatie tussen IC's en staat voor Inter-IC. Het werd ontwikkeld door Philips (nu NXP) en wordt veel gebruikt in consumentenelektronica.

Het protocol wordt ondersteund door veel microcontrollers.

11.2.1 De I²C-bus

De I²C-bus is een synchrone seriële tweedraads-verbinding tussen een master en (meerdere) slave(s). Het kloksignaal dat steeds door de master gegenereerd wordt is beschikbaar op de SCL-lijn (Serial CLock). Het datasignaal dat op de SDA-lijn (Serial DAta) geplaatst wordt gaat van master naar slave of omgekeerd.

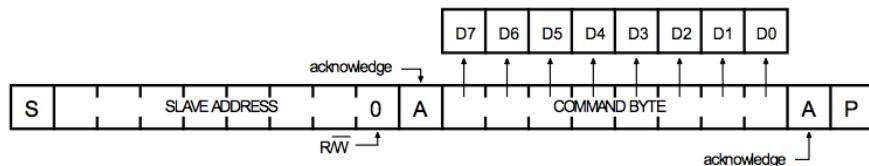
De meest gebruikte opstelling is één master en meerdere slaves (figuur 11.6). Op de SCL- en SDA-lijn zijn normaal gezien pull-up weerstanden nodig, bij een Arduino zijn die ingebouwd en niet meer nodig.



Figuur 11.6: I²C-bus

11.2.2 Adressering

Aangezien alle IC's op dezelfde bus geplaatst worden is er een adressering nodig. Normaal wordt een 7-bit adres gebruikt, soms is dit 10-bit. Standaard zijn er dus 128 adressen mogelijk. Het achtste bit van het adres is de R/W-bit (figuur 11.7).



Figuur 11.7: Adressering I²C

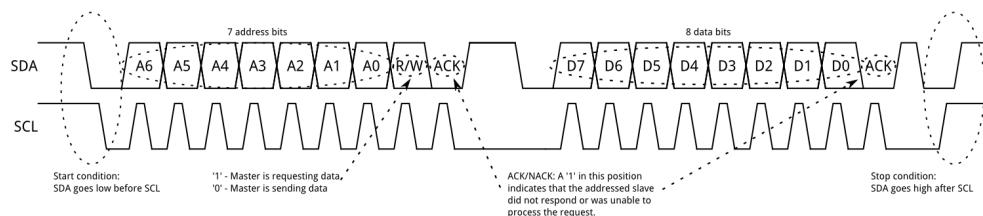
11.2.3 Kloksignaal en timing data

Het kloksignaal wordt gegenereerd door de master en bedraagt meestal 100kHz (standard) of 400kHz (fast). Dit moet niet exact zijn, de slave moet het wel aankunnen. Er zijn bij sommige IC's ook hogere snelheden mogelijk: 1MHz (fast plus) of 3,4MHz (high speed).

Vooraleer er data verstuurd wordt controleert de master eerst de spanning op SCL en SDA, dit om botsingen te vermijden. Als de SCL- en SDA-lijn hoog zijn kan de communicatie starten. De pull-up weerstanden zijn dus nodig om de lijnen hoog te houden.

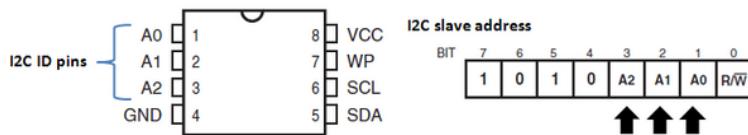
Vervolgens maakt de master de SDA-lijn laag en daarna is de master baas over de bus. Dit wordt door de slaves herkend als startsignaal.

Het I²C-protocol bepaalt dat na de start het adres verstuurd wordt. Dit wordt door de geadresseerde slave bevestigd met een ACK. Hierna kan de eigenlijke data verstuurd worden (figuur 11.8).



Figuur 11.8: Kloksignaal I²C

Elke slave moet een uniek adres hebben, bij het gebruik van twee of meer identieke componenten (bv. verschillende temperatuursensoren) moet er een mogelijkheid zijn om het adres van de slave in te stellen. Meestal zal een deel van de 7 bits van het adres vast zijn (bv. 4-bits), de andere zijn dan instelbaar (bv. 3-bits of 8 mogelijkheden, zie figuur 11.9).



Figuur 11.9: Instellen adres

Lezen en schrijven

Bij het schrijven door de master stuurt de master het adres (met het R/W-bit op 0), de slave bevestigt met een ACK. Vervolgens stuurt de master de data en zal de slave terug bevestigen met een ACK.

Als de master wilt lezen dan stuurt hij het adres (met het R/W-bit op 1), de slave bevestigt met een ACK. Daarna stuurt de slave de data en bevestigt de master met een ACK.

De klok wordt steeds door de master gegenereerd.

I²C en Arduino

I²C wordt ondersteund door de *Wire*-bibliotheek op het Arduino platform. Deze bevat alle benodigde functies: `setup()`, `beginTransmission()`, `write()`, `available()`, `read()`, ...

De Aduino UNO bevat één I²C-interface (pinnen A4 (SDA) en A5 (SCL)), vanaf de UNO R3-versie zijn de pinnen te vinden naast AREF. Net zoals bij de seriële poort bestaat er ook een software I²C-bibliotheek met ongeveer dezelfde voor- en nadelen.

Ook de ESP8266 bevat een hardware I²C-interface.

11.2.4 Praktische voorbeelden

Poortuitbreiding

Wanneer extra pinnen nodig zijn kan men deze uitbreiden met een poortuitbreidings-IC. Dit om bijvoorbeeld een LCD via I²C aan te sluiten.

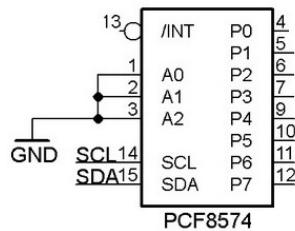
Een voorbeeld is de PCF8574 port expander, deze voorziet 8-datapinnen (figuur 11.10). Er kunnen maximum 8-IC's op één bus geplaatst worden.

We nemen een looplicht met 3x een PCF8574 als praktisch voorbeeld. Op die manier kunnen er 24 LED's verbonden worden met de IC's. De adressen worden ingesteld op 000, 001 en 002.

```

1 #include <Wire.h>
2 #define addr0 0b00100000 // adres eerste PCF8574

```



Figuur 11.10: PCF8574

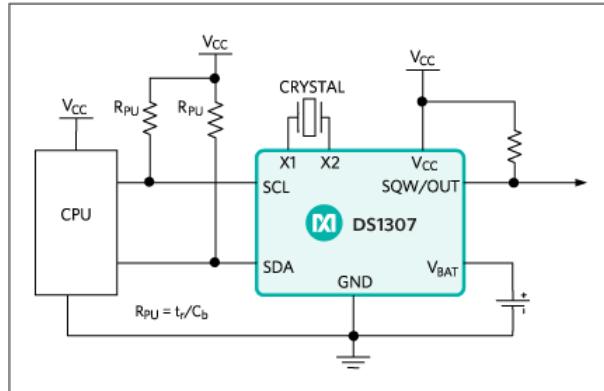
```

3   // addr0 + 1: adres tweede PCF8574
4   // addr0 + 2: adres derde PCF8574
5
6   void setup() {
7       Wire.begin(); // I2C master
8   }
9
10 // eerste lus: adressen aflopen
11 // tweede lus: alle 8 bits 1 voor 1 hoog maken
12 void loop()
13     for (int k=0; k<=2; k++){ // 3 adressen
14         for (int i=1; i<=128; i<<=1){ // i x2
15             pcf8574Write(k, i);
16             delay(100);
17         }
18         pcf8574Write(k, 0);
19     }
20
21 // LED brandt als pin laag is: ~
22 // van links naar rechts: 128 / data
23 void pcf8574Write(byte addr, byte data) {
24     Wire.beginTransmission(addr0 + addr);
25     if (data > 0) {
26         Wire.write(~(128 / data));
27     }
28     else
29         Wire.write(255);
30     }
31     Wire.endTransmission();
32 }
```

Real-Time Clock

Als voorbeeld nemen we een DS1307 IC, dit is een I²C Real-Time Clock (RTC). Het IC bevat een klok, kalender en 56 bytes aan geheugen. Er is een automatische aanpassing aan het einde van een maand en bij een schrikkeljaar (figuur 11.11).

Verder is er een backup voorzien d.m.v. externe batterij. Een RTC ontlast de microcontroller van tijdstaken.



Figuur 11.11: DS1307 RTC

Het IC bevat meerdere register die per adres een tijdswaarde in BCD-formaat bevatten. De laatste adressen dienen voor het extra RAM geheugen (figuur 11.12).

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds				Seconds	00-59
01H	0	10 Minutes			Minutes				Minutes	00-59
02H	0	12	10 Hour	10 Hour	Hours				Hours	1-12 +AM/PM 00-23
		24	PM/AM							
03H	0	0	0	0	0	DAY			Day	01-07
04H	0	0	10 Date		Date				Date	01-31
05H	0	0	0	10	Month			Month		01-12
06H	10 Year				Year				Year	00-99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H-FFH

0 = Always reads back as 0.

Figuur 11.12: Registers DS1307

BCD-code

Bij een BCD-code wordt elk cijfer van een decimaal getal voorgesteld door vier bits. Daarbij ziet de hexadecimale waarde (die 2 cijfers voorstelt) er hetzelfde uit als de decimale.

De BCD-code van 67d bijvoorbeeld is dus 0110 0111b (67h of 103d). De twee decimale cijfers kunnen dus in één byte.

Qua geheugengebruik is BCD-code niet efficient en het is ook moeilijk om met te rekenen. BCD-code wordt vooral gebruikt om de elektronica te vereenvoudigen en is ideaal om displays aan te sturen.

Als we een getal van decimaal naar BCD willen omzetten moet men het getal delen door 10 (gehele deling) en vermenigvuldigen met 16. Daarna wordt de rest van de gehele deling (door 10) er bij opgeteld.

Omgekeerd om BCD naar decimaal om te zetten voert men een gehele deling door 16 uit en vermenigvuldigt men met 10. Daarna wordt de rest van de gehele deling (door 16) er bij opgeteld.

De listing hieronder toont beide functies in Arduino code:

```

1 byte decToBcd(byte val)
2 {
3     return ( (val/10*16) + (val%10) );
4 }
5
6 byte bcdToDec(byte val)
7 {
8     return ( (val/16*10) + (val%16) );
9 }
```

Enkele voorbeelden:

67d:

$67/10 = 6$, $6*16 = 96$
 $67\%10 = 7$
BCD: $96 + 7 = 103$

103bcd:

$103/16 = 6$, $6*10 = 60$
 $103\%16 = 7$
DEC: $60 + 6 = 67$

Om de tijd in te stellen in het RTC-IC worden de juiste waarden er naar toe geschreven.

```

1 void setTime()
2 {
3     Wire.beginTransmission(DS1307_I2C_ADRES);
4     Wire.write((byte)0x00);
5     Wire.write(decToBcd(second)); // 0 in bit 7 start de klok
6     Wire.write(decToBcd(minuut));
7     Wire.write(decToBcd(uur));
8     ...
```

 9 }

Men start met het register pointer in te stellen op 0x00, deze bevat de seconden. Daarna wordt het adres automatisch verhoogd. Let op: sommige andere I²C IC's hebben deze verhoging niet. Eénmaal de tijd correct ingesteld kan die uitgelezen worden wanneer die nodig is.

```

1 void getTime()
2 {
3     Wire.beginTransmission(DS1307_I2C_ADRES);
4     Wire.write((byte)0x00);
5     Wire.endTransmission();
6     Wire.requestFrom(DS1307_I2C_ADRES, 7);      // 7 bytes ↓
    →opvragen
7     second = bcdToDec(Wire.read() & 0x7f);      // bit 7 skippen
8     minuut = bcdToDec(Wire.read());
9     uur = bcdToDec(Wire.read() & 0x3f);          // am/pm
10    ...
11 }
```

Het IC bevat ook een AM/PM-bit. In de VS gebruikt men een 12u-systeem met een AM/PM aanduiding i.p.v. het Europese 24u-systeem. Dit systeem wordt ook in onze spreektaal gebruikt (bv. het is vier uur in de namiddag).

Om plaats te besparen worden niet gebruikte bits uit het uur-register gebruikt. Zo zal bit 6 uit het uur-register 12u/24u aanduiden en bit 5 duidt AM/PM aan (bij een 12u instelling).

Om deze bits in te stellen in het uur-register kan men de bitsgewijze OR-bewerking gebruiken. Voor bijvoorbeeld het 12u systeem is dat dan: bit 6 = 1 → | 0x40 en voor PM: bit 5 = 1 → | 0x20.

In Arduino code vertaalt dit zich als volgt:

```

1     Wire.beginTransmission(DS1307_I2C_ADRES);
2     Wire.write((byte)0x00);
3     Wire.write(decToBcd(seconde));
4     Wire.write(decToBcd(minuut));
5     if(pmFlag) Wire.write(decToBcd(uur)|0x60); // bit 6 (12u) ↓
    →en bit 5 (PM)
6     else Wire.write(decToBcd(uur)|0x40); // bit 6 (12u)
7     Wire.endTransmission();
```

Omgekeerd kunnen ook de instellingen uitgelezen worden, hier voor gebruikt men de AND-bewerking.

```

1     Wire.beginTransmission(DS1307_I2C_ADRES);
2     Wire.write((byte)0x00);
```

```

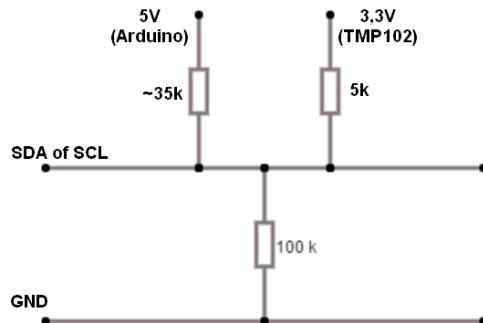
3   Wire.endTransmission();
4   Wire.requestFrom(DS1307_I2C_ADRES, 3);
5   seconde = bcdToDec(Wire.read() & 0x7f);
6   minuut = bcdToDec(Wire.read()); // BCD omzetting na ↴
    →filtering
7   uur = Wire.read();
8   pmFlag = bitRead(uur,5); // bit 5 lezen
9   uur = bcdToDec(uur & 0x1f); // bit 5,6 en 7 filteren

```

Temperatuursensor

De TMP102 is een 12-bits I²C temperatuursensor met thermostaatfunctie. Deze werkt op 3,3V en de Arduino Uno op 5V. Bij het gebruik van een UNO is er dus logische conversie nodig, dit gebeurt typisch met logic level converter (mosfets) al kan het ook eenvoudig met een extra pull-down weerstand (figuur 11.13).

Op de ESP8266 kan deze sensor rechtstreeks aangesloten worden, beiden werken op 3,3V.



Figuur 11.13: Pull-down weerstand

Extern wordt de TMP102 aangesloten zoals in figuur 11.14.

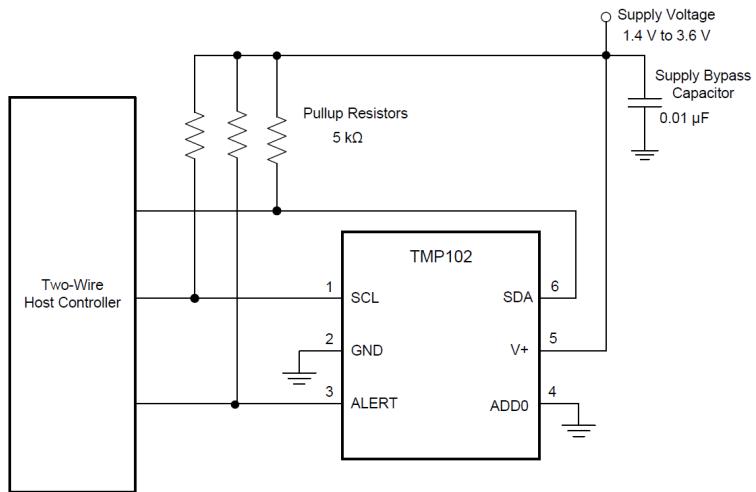
Intern zijn de registers ter beschikking zoals in figuur 11.15.

Het uitlezen gebeurt analoog zoals bij het RTC-IC.

```

1 #include "Wire.h" // I2C bibliotheek
2 #define TMP102_ADDRESS 72 // I2C adres, ADD0 verbonden met ↴
    →GND
3
4 void setup() {
5   Wire.begin(); // I2C initialiseren

```

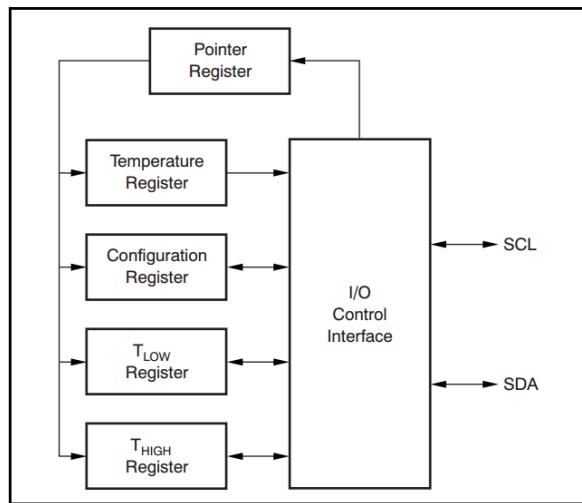


Figuur 11.14: Externe aansluiting TMP102

```

6   Serial.begin(9600);
7 }
8
9 void loop() {
10   Serial.println(getTemp());
11   delay(2000); // 2s tussen metingen
12 }
13
14 float getTemp(){
15   Wire.beginTransmission(TMP102_ADDRESS);
16   Wire.write(0x00); // pointer register
17   Wire.endTransmission();
18   Wire.requestFrom(TMP102_ADDRESS, 2); // 2 bytes opvragen
19   Wire.endTransmission();
20
21   byte MSB = Wire.read();
22   byte LSB = Wire.read();
23   ...
24 }
```

Bij de TMP102 zit de temperatuurswaarde verspreid over 2 bytes (MSB en LSB).



Figuur 11.15: Registers TMP102

MSB	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
waarde	teken	64	32	16	8	4	2	1
LSB	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
waarde	0.5	0.25	0.125	0.0625	x	x	x	x

In de MSB zit dus de gehele temperatuurswaarde, dit converteren naar een (signed) integer geeft de gehele waarde. De LSB bevat de cijfers na de komma.

De volledige temperatuur bestaat in totaal uit 12-bits, 8 bits in MSB en 4 bits in LSB. Als we MSB 4 bits naar links shiften en LSB 4 bits naar rechts shiften en dan beide optellen met de OR-functie geeft dit een resultaat dat nog 16x te groot is (4 bits).

Om de juiste waarde te bekomen moet men het resultaat nog vermenigvuldigen met 0,0625 of delen door 16.

```

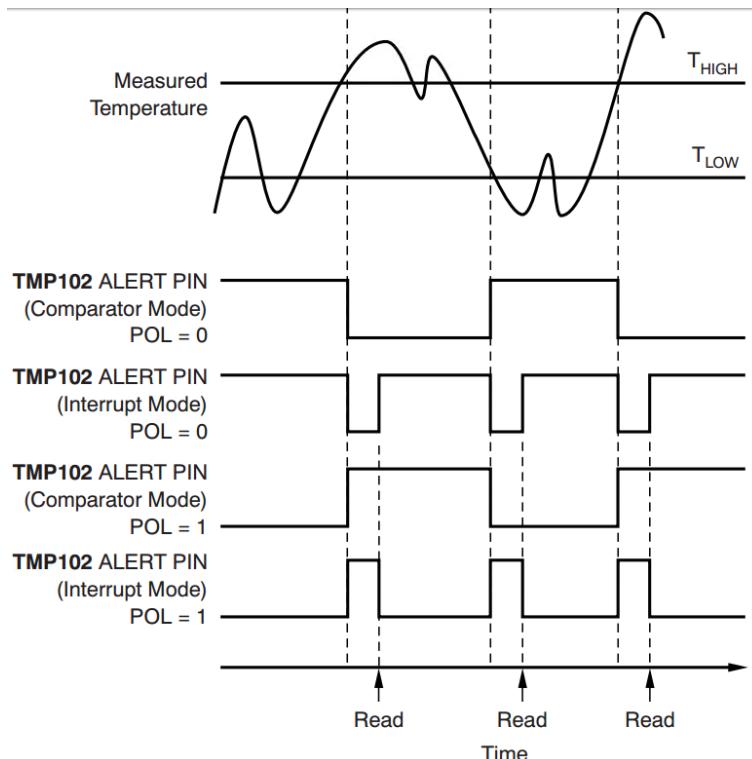
1  MSB 4 bits naar links
2  LSB 4 bits naar rechts
3  int waarde = MSB | LSB
  
```

Voorbeeld:

- MSB = 0001 0111
- LSB = 1000 0000
- waarde = 1 0111 1000b = 376d
- temperatuur = waarde * 0,0625 = 23,5

De TMP102 bevat een thermostaat functie en heeft hiervoor twee vergelijkingsregisters T_{HIGH} en T_{LOW} . Indien één van beiden overschreden wordt dan wordt er een signaal via de ALERT-pin gegenereerd.

Er zijn twee modes: comparator en interrupt (figuur 11.16). Bij de comparator mode blijft de ALERT-pin hoog of laag tot een stijging of daling van de temperatuur optreedt. Bij de interrupt mode wordt er een puls gegenereerd op de ALERT-pin. Het niveau van de ALERT-pin kan ook geïnverteerd worden.



Figuur 11.16: Modes TMP102

De thermostaat-registers worden ingesteld door eerst de juiste waarde naar het pointer register te schrijven (adres) waarna men de twee bytes met data kan schrijven.

```

1  Wire.beginTransmission(TMP102_ADDRESS);
2  Wire.write(0x02); // pointer = TLOW register
3  Wire.write(0x20); // eerste byte
4  Wire.write(0x80); // tweede byte
5  Wire.endTransmission();

```

I²C scanner

De scanner toont een lijst van I²C-devices op de bus. Alle mogelijke adressen worden afgegaan en wanneer een ACK ontvangen wordt van een slave, dan is er een slave aanwezig op dat adres.

```

1  Serial.begin (9600);
2  byte count = 0;
3  Wire.begin();
4  for (byte i = 8; i < 120; i++) {
5      Wire.beginTransmission (i);
6      if (Wire.endTransmission () == 0) {
7          Serial.print ("Found address: 0x");
8          Serial.print (i, HEX);
9          count++;
10         delay (1);
11     }

```

Dialecten

Wegens patentredenen gebruiken sommige sensors een variant van I²C, dit geeft als voordeel dat er goedkopere sensors kunnen geproduceerd worden. Het nadeel is dat de standaard I²C functies niet werken en men dus aangepaste code nodig heeft.

Enkele voorbeelden: SHT15 temperatuurssensor, TM1636 display driver, ...

11.3 SPI

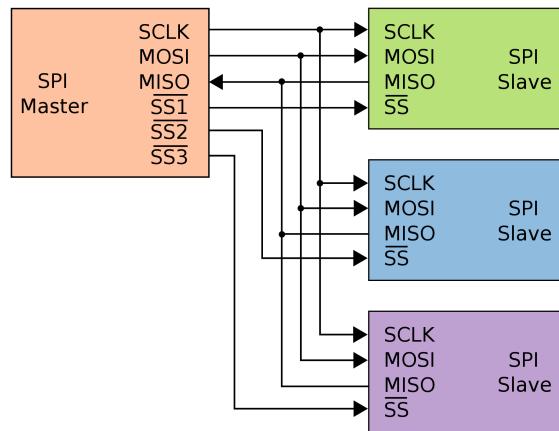
11.3.1 Werking

Net zoals I²C is dit protocol ontwikkeld voor de communicatie tussen IC's. SPI staat voor Serial Peripheral Interface en is bedoeld voor korte afstanden.

Er zijn één of enkele slaves mogelijk, per slave is er telkens een extra pin nodig. SPI werkt zonder adressen en is eenvoudiger dan I²C. Het protocol heeft wel meer pinnen nodig en is bedoeld voor een klein aantal slaves (figuur 11.17).

De SPI-bus is gebaseerd op een synchrone seriële communicatie. Er is één kloklijn SC(L)K en twee datalijnen: MISO: Master In Slave Out (master lezen) en MOSI: Master Out Slave In (master schrijven).

Naast de klok en datalijnen is er nog een SS-lijn (Slave Select) per slave, deze moet men laag maken om de gewenste slave te selecteren.



Figuur 11.17: De SPI-bus

11.3.2 Arduino ICSP-header

Op Arduino UNO is er een aparte ICSP-header beschikbaar met de SPI-bus (figuur 11.18). De SPI-signalen zijn verder ook beschikbaar op pinnen 11, 12 en 13. Bij het gebruik de SPI-bus zijn deze pinnen niet beschikbaar voor iets anders.



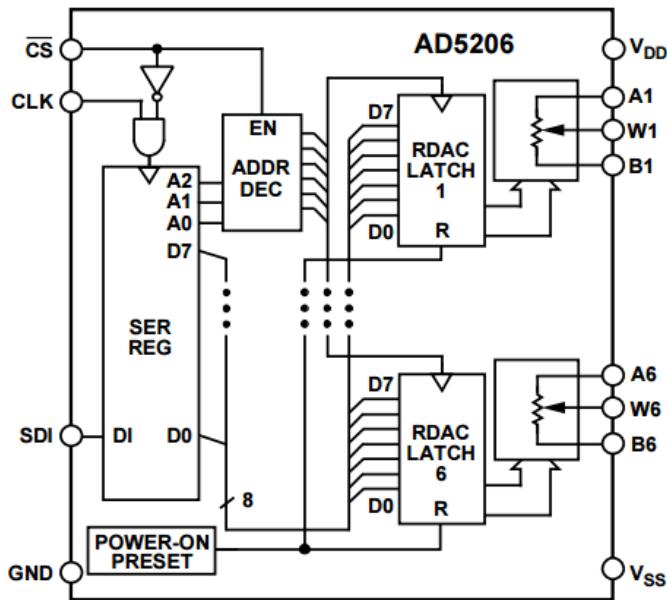
Figuur 11.18: ICSP-header

11.3.3 Praktische voorbeelden

AD5206 digitale potmeter

Dit IC bevat 6 digitale potmeters en zijn SPI compatibel (figuur 11.19). De master (bv. een Arduino UNO) stuurt 2 bytes naar het IC, het gewenste kanaal (0-5) en de gewenste waarde (0-255). Aangezien er niet gelezen hoeft te worden is er geen MISO connectie nodig.

Een codevoorbeeld voor de Arduino ziet er als volgt uit:



Figuur 11.19: Digitale potmeter

```

1 #include <SPI.h>
2
3 // pin 10 als slave select digital pot:
4 const int slaveSelectPin = 10;
5
6 void setup() {
7     // SS als output:
8     pinMode (slaveSelectPin , OUTPUT);
9     // initialiseer SPI:
10    SPI.begin();
11 }
12
13 void loop() {
14     // 6 kanalen doorlopen:
15     for (int channel=0; channel<6; channel++){
16         // weerstandswaarden instellen:
17         for (int level=0; level<255; level++){
18             digitalPotWrite(channel , level);
19             delay(10);
20         }
21     }
22 }
```

```
23
24 void digitalPotWrite(int address,int value){
25     // SS laag om IC te selecteren:
26     digitalWrite(slaveSelectPin, LOW);
27     // kanaal + waarde doorsturen:
28     SPI.transfer(address);
29     SPI.transfer(value);
30     // SS hoog om IC te deselecteren:
31     digitalWrite(slaveSelectPin, HIGH);
32 }
```

12

Bluetooth

Dit hoofdstuk handelt over Bluetooth legacy en niet over Bluetooth low energy (BLE).

Bluetooth is een draadloze radioverbinding op korte afstand met een gebruikte frequentieband van 2,4GHz.

Het is een open standaard en dus niet gekoppeld aan één fabrikant. Er bestaan verschillende classes, versies en profielen. Deze geven respectievelijk de afstand, het verbruik, het gebruik en de maximum snelheid aan.

Bluetooth wordt gebruikt voor data-overdracht tussen 2 nabije toestellen.

12.1 Indelingen

12.1.1 Class

De class geeft de maximum afstand aan:

- Class 1: tot 100m
- Class 2: tot 10m
- Class 3: tot 1m

12.1.2 Versies

De versie bepaalt o.a. de maximum snelheid en het verbruik:

- Versie 1: 1Mbit/s

- Versie 1.2: 2Mbit/s
- Versie 2: hogere snelheid, betere foutcorrectie en lager verbruik
- Versie 3: sneller en betrouwbaarder
- Versie 4: energiezuiniger
- Versie 5: groter bereik, zuiniger, sneller en betere beacons

12.1.3 Profielen

Het profiel geeft de aard van gebruik aan:

- SPP: serial port profile
- FTP: file transfer profile
- HID: muizen, toetsenborden, ...
- HSP: headset profile
- ...

12.2 Serial Port Profile

Dit profiel emuleert een seriële verbinding en laat draadloze seriële communicatie toe. Het kan gebruikt worden om gegevens uit te wisselen tussen bv. een microcontroller en een sensor. Het kan ook gebruikt worden om een microcontroller te bedienen via een smartphone.

12.2.1 Bluetooth Module

Deze module zorgt voor een draadloze vervanging van een seriële verbinding en kan eenvoudig aangesloten worden op een Arduino. De aansturing is bijna identiek als de seriële poort. Er zijn verschillende uitvoeringen (classes).

Eén hiervan is de Sparkfun BlueSMiRF module (figuur 12.1 en figuur 12.2) die bestaat in een **Silver**, **Gold** en **HID** uitvoering.

De module kan als volgt aangesloten worden (figuur 12.3):

- VCC verbinden met 5V en GND met GND
- RX Arduino met TX BlueSMiRF



Figuur 12.1: Voorzijde BlueSMiRF



Figuur 12.2: Achterzijde BlueSMiRF

- TX Arduino met RX BlueSMiRF

De module kan in 2 modi staan (figuur 12.4):

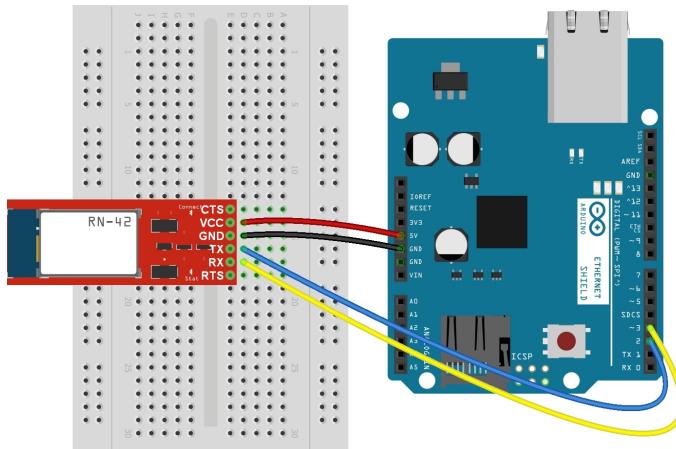
- data-mode: dit is de gewone werking voor het versturen van data
- command-mode: in deze mode kunnen de configuratieparameters ingesteld worden

Enkele configuratieparameters zijn:

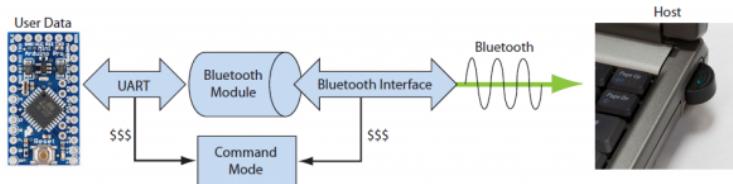
- device name
- baudrate
- PIN nummer
- scannen naar en connectie maken met andere modules
- configuratietimer: 60s na power up

Met de status LED's kan men de mode en connectie van de module nagaan:

- rode LED (Stat): toont status v/d module
 - 10x/s: config mode
 - 2x/s: data mode, timer is nog niet afgelopen
 - 1x/s: data mode, timer is afgelopen
- groene LED (Connect): licht op bij connectie



Figuur 12.3: Aansluiting BlueSMiRF



Figuur 12.4: Modes BlueSMiRF

Command mode

De configuratieparameters instellen gebeurt a.d.h.v. opdrachten of commando's. Dit kan zowel langs UART-zijde als langs bluetooth-zijde.

Je kan als volgt naar command mode overschakelen:

- maak een seriële connectie met de module (115200)
- verstuur de karakters: "\$\$\$"
- verstuur commando's
- verstuur de karakters: " --- "

De betekenis van de commando's is als volgt:

- U: tijdelijk de baudrate veranderen:
 - vb. baudrate instellen: U,9600,N

- dit stelt de baudrate in op 9600 (no parity)
- na dit commando gaat men automatisch naar data mode
- V: firmware versie tonen
- L: link kwaliteit tonen
- H: help opvragen

De SET en GET commando's laten toe om instellingen in te stellen of te raadplegen.

De SET commando's wijzigen instellingen permanent en worden in het flash geheugen van de module bewaard. Dit is bijvoorbeeld de device name, het vermogen, het profiel, ... Wees steeds voorzichtig met het gebruik van SET, dit kan de module (tijdelijk) onbruikbaar maken.

Met de GET commando's kunnen de instellingen opgevraagd worden, deze zijn dus ongevaarlijk voor de module.

In command mode kan met het commando D de instellingen opgevraagd worden:

- stuur " \$\$" door (zonder Newline)
- de module antwoordt met: CMD
- stuur een 'D' door (met Newline)
- de module toont de gegevens
- sluit af met " --- "

De typische gegevens die je te zien krijgt zijn o.a. pincode, baudrate, bluetooth name, ...

```
1 ***Settings***  
2 BTAddress=0006667D4E48  
3 BTName=RNBT-4E48  
4 BaudRate(SW4)=9600  
5 Mode=Slave  
6 Authen=0  
7 PinCode=1234  
8 Bonded=0  
9 Rem=0006667D4E4E
```

Hieronder vind je een voorbeeld van het gebruik op een Arduino. De hardware seriële poort wordt gebruikt voor de seriële monitor en voor het uploaden van nieuwe code, voor de bluetooth communicatie wordt gebruikt gemaakt van een software seriële poort. Een software seriële poort werkt niet stabiel genoeg bij hoge snelheden, de snelheid zal dus verlaagd worden in `setup()`.

```

1 #include <SoftwareSerial.h>
2
3 int bluetoothTx = 2; // TX-pin bluetooth
4 int bluetoothRx = 3; // RX-pin bluetooth
5
6 SoftwareSerial bluetooth(bluetoothTx, bluetoothRx); // RX, TX
7
8 void setup()
9 {
10    Serial.begin(9600); // serial monitor
11    bluetooth.begin(115200); // standaard baudrate module
12    bluetooth.print("$$$"); // command mode
13    delay(100); // CMD wordt teruggestuurd
14    bluetooth.println("U,9600,N"); // tijdelijke verandering
15    bluetooth.begin(9600); // nieuwe baudrate gebruiken
16 }
17
18 void loop()
19 {
20    if(bluetooth.available()) // ontvangen
21    {
22        Serial.print((char)bluetooth.read());
23    }
24    if(Serial.available()) // versturen
25    {
26        bluetooth.print((char)Serial.read());
27    }
28 }
```

12.2.2 Smartphone

De meeste smartphone's ondersteunen bluetooth, er zijn app's ter beschikking die het SPP profiel ondersteunen. Deze kunnen als bluetooth monitor gebruikt worden. Voor Android is er bijvoorbeeld *Serial Bluetooth Terminal (Kai Morich)* en voor iOS bijvoorbeeld *BluTerm (Vensi)*. Merk op dat niet alle bluetooth modules compatibel zijn met iOS.

Verbinding maken op smartphone

Deze stappen moeten ondernomen worden om een verbinding te maken met de bluetooth module:

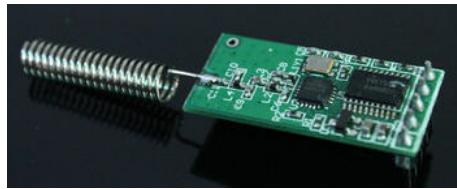
- bluetooth inschakelen
- het device koppelen, let daarbij op het MAC-adres
- standaard pin BlueSMIRF: 1234
- app openen en device kiezen

12.2.3 Andere draadloze verbindingen

Naast bluetooth zijn er nog andere technologien die toelaten om een draadloze verbinding te maken

RF

Deze technologie werkt meestal op een radiofrequentie van 433MHz. Het voorziet een draadloze overdracht met bijvoorbeeld het serieel protocol. Sommige modules laten een afstand tot 1km toe (figuur 12.5).



Figuur 12.5: RF module

IrDA

Bij IrDA gebeurt de overdracht via infrarood licht. Het is bedoeld voor korte afstand en gevoelig voor storing (figuur 12.6).

IRComm is het serieel protocol over IR en de voorganger van bluetooth. De IrDA technologie wordt nog weinig gebruikt.

ZigBee

Dit is een open standaard voor draadloze communicatie en een aanvulling op Bluetooth en wifi. Het wordt gebruikt voor sensoren, huisalarmen, panic alarm, smart home systemen, enz (figuur 12.7).



Figuur 12.6: IrDA module



Figuur 12.7: ZigBee module

LoRa

LoRa werd ontwikkeld door Proximus (figuur 12.8). Het staat voor een laag stroomverbruik en grote afstanden en is speciaal bedoeld voor IoT-devices.



Figuur 12.8: Logo LoRa

13

Geheugen

Het geheugen op de Arduino is beperkt, bij grotere projecten moet dit optimaal benut worden. Zowel de code als data-opslag is beperkt en kan eventueel uitgebreid worden met extern geheugen.

13.1 Geheugenstructuur

Bij de Arduino UNO (met ATmega328 processor) zijn er 3 types geheugen ingebouwd:

- Flash (32kB)
- SRAM (2kB)
- EEPROM (1kB)

13.1.1 Flash

In dit gedeelte wordt de uitvoerbare code bewaard, dit na het compileren en uploaden. De code blijft in het geheugen, ook na een power off. Meestal wordt hier ook een kleine bootloader geïnstalleerd zodat programma's via het serieel protocol kunnen geupload worden over de USB-poort.

13.1.2 EEPROM

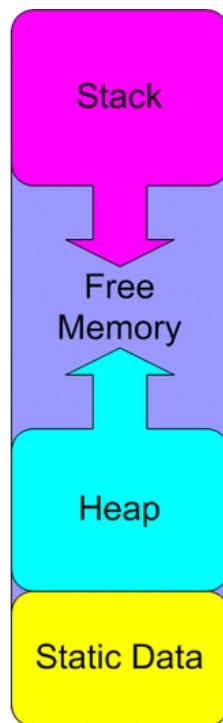
EEPROM of Electronically Erasable Programmable Read Only Memory is niet vluchtig en wordt gebruikt als lange termijn geheugen voor data. Het is traag in gebruik en kan een beperkt aantal keer overschreven worden (100000x). Het is ideaal om een aantal instellingen te bewaren of een beperkte hoeveelheid meetwaarden.

13.1.3 SRAM

SRAM staat voor Static Random Access Memory en wordt gebruikt om (vluchtige) data te bewaren. Dit kunnen bijvoorbeeld variabelen zijn die in de code gebruikt worden. Deze data gaat verloren na een reset. SRAM is een zeer snel geheugen (figuur 13.1).

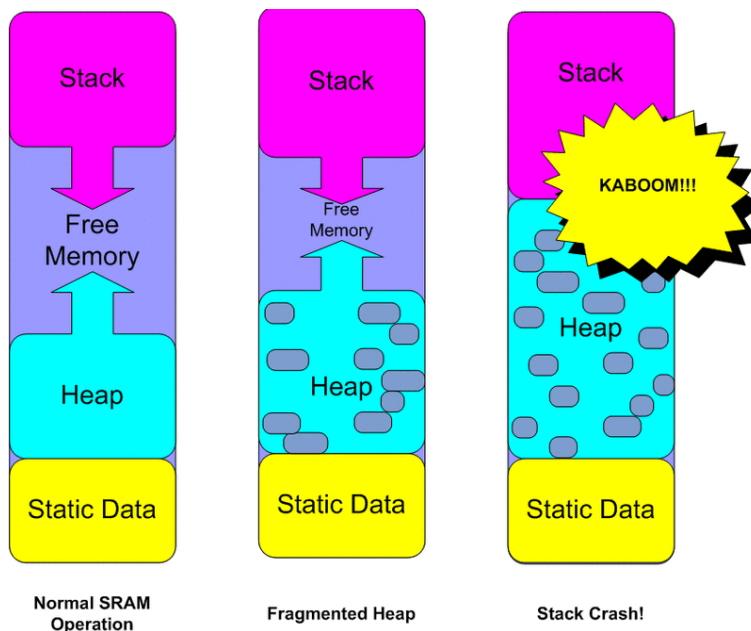
Het SRAM geheugen bestaat uit verschillende delen:

- Stack: lokale variabelen, interrupts en functie aanroepen. De stack groeit van boven naar beneden naargelang het gebruik)
- Heap: dynamisch gealloceerde data (bv. arrays die van grootte veranderen). De heap groeit van beneden naar boven naargelang het gebruik.
- Static Data: globale en statische variabelen. De static data is vast in grootte tijdens de run van een programma.



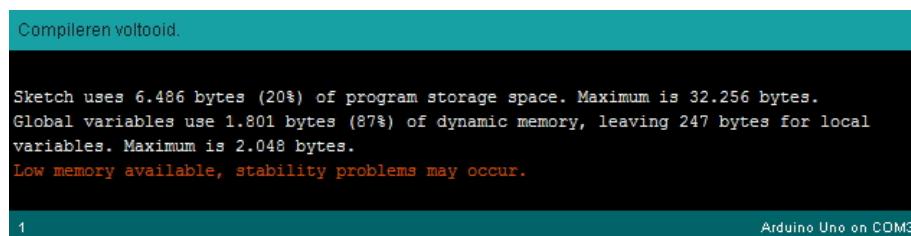
Figuur 13.1: SRAM geheugen

Er kunnen conflicten optreden wanneer zowel de Heap als de Stack groeien (figuur 13.2). Bij intens geheugenbereik kunnen beide mekaar overlappen, dit geeft geen foutmelding maar levert wel verkeerde data op. De situatie is te vermijden door het geheugengebruik binnen de perken te houden en te optimaliseren.



Figuur 13.2: SRAM conflicten

Indien een groot deel van het geheugen in beslag wordt genomen treedt tijdens het compileren in de Arduino IDE een waarschuwing op (figuur 13.3), stabiliteitsproblemen kunnen dan optreden.



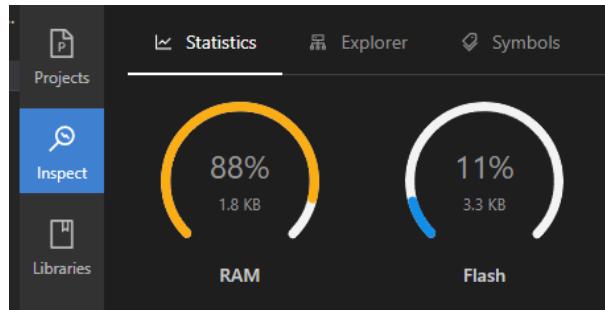
Figuur 13.3: Waarschuwing compileren

Bij het gebruik van PlatformIO kan je het gebruik van het geheugen inspecteren via *PlatformIO Home* en het tabblad *Inspect* (figuur 13.4).

13.2 Geheugen optimaliseren

Er zijn verschillende manieren om het geheugengebruik binnen de perken te houden:

- dode code verwijderen:



Figuur 13.4: Inspecteren geheugen

- ongebruikte bibliotheken, functies, variabelen
- onbereikbare code (bv. in een if/else)
- functies maken van herhaalde code
- constante data naar PROGMEM (flash) verhuizen
- te grote zaken verkleinen:
 - buffers: arrays niet te groot maken
 - datatypes: gebruik steeds een zo klein mogelijk type
- lokale variabelen gebruiken i.p.v. globale zodat het geheugengebruik gekend is

Figuur 13.5 toont de beschikbare data types.

13.3 Gebruik EEPROM

13.3.1 Bibliotheek

Om het EEPROM geheugen te gebruiken heb je speciale functies nodig, deze zitten in een standaard bibliotheek EEPROM.h.

Er zijn twee functies aanwezig:

- `read(address)`: een byte lezen
- `write(address,value)`: een byte schrijven

Bij de Arduino UNO liggen de adressen tussen 0 en 1023 (1kB geheugen).

Helaas kan je schrijven naar een adres hoger dan 1023, er wordt dan weer vanaf 0 geteld en daar weggeschreven. Zo zal bijvoorbeeld het adres 1024 schrijven naar adres 0. Opgeletten dus met het toekennen van adressen.

Data Types	Size in Bytes	Can contain:
boolean	1	true (1) or false (0)
char	1	ASCII character or signed value between -128 and 127
unsigned char, byte, uint8_t	1	ASCII character or unsigned value between 0 and 255
int, short	2	signed value between -32,768 and 32,767
unsigned int, word, uint16_t	2	unsigned value between 0 and 65,535
long	4	signed value between 2,147,483,648 and 2,147,483,647
unsigned long, uint32_t	4	unsigned value between 0 and 4,294,967,295
float, double	4	floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.)

Figuur 13.5: Data types

13.3.2 Schrijven en lezen

Het eerste codefragment toont het wegschrijven van één byte naar het EEPROM geheugen.

```

1 #include <EEPROM.h>
2
3 int addr = 100;
4
5 void setup()
6 {
7     int val = analogRead(0) / 4;
8     EEPROM.write(addr, val);
9 }
```

Een byte lezen uit het geheugen kan als volgt:

```

1 #include <EEPROM.h>
2
3 int addr = 100;
4 byte value;
5
6 void setup()
7 {
```

```
8   Serial.begin(9600);
9   value = EEPROM.read(addr);
10  Serial.println(value, DEC);
11 }
```

Aan het EEPROM geheugen zijn er beperkingen verbonden. De standaard lees- en schrijffuncties laten enkel bytes toe (getallen tussen 0 en 255). Andere datatypes zijn mogelijk met externe bibliotheken of men kan ook zelf opsplitsen:

```
1 getal = 256 * EEPROM.read(addrH) + EEPROM.read(addrL);
```

13.3.3 Extern geheugen

Externe IC's laten toe om het geheugen uit te breiden. Deze zijn meestal benaderbaar via het I²C- of SPI-protocol.

Er zijn verschillende types met de voor- en nadelen die ook aan het intern geheugen gekoppeld zijn:

- flash: groot
- SRAM: snel en vluchtig
- EEPROM: traag en niet vluchtig

14

Netwerken

Om zaken van buitenaf te besturen is er een verbinding nodig, er zijn verschillende mogelijkheden waarvan sommige reeds besproken zijn in andere hoofdstukken:

- IR
- serieel
- bluetooth
- ethernet / wifi
- GPRS (GSM-netwerk)

Ethernet en wifi bieden het voordeel dat het netwerk overal bereikbaar is, meestal zonder extra kosten.

14.1 Hardware

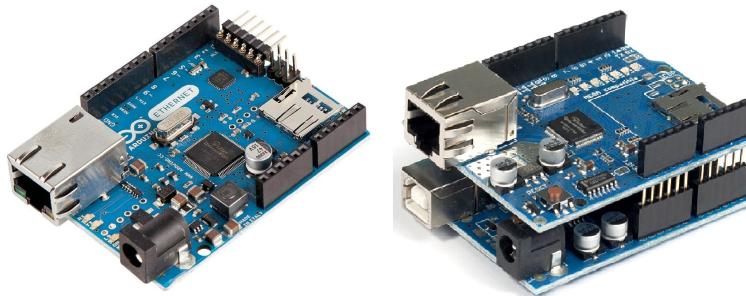
14.1.1 Arduino UNO

Ethernet (shield)

Er zijn twee mogelijkheden: onboard (Arduino Ethernet) of een apart shield op de Arduino UNO. De Arduino Ethernet is kleiner maar heeft geen USB-aansluiting en heeft dus een extra programmer nodig (figuur 14.1).

Beide versies hebben een RJ-45 connectie voor de ethernet verbinding en een SD-kaart lezer. Er wordt gebruik gemaakt van het SPI-protocol voor de Arduino-Ethernet communicatie.

Er is ook de mogelijkheid tot een extra PoE (Power over Ethernet) module zodat geen externe voeding meer nodig is.



Figuur 14.1: Arduino Ethernet en Ethernet shield

WiFi Shield

Dit shield (figuur 14.2) heeft dezelfde mogelijkheden als het ethernet shield maar dan draadloos.

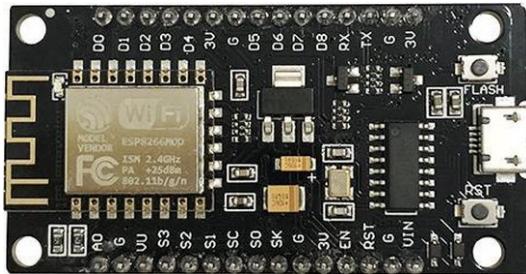


Figuur 14.2: WiFi shield

14.1.2 ESP8266

De ESP8266 module heeft reeds een ingebouwde WiFi module en is dus ideaal voor netwerk gerelateerde projecten (figuur 14.3).

De documentatie over de ESP8266WiFi library kan men raadplegen op:
<https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html>



Figuur 14.3: ESP8266 NodeMCU

14.2 TCP-verbindingen

Een netwerkconnectie kan als zuiver communicatiekanaal gebruikt worden, dit zonder overhead van bijvoorbeeld een browser en de bijhorende HTML-code. Een typisch voorbeeld is een telnet client en server. Dit is vergelijkbaar met een seriële poort, namelijk karakters verzenden en ontvangen.

Er zijn ook verschillen met de seriële poort: bij een TCP-verbinding moet men wachten op een client connectie, er zijn meerdere connecties tegelijk mogelijk met de server en in tegenstelling tot een seriële verbinding is er wel een echo in de telnet client. Tenslotte wordt er geen baudrate gebruikt maar is er wel een poortnummer.

14.2.1 Voorbeeld

Dit voorbeeld is analoog aan het voorbeeld bij de seriële poort en toont een menu aan de gebruiker. Hierbij fungeert de Arduino als telnet server en stuurt een menu met opties naar de client(s). De client kan keuze maken uit het menu waarna de server de keuze verwerkt.

```

1 ...
2 // initialisatie
3 WiFiServer telnetServer(23);
4 WiFiClient serverClient;
5
6 bool printmenu = true;
7
8 void setup() {
9     Serial.begin(115200);
10
11     connectWifi();
12
13     telnetServer.begin();
14     telnetServer.setNoDelay(true);

```

```

15 }
16
17 ...
18 // menu doorsturen
19 if (serverClient && serverClient.connected()) {
20     if (printmenu) {
21         serverClient.println("Toets de letter van uw keuze in: ↴");
22         serverClient.println(" a: alle LED's aan");
23         serverClient.println(" b: alle LED's uit");
24         ...
25     printmenu = false;
26 }
27
28 ...
29 // keuze verwerken
30 if (serverClient.available() > 0) {
31     char keuze = serverClient.read();
32     if (keuze == 'a') {
33         ...
34     }
35     if (keuze == 'b') {
36         ...
37 }

```

Als TCP-client kan men bijvoorbeeld *Putty* gebruiken op een PC, of *Simple TCP Socket Tester* op een Android smartphone.

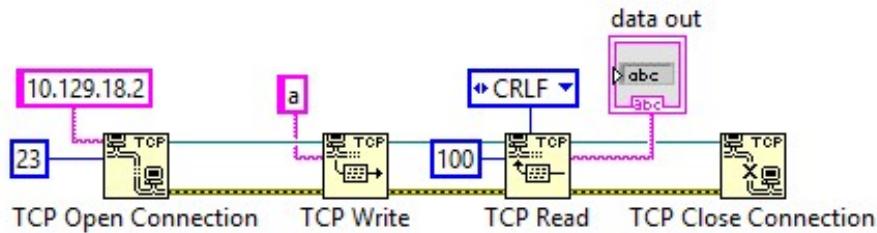
14.2.2 TCP-client in hogere programmeertalen

De Arduino telnet server kan ook bediend worden met een zelfgeschreven app of programma. Er is weinig overhead nodig, enkel het nodige wordt verzonden. De app of programma treedt op als netwerk client.

Er zijn vier taken die moeten verricht worden: een connectie maken, lezen, schrijven en de connectie verbreken.

Labview

In dit voorbeeld wordt een TCP connectie geopend op poort 23, daarna wordt karakter 'a' geschreven naar de server en wordt het menu ingelezen. Tenslotte wordt de connectie verbroken (figuur 14.4).



Figuur 14.4: TCP connectie in Labview

Powershell

Ook met Powershell kan eenvoudig een TCP-verbinding opgezet worden. De listing toont een voorbeeld van een Powershell-script:

```

1 $IPadres = "10.129.18.2"
2 $poort = "23"
3
4 $tcpConnection = New-Object System.Net.Sockets.TcpClient(→
    →$IPadres, $poort)
5 $tcpStream = $tcpConnection.GetStream()
6 $reader = New-Object System.IO.StreamReader($tcpStream)
7 $writer = New-Object System.IO.StreamWriter($tcpStream)
8 $writer.AutoFlush = $true
9
10 while ($tcpConnection.Connected) {
11     while ($tcpStream.DataAvailable) {
12         $reader.ReadLine()
13     }
14     if ($tcpConnection.Connected) {
15         Write-Host -NoNewline "prompt> "
16         $command = Read-Host
17         if ($command -eq "escape") break
18         $writer.WriteLine($command) | Out-Null
19     }
20 }
21 $reader.Close()
22 $writer.Close()
23 $tcpConnection.Close()
```

De output in de Powershell console kan er dan als volgt uitzien:

```

1 PS H:\> C:\Users\mario.wyns\Documents\tcpclient.ps1
2 prompt> a
```

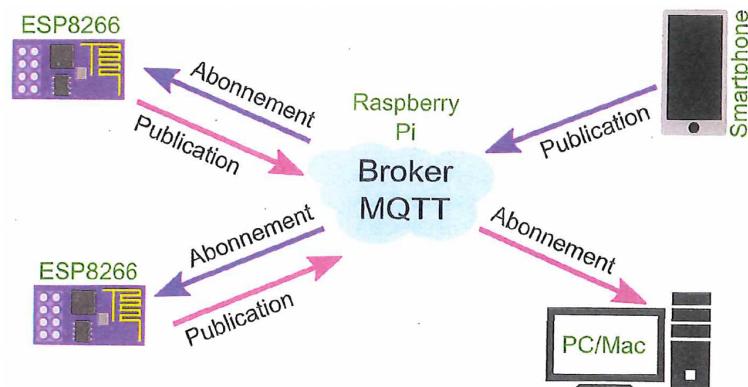
- 3 Toets de letter van uw keuze in:
 4 a: eerste optie
 5 q: connectie verbreken
 6 prompt >
-

14.3 MQTT

Als er communicatie tussen sensoren, actuatoren, machines en dergelijke nodig is dan is er een gemeenschappelijke taal nodig. Eén daarvan is MQTT (Message Queue Telemetry Transport), dit is een lichtgewicht TCP/IP-protocol met een centrale bemiddelaar ook wel *broker* genoemd.

De apparaten hoeven elkaar niet te kennen, er is ook een mogelijkheid tot het instellen van QoS (Quality of Services).

14.3.1 Architectuur



Figuur 14.5: Architectuur MQTT

Broker

De letterlijke vertaling voor broker is tussenpersoon. Deze zorgt voor de verbinding tussen de verschillende apparaten. Hij ontvangt gegevens van sensoren, schakelaars en dergelijke en verstuur gegevens naar bijvoorbeeld actuatoren en monitoren. De broker voorziet eventueel ook beveiliging a.d.h.v. authenticatie en TLS (figuur 14.5).

Topics

De communicatie tussen de verschillende componenten gebeurt a.d.h.v. topics, een topic is niet meer dan een aaneenschakeling van karakters. Topics bevatten een hiërarchie, bijvoorbeeld huis/keuken/temperatuur.

Men kan ook jokertekens gebruiken: + en #. Deze laten controle toe op meerdere topics tegelijk. /huis/+/temperatuur bevat bv.: (één niveau)

- /huis/keuken/temperatuur
- /huis/living/temperatuur

/huis/# bevat bv.: (meerdere niveau's)

- /huis/salon/hygro
- /huis/gang/temperatuur
- # is steeds het laatste karakter

Publication

Sensoren en dergelijke kunnen gegevens publiceren, het topic is vrij te kiezen. Zonder beveiliging staat in principe alles open. De broker stockeert alle binnenkomende informatie en geeft deze informatie door aan alle apparaten die geabonneerd zijn op dit topic.

Subscription

Actuatoren (verwarming, verlichting, ...) kunnen gegevens ontvangen en nemen daarvoor een abonnement (subscription) op een bepaald topic. Met joker tekens kunnen meerdere topics tegelijk gevolgd worden. Op basis van ontvangen gegevens wordt er dan een beslissing genomen.

14.3.2 Software

De broker draait meestal op een (linux)server, bijvoorbeeld een Raspberry PI. Een veel gebruikt pakket als broker is *Mosquitto*. Na de installatie is Mosquitto klaar voor gebruik.

Op de apparaten, bijvoorbeeld een Arduino of andere microcontroller kan men gebruik maken van een bibliotheek, bijvoorbeeld *PubSubClient*. Deze kan connecteren met de broker en zowel publiceren als ontvangen.

Mosquitto voorziet niet enkel de broker (server) maar ook de clients. Dit zijn twee tools om manueel topics in te stellen of te raadplegen.

`mosquitto_pub` laat toe om gegevens te publiceren:

```
student:~$  
    mosquitto_pub -h localhost -t huis/gang/temp -m 18
```

mosquitto_sub toont bepaalde topics:

```
student:~$ mosquitto_sub -v -h localhost -t huis/+temp  
huis/keuken/temp 21  
huis/gang/temp 18
```

ESP8266: mqtt_basic

De bibliotheek *PubSubClient* kan ook op de ESP8266 module gebruikt worden, zie hiervoor het voorbeeld `mqtt_esp8266` uit de bibliotheek.

14.4 Het web protocol

Naast een eenvoudige TCP client/server structuur kan men ook gebruik maken van een web-server. Dit heeft als voordeel dat je microcontroller eenvoudig bereikbaar is vanuit om het even welke webbrowser. Het nadeel is dat de code complexer wordt.

Webservers en -clients gebruiken het HTTP-protocol waarbij het opzetten van een connectie vast ligt en beschreven wordt in een RFC document. De client stuurt een HTTP-request naar de server, de server antwoordt vervolgens met een HTTP-response. Deze response bevat o.a. de inhoud van de standaard webpagina.

De request en response bestaan uit gewone tekstkarakters (figuur 14.6).

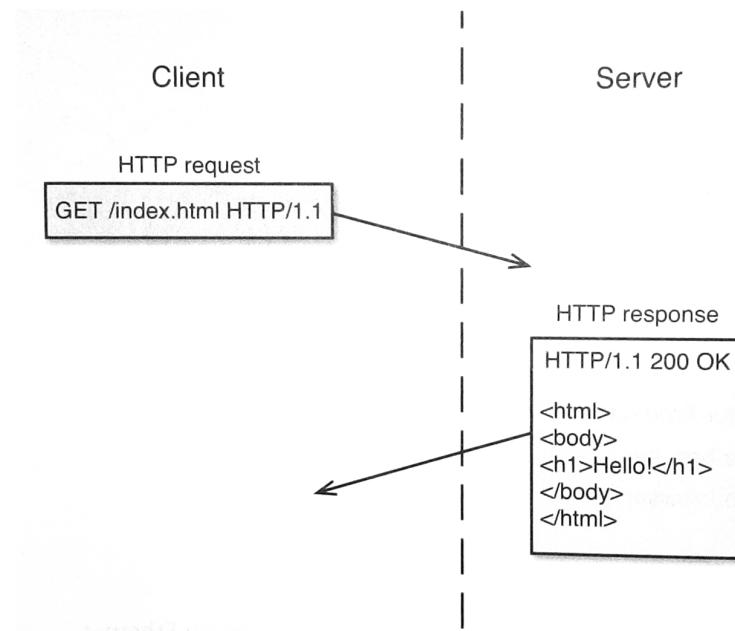
14.4.1 HTTP-request

De request bestaat uit 3 delen:

- een request lijn
- request header lijnen (optioneel)
- lege lijn gevolgd door een CR en LF

Met deze combinatie vraagt de client wat hij wil zien van de server.

De request lijn bevat o.a. de gewenste actie en data en het gebruikte protocol. De actie wordt gedefinieerd door de Method Token, dat is bijvoorbeeld GET om een pagina op te vragen. De data is bijvoorbeeld `index.html`.



Figuur 14.6: HTTP protocol

Een voorbeeld van een request lijn is: GET /index.html HTTP/1.1

De client kan optioneel bijkomende info versturen, een compleet voorbeeld:

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0
Connection: Close
```

De client geeft in dit voorbeeld zijn identiteit mee en de connectie wordt verbroken na het antwoord van de server.

14.4.2 HTTP-response

De response van de server bestaat eveneens uit 3 delen:

- de status lijn
- response header lijnen (optioneel)
- lege lijn gevolgd door CR en LF

Na de response volgt de eigenlijke data.

De statuslijn vertelt aan de client hoe hij de request gaat afhandelen en bestaat uit 3 delen: versie, status-code en beschrijving. De versie is meestal: HTTP/1.1, de status-code is bijvoorbeeld 200 (OK) en de beschrijving hangt af van de server.

Een voorbeeld van een response is: HTTP/1.1 200 OK

De server kan optioneel bijkomende info versturen, een compleet voorbeeld:

```
HTTP/1.1 200 OK
Host: odisee.be
Connection: Close
```

In het voorbeeld geeft de server zijn identiteit mee en wordt de connectie verbroken na het antwoord van de server.

14.4.3 HTTP op de ESP8266

WebClient voorbeeld

Een goed voorbeeld vind je in de Arduino voorbeelden onder *ESP8266HTTPClient* namelijk *BasicHttpClient*.

WebServer voorbeeld

Een webserver (zonder encryptie) luistert naar poort 80. Het voorbeeld ontvangt de request van een client zonder verdere controle. Vervolgens wordt de response-header verstuurd gevolgd door de data in HTML-formaat.

Op het einde wordt de verbinding verbroken.

Het voorbeeld hieronder toont een teller:

- lijn 3-30: de html code die naar de client zal gestuurd wordt
- lijn 33-40: WiFi credentials
- lijn 78-83: hier wordt de html code naar de client gestuurd

```

1 String header = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n"
   →\n\r\n";
2
3 String html_1 = R"=====(
4 <!DOCTYPE html>
5 <html>
```

```
6 <head>
7 <meta name='viewport' content='width=device-width, initial-
8   scale=1.0' />
9 <meta charset='utf-8'>
10 <meta http-equiv='refresh' content='1'>
11 <style>
12   body {font-size:100%;}
13   #main {display: table; margin: auto; padding: 0 10px 0 10
14     px; }
15   h2 {text-align:center; }
16   p { text-align:center; }
17 </style>
18   <title>Auto Update Example Using HTML</title>
19 </head>
20
21 <body>
22   <div id='main'>
23     <h2>Auto Update Example Using HTML</h2>
24     <div id='count'>
25       <p>Count = %count%</p>
26     </div>
27   </div>
28 </body>
29 </html>
30 )=====";
31
32 // change these values to match your network
33 char ssid[] = "BeagleboneAP";
34 char pass[] = "Azerty123";
35
36 WiFiServer server(80);
37
38 String tmpString = "";
39 unsigned int count = 0;
40
41 void setup()
42 {
43   Serial.begin(115200);
44   Serial.println();
45   Serial.println("Serial started at 115200");
46   Serial.println();
```

```
47 // Connect to a WiFi network
48 Serial.print(F("Connecting to "));  Serial.println(ssid);
49 WiFi.begin(ssid, pass);
50
51 while (WiFi.status() != WL_CONNECTED)
52 {
53     Serial.print(".");
54     delay(500);
55 }
56
57 Serial.println("");
58 Serial.println(F("[CONNECTED]"));
59 Serial.print("[IP ");
60 Serial.print(WiFi.localIP());
61 Serial.println("]");
62 Serial.println("]");
63
64 // start a server
65 server.begin();
66 Serial.println("Server started");
67
68 } // void setup()
69
70 void loop()
71 {
72     // Check if a client has connected
73     WiFiClient client = server.available();
74     if (!client) { return; }
75
76     count++;
77
78     tmpString = html_1;
79     tmpString.replace("%count%", String(count));
80
81     client.flush();
82     client.print(header);
83     client.print(tmpString);
84
85     Serial.print("count = "); Serial.println(count);
86
87     delay(5);
88
89 // The client will actually be disconnected when the ↴
```

```

    →function returns and 'client' object is destroyed
90 } // void loop()

```

14.4.4 De ESP8266 aansturen via het web

Het is mogelijk om de ESP8266 output pinnen te sturen via een web client. De ESP8266 die optreedt als webserver leest de data van de webclient en koppelt daar de gepaste actie aan.

LED aansturen via het web

De volgende listing is een voorbeeld om een LED aan te sturen, deze code bestaat uit een aantal blokken:

- lijn 3-6: initialisatie parameters
- lijn 8-26: netwerk en server starten
- lijn 30-41: wachten op binnenkomende connectie en tekst ontvangen
- lijn 43-51: tekst ontleden en eventueel LED besturen
- lijn 55-70: nieuwe pagina naar browser sturen
- lijn 72-74: connectie verbreken (refresh)

```

1 #include <ESP8266WiFi.h>
2
3 const char* ssid = "BeagleboneAP";
4 const char* password = "Azerty123";
5
6 int LED = D1;      // led connected to D1
7
8 WiFiServer server(80);
9
10 void setup(){
11     Serial.begin(115200); //Default Baudrate
12     pinMode(LED, OUTPUT);
13     digitalWrite(LED, LOW);
14
15     Serial.print("Connecting to the Network");
16     WiFi.begin(ssid, password);
17     while (WiFi.status() != WL_CONNECTED) {

```

```
18     delay(500);
19     Serial.print(".");
20 }
21 Serial.println("WiFi connected");
22 server.begin(); // Starts the Server
23 Serial.println("Server started");
24
25 Serial.print("IP Address of network: "); // will IP address ↴
26     → on Serial Monitor
26 Serial.println(WiFi.localIP());
27 }
28
29 void loop() {
30 WiFiClient client = server.available();
31 if (!client) {
32     return;
33 }
34 Serial.println("Waiting for new client");
35 while(!client.available()) {
36     delay(1);
37 }
38
39 String request = client.readStringUntil('\r');
40 Serial.println(request);
41 client.flush();
42
43 int value = LOW;
44 if(request.indexOf("/LED=ON") != -1) {
45     digitalWrite(LED, HIGH); // Turn LED ON
46     value = HIGH;
47 }
48 if(request.indexOf("/LED=OFF") != -1) {
49     digitalWrite(LED, LOW); // Turn LED OFF
50     value = LOW;
51 }
52
53 //-----HTML Page Code-----*/
54
55 client.println("HTTP/1.1 200 OK"); //
56 client.println("Content-Type: text/html");
57 client.println("");
58 client.println("<!DOCTYPE HTML>");
59 client.println("<html>");
```

```

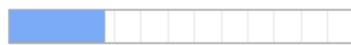
60     client.print(" CONTROL LED: ");
61     if(value == HIGH) {
62         client.print("ON");
63     }
64     else {
65         client.print("OFF");
66     }
67     client.println("<br><br>");
68     client.println("<a href=\"/LED=ON\"><button>ON</button></a>");
69     client.println("<a href=\"/LED=OFF\"><button>OFF</button></a><br />");
70     client.println("</html>");
71
72     delay(1);
73     Serial.println("Client disconnected");
74     Serial.println("");
75 }
```

Deze tekst ontvangt de ESP8266 van de browser:

- bij de eerste connectie: GET / HTTP/1.1
- bij een klik op een knop in de webpagina: GET /?LED=ON HTTP/1.1
- extra bij sommige browsers:
 - GET /favicon.ico HTTP/1.1
 - GET /browserconfig.xml HTTP/1.1

Progress Tag

Om een vooruitgang te tonen kan men een Progress Tag (figuur 14.7) gebruiken in HTML.



Figuur 14.7: Progress Tag

Voorbeeldfragment:

```

1   client.print("<progress value=\"250\" max=\"1023\"></progress>");
```

Range Tag

Om een waarde in te stellen kan men gebruik maken van een Range Tag (figuur 14.8).



Figuur 14.8: Range Tag

Bij deze Tag worden twee parameters meegeven:

- bij meerdere *ranges*: welke *range*?
- de waarde van de *range*

De HTML opbouw van een Range Tag zie je in onderstaande listing:

```

1 client.print("<input type=range " );
2 client.print("onchange=location.href=\"/?LED=3&V=\"+this.    ↴
   →value \"");
3 client.println("min=\"0\" max=\"100\" value=\"oudewaarde\">    ↴
   →");

```

Met `this.value` wordt de huidige instelling opgevraagd en terug meegegeven zodat de Tag blijft staan waar hij ingesteld stond door de gebruiker.

15

MicroPython

MicroPython is een slanke en efficiënte implementatie van de programmeertaal Python die een kleine subset van de Python-standaardbibliotheek bevat en is geoptimaliseerd voor gebruik op microcontrollers in beperkte omgevingen.

De taal bevat veel functies en een interactieve prompt. Toch is het compact genoeg om te passen en te werken binnen slechts 256k coderuimte en 16k RAM. MicroPython streeft ernaar om zo compatibel mogelijk te zijn met de normale Python, zodat je eenvoudig code van de desktop naar een microcontroller of embedded systeem kunt overbrengen.

MicroPython is een interpreter die werkt op het bordje zelf, je kan dus scripts en opdrachten via een prompt op het bordje zelf uitvoeren. Door zijn compactheid kan het o.a. geïmplementeerd worden op een ESP8266, ESP32 en een Raspberry PICO. De Arduino UNO bevat te weinig geheugen en kan niet gebruikt worden.

15.1 Python

De syntax van Python verschilt op een aantal vlakken van andere hogere programmeertalen, de voornaamste zijn

- lijnen code (bv. in lussen) worden gegroepeerd a.d.h.v. een insprong en niet met accolades
- er wordt geen gebruik gemaakt van puntkomma's op het einde van een lijn

Hieronder zie je een kort voorbeeld:

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
```

15.2 MicroPython op de ESP8266

MicroPython werkt a.d.h.v. een firmware die op de ESP8266 geïnstalleerd wordt. Je kan deze zelf downloaden en uploaden naar het bordje of je kan een tool gebruiken die je in een editor vindt zoals Thonny of uPyCraft. Deze tool downloadt de laatste firmware voor de gebruikte hardware en installeert die op het bordje.

Na de installatie kan je via de seriële poort over USB connectie maken met je bordje en kan je op de opdrachtprompt rechtstreeks opdrachten ingeven of een script uitvoeren.

Door dat het gebruik van MicroPython een andere firmware vereist op de ESP8266 zal je indien je je bordje terug in de Arduino IDE wenst te gebruiken de Arduino firmware (met bootloader) opnieuw geïnstalleerd moeten worden. Dit wordt automatisch uitgevoerd bij het uploaden van de code.

15.2.1 De REPL prompt

Met de MicroPython firmware is het mogelijk om rechtstreeks opdrachten uit te voeren op je ESP8266 bordje via de REPL prompt. De prompt kan zichtbaar gemaakt worden in een editor of kan via putty en een connectie met de juiste seriële poort gebruikt worden.

Op de REPL prompt kan je alle Python opdrachten één voor één uitvoeren of een script starten. De prompt is vooral handig om korte opdrachten te testen of te navigeren door de aanwezige scripts.

Het voorbeeld toont hoe je een pin hoog en laag kan maken via de prompt:

```
1 >>> import machine  
2 >>> pin = machine.Pin(2, machine.Pin.OUT)  
3 >>> pin.on()  
4 >>> pin.off()
```

15.2.2 Mogelijkheden

De MicroPython taal heeft heel wat mogelijkheden. De grote kracht is dat je gewone Python code kan (her)gebruiken.

In principe kan je alle mogelijkheden die voor de Arduino taal beschikbaar zijn ook in MicroPython gebruiken. Voor sommige hardware is het soms wel moeilijker om een geschikte bibliotheek te vinden.

15.2.3 Start up scripts

MicroPython maakt gebruik van 2 speciale scripts: `boot.py` en `main.py`.

Het script `boot.py` wordt eerst uitgevoerd bij het opstarten en kan handig zijn om bv. een netwerkverbinding via WiFi te maken en GPIO-pinnen in te stellen.

Na het uitvoeren van `boot.py` wordt `main.py` uitgevoerd. Hier kan je in principe de code plaatsen die je wil uitvoeren telkens als je bordje wordt herstart.

Alle andere scripts worden niet automatisch gestart bij het booten, tenzij je ze aanroept via één van de twee start up scripts.



Geschiedenis van de Computer

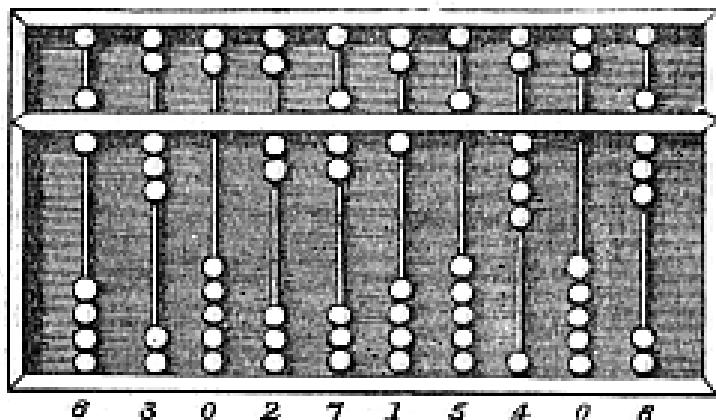
Computers zijn nog maar enkele tientallen jaren wijdverspreid. In 1971 werd de moderne computer geboren door het uitbrengen van de 4004 microprocessor door Intel. Deze chip is echter ontstaan uit een lange rij van uitvindingen.

A.1 Het Mechanisch Tijdperk

A.1.1 Abacus

De abacus (figuur A.1) is een telraam die het mogelijk maakt om op een snelle en logische manier berekeningen te maken. Ze is uitgevonden in de periode tussen 2400 en 300 voor Christus. Nog altijd wordt de abacus in verschillende vormen gebruikt en geoefende gebruikers rekenen hierop sneller dan rekenmachinegebruikers.

Een getal wordt nu weergegeven door een aantal kralen in een kolom tegen de middenbalk te schuiven, waarbij de bovenste kralen voor 5 tellen en de onderste voor 1. Het getal 1 is dus een kraal aan de onderkant tegen de middenbalk, 5 een kraal aan de bovenkant tegen de middenbalk, en 6 een kraal van zowel de boven- als de onderkant tegen de middenbalk. Stellen we op deze manier 6 in en tellen er 6 bij op, dan doen we dat door nog een bovenkraal en nog een onderkraal tegen de middenbalk te schuiven. De resulterende som 12 wordt weergegeven door de twee 5-kralen weer weg te schuiven en in plaats daarvan op de kolom links daarvan een eenhedenkraal tegen de middenbalk te zetten, die dan 1 tiental aangeeft (transport). Iedere kolom is een cijfer meer nauwkeurigheid, en de grootte van de berekeningen die kunnen worden uitgevoerd kan naar believen worden opgevoerd door het aantal kralenstaven te vergroten. (bijvoorbeeld door er een abacus naast te zetten). Een snelle schudbeweging kan worden gebruikt om het telraam op nul te zetten.



Figuur A.1: De Abacus

A.1.2 Charles Babbage (°1791 †1871)

Het principe van de moderne computer wordt door velen toegeschreven aan Charles Babbage en Lady Ada Lovelace, een Engelse wiskundige. Tot voor Babbage werd er gebruik gemaakt van arabische cijfers (0-9). Charles Babbage besliste gebruik te maken van het binair stelsel.

In het Verenigd Koninkrijk waren naar aanleiding van de koloniale scheepvaart veel centra met menselijke computers ontstaan. Deze maakten tabellen welke navigatoren voor navigatie konden gebruiken. Ook in andere gebieden vonden deze tabellen gretig aftrek, zoals de astronomie. Charles Babbage, een wiskundige, was erg geïnteresseerd in de astronomie. Een grote kwelling voor een astronoom was echter het feit dat in iedere tabel onvermijdelijk fouten zaten. Babbage vroeg zich af of de tabellen niet machinaal gegenereerd konden worden. Babbage ontwierp in 1821 een mechanische rekenmachine, de Difference Engine, om wiskundige tabellen te genereren. De machine werd echter maar voor een deel gebouwd en heeft nooit gewerkt. Afgebouwd zou hij uit 25.000 delen hebben bestaan en 15 ton hebben gewogen. In de periode van 1847 tot 1849 ontwierp hij de Difference Engine 2, die veel kleiner zou zijn geweest (4000 delen, 2,6 ton) maar eveneens niet werd gebouwd. Ook zouden ze niet kunnen werken, omdat de onderdelen met de stand van de techniek van dat moment niet klein genoeg gemaakt konden worden. Het idee was wel goed.

Van 1834 tot aan zijn dood in 1871 was Babbage bezig met het concept van de eerste programmeerbare (mechanische) rekenmachine, die hij de analytische motor noemde (Analytical Engine). Een waardige voorloper van de computer, omdat hij in principe alle functies van een echte computer zou hebben. Hij zou met ponskaarten werken, beslissingen nemen, berekeningen maken en uitkomsten onthouden. Als hij zou zijn gebouwd zou het een gigantisch apparaat geweest zijn, dat zou moeten worden aangedreven door een stoommachine. Het was in die tijd niet mogelijk om het apparaat te bouwen.

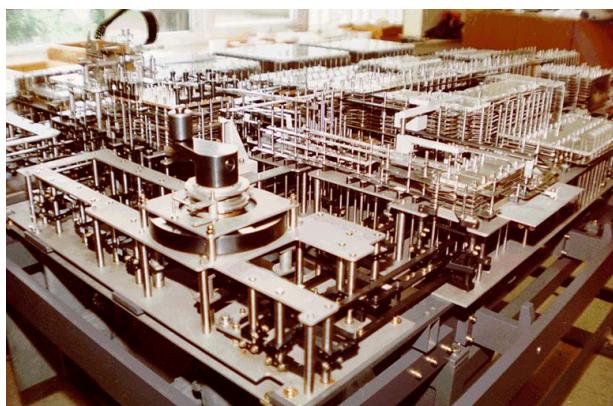
Ada Lovelace schreef voor de Analytical Engine van Babbage het eerste computerprogramma. Het feit dat een vrouw zich hiermee bezig hield was voor die tijd zeer bijzonder. Vrouwen werden in het onderwijs buitengesloten. Ze kreeg wiskundeles van privéleraren en was goed bevriend met Babbage. Omdat de machine niet gebouwd is, heeft ze haar programma's nooit zien werken.

Het Science Museum in Londen heeft in 1991 ter ere van de 200e geboortedag van Charles Babbage de Difference Engine No. 2 gebouwd en deze werkte inderdaad.

A.2 Het Elektrisch Tijdperk

A.2.1 Konrad Zuse ([°]1910 †1995)

Zuse werd geboren te Berlijn. Hij studeerde vanaf 1927 machinebouw, en haalde zijn diploma in 1935. In 1938 voltooide Zuse de Z1 (figuur A.2), de eerste programmeerbare rekenmachine ter wereld. Deze werkte nog geheel mechanisch, maar wel binair.



Figuur A.2: De Z1

In 1939 volgde de Z2, een proefmodel dat bestond uit het mechanische geheugen van de Z1, een kaartlezer, en een uit tweehonderd relais opgebouwde processor. En op 5 december 1941 was de eerste volledig elektromagnetische computer ter wereld een feit. Konrad Zuse lag met deze Z3 bijna twee jaar voor op de geallieerden.

In 1944 gingen de Z1 en de Z3 bij geallieerde bombardementen verloren. De bouw van de Z4, een 32-bits computer opgebouwd uit relais, werd in een kelder voortgezet. In 1949 werd Zuse met zijn Z4 voor vijf jaar gehuurd door het Instituut voor Toegepaste Wiskunde uit Zürich. De Z4 was toen de enige werkende computer op het vasteland van Europa. Hij werd onder meer gebruikt bij het ontwerpen van stuwdammen.

Van het Zwitserse geld kon Zuse zijn bedrijf weer heroprichten. Zuse begon met seriebouw van zijn computers. Zuse KG leverde onder meer aan Leitz. In de jaren zestig werd het bedrijf overgenomen door Siemens.

A.2.2 Enigma

Dit is de naam van een familie van elektromechanische codeermachines van het type rotormachines. Het toestel (figuur A.3) werd in de jaren '20 van de twintigste eeuw gecommercialiseerd door Chiffriermaschinen AG en in gebruik genomen door verscheidene Europese bedrijven, diplomatieke diensten en legers, maar werd vooral berucht als codeermachine van de Wehrmacht vóór en tijdens de Tweede Wereldoorlog in Nazi-Duitsland.



Figuur A.3: Voorbeeld van een Enigmatoestel

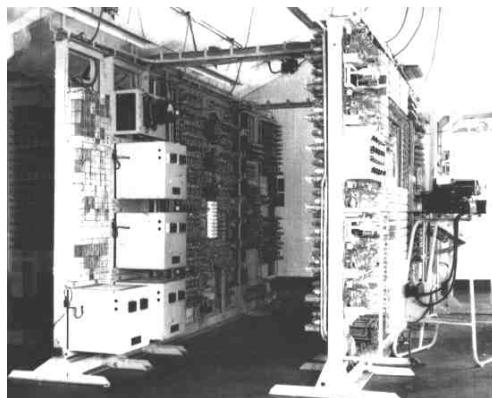
De Enigmamachine is een elektromechanisch systeem. Het toestel bestaat uit een toetsenbord, drie of vier draaiende contactschijven, rotors genaamd, die worden voortbewogen door een stappenmechanisme en een paneel met lampen. De combinatie van het elektrische circuit en de mechanische beweging noemen we de codering. Het drukken van een toets wordt via het elektromechanisch systeem vertaald in een oplichtende lamp, die de gecodeerde letter voorstelt.

Een belangrijke fout in het ontwerp van de Enigma was dat een letter nooit in zichzelf verscijferd werd. Ondermeer deze fout in combinatie met veel rekenkracht kon de code kraken. Hiertoe werd de Colossus ontworpen.

A.2.3 Colossus (1943)

De Colossus (figuur A.4) was de eerste elektronische computer en werd eind 1943 in productie genomen. Daarmee was de topgeheime computer twee jaar eerder in bedrijf dan de publiekelijk bekende ENIAC, de eerste Amerikaanse computer. Hij werd tijdens de Tweede Wereldoorlog door de Britten gebruikt om het Duitse berichtenverkeer te ontcijferen.

De Colossus was opgebouwd uit twee grote rekken met ongeveer 1500 elektronenbuizen voor de tellers, schuifregisters en logische bewerkingen. Hij had een systeem om ponsbanden te



Figuur A.4: Een Colossus computer

lezen met de Baudotcode, zoals ook de telexmachines in die tijd gebruikten, maar dan met fotosensoren in plaats van mechanische aftasting. Het leessysteem haalde een snelheid van 5000 karakters per seconde. De Colossus kon geprogrammeerd worden via een paneel met schakelaars, stekkers en kabels. Om geen gebruik te moeten maken van ponsbanden tijdens de cyclus van berekeningen gebruikte men een enorm aantal elektronenbuizen om de gegevens tijdens de verwerking op te slaan, wat meteen een aanzienlijke snelheidswinst betekende. Bovendien maakte het parallelle ontwerp van de Colossus hem ongelofelijk snel. Een pc met een Pentium I-processor, geprogrammeerd om dezelfde taak te verrichten, doet er eens zo lang over om dezelfde code te breken.

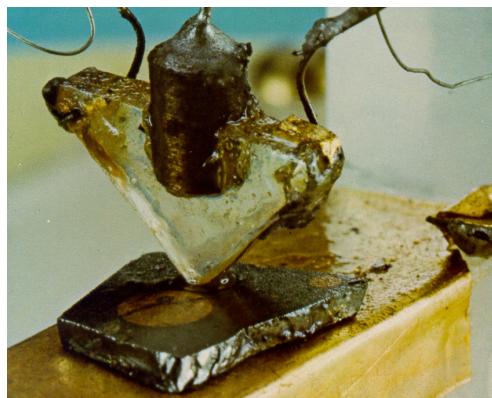
A.3 Belangrijke Uitvindingen

A.3.1 Transistor

De transistor (figuur A.5) werd in 1948 uitgevonden door John Bardeen, Walter Brattain en William Shockley, die er in 1956 de Nobelprijs voor de natuurkunde voor kregen. Binnen tien jaar bracht de transistor een revolutie in de computerwereld teweeg, en tegen het einde van de jaren vijftig waren de elektronenbuizen achterhaald.

A.3.2 Geïntegreerd circuit

De computers begonnen pas echt kleiner te worden in 1969 na de uitvinding van de geïntegreerde schakeling (IC). Op zo een 'chip' konden meerdere transistors worden samengevoegd. De complexiteit van de chips werd in de jaren 70 verbeterd zodat het mogelijk werd om een complete processor (CPU) op een chip te integreren. Het werd daardoor veel goedkoper om een computer te bouwen.



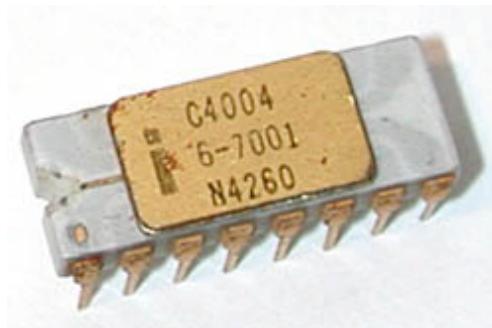
Figuur A.5: De eerste transistor

A.4 Het Microprocessor Tijdperk

A.4.1 Intel 4004 en 8008

In 1968 werd Intel Corporation opgericht voor het maken van geheugenchips. Kort daarna werd het bedrijf benaderd door een bedrijf van calculators die één chip wilde voor een calculator en door een fabrikant van terminals die een uit één chip bestaande controller voor zijn terminal wilde. Intel maakte deze twee chips, de 4004 (figuur A.6) , een 4-bits CPU, en de 8008, een 8-bits CPU. Dit waren **de eerste uit één chip bestaande CPU's**.

Intel verwachtte niet dat er nog anderen dan de oorspronkelijke klanten in deze chips geïnteresseerd zouden zijn, zodat men zich op een kleine productie instelde. Maar de mensen bij Intel kregen ongelijk. Er was enorm veel belangstelling, zodat ze aan de slag gingen om een general-purpose CPU-chip te ontwerpen. Dit ontwerp leidde tot de 8080. Deze chip veroverde de markt op een stormachtige manier en Intel verkocht er miljoenen van.



Figuur A.6: De Intel 4004

Enkele technische specificaties:

- maximale kloksnelheid van 740kHz
- 0,06 MIPS (miljoen instructies per seconde)
- 2.300 transistoren
- 4-bits data

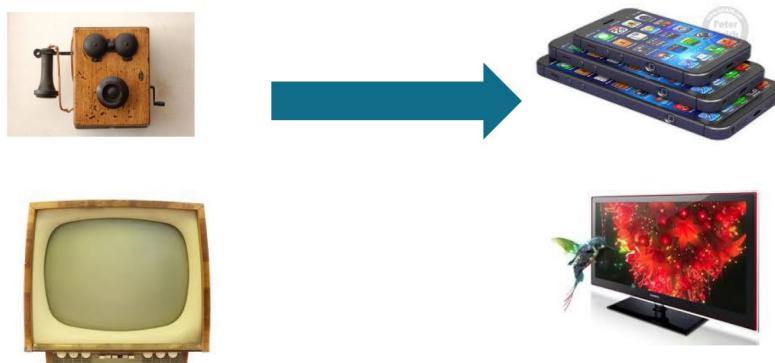
A.4.2 Intel 8086

In 1978 kwam intel met de 8086 op de proppen. Het was de eerste 16-bit processor en ook de eerste processor met de x86-instructieset. De eerste commerciële microcomputer gebaseerd op de 8086 was de Mycron 2000. De IBM PC, de originele versie van het IBM PC compatible platform, was gebaseerd op het broertje van de 8086, namelijk de 8088. Deze werkte intern als de 8086, maar had naar de buitenkant slechts een 8-bit bus.

De 8086 had de volgende kenmerken:

- kloksnelheid van 4,77 tot 10 MHz
- 0,33 (bij 4,77 MHz) tot 0,75 MIPS (bij 10MHz)
- 29.000 transistoren
- 16-bits data

A.4.3 Systemen worden complexer



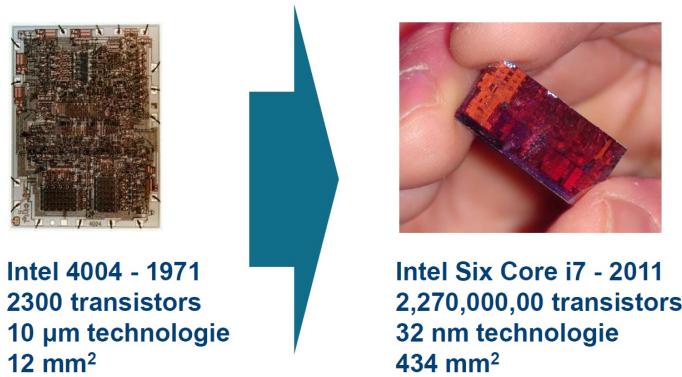
Figuur A.7: Evolutie telefoon en televisie

In de recente geschiedenis is het duidelijk dat elektronische systemen steeds complexer worden in de loop van de jaren (figuur A.7) . Twee voorbeelden maken dit duidelijk. De telefoon van 100

jaar geleden was in feite een elektrisch toestel zonder al te veel componenten. De smartphone van vandaag is een volledige computer die veel meer toelaat dan enkel spraakoverdracht.

De televisie van 60 jaar geleden was een elektronisch toestel gebaseerd op buizen, de voorloper van de transistor. Het principe is nog steeds hetzelfde, namelijk beeld vanop afstand tonen, maar de huidige toestellen zijn dankzij het invoeren van complexe IC's lichter, veelzijdiger geworden en leveren een veel beter beeld.

De eerste microprocessoren zijn qua aantal transistoren en complexiteit (figuur A.8) in het niks te vergelijken met de huidige exemplaren.



Figuur A.8: Evolutie transitoren

Om een idee te geven van de evolutie tonen onderstaande voorbeelden de vergelijking met een aantal legoblokjes t.o.v. het aantal transistoren. De figuren maken veel duidelijk.

- de 8086: 1978 , 29.000 transistors (figuur A.9)



Figuur A.9: Vergelijking 8086

- de Pentium: 1993 , 3.300.000 transistors (figuur A.10)



Figuur A.10: Vergelijking Pentium

- de Pentium 4: 2000 , 55.000.000 transistors (figuur A.11)



Figuur A.11: Vergelijking Pentium 4

- de Core i7: 2011 , 1.200.000.000 transistors (figuur A.12)

A.4.4 MIPS

Om de verwerkingssnelheid van een computer aan te duiden, zou gebruik gemaakt kunnen worden van de klokfrequentie. Maar het probleem hiervan is dat de ene processor gemiddeld 20 klok pulsen nodig heeft per instructie en de ander 100 per instructie. Daarom werd de MIPS-eenheid ingevoerd. Dat staat voor **miljoen instructies per seconde** en wordt berekend door de klokfrequentie te delen door het gemiddeld aantal klok pulsen die nodig zijn voor een instructie:



Figuur A.12: Vergelijking Core i7

MIPS = kloksnelheid CPU in MHz / gemiddeld aantal benodigde klok pulsen per instructie

Zo is het aantal MIPS bij een 8086 0,33 tot 0,75 (afhankelijk van de kloksnelheid) over 1354 bij een PIII tot 61.119 bij een Intel Core 2 Extreme QX6700.

Tegenwoordig wordt ook het aantal MIPS per Watt steeds belangrijker.

Oefeningen

Oefening 1

Bereken het aantal MIPS van de volgende processoren:

- microprocessor A: 1GHz - 100 klok pulsen/instructie
- microprocessor B: 2GHz - 250 klok pulsen/instructie
- microprocessor C: 3GHz - 500 klok pulsen/instructie



Project prototyping

Als we willen starten met een nieuw Arduino project komt er heel wat bij kijken. Zo moeten we ons de vraag stellen wat we juist willen doen en welke hardware we gaan gebruiken zoals sensoren en displays.

Daarna kan er een elektronisch schema ontworpen worden en kan de software voor de sketch geschreven worden.

B.1 Project benodigdheden

Om de juiste benodigde hardware te bepalen moeten we ons een aantal algemene vragen stellen:

- wat willen we controleren en/of meten?
- welke data willen we meten?
- welke berekeningen moeten er gemaakt worden?
- willen we de informatie tonen?
- moet de data bijgehouden worden en hoelang?
- is er netwerkverbinding nodig?

Door deze vragen één voor één te beantwoorden krijgen we reeds een globaal zicht op de benodigdheden.

B.1.1 Temperatuurmonitor

Als voorbeeld willen we een temperatuurmonitor ontwerpen die aantoont of het warm of koud is.

De temperatuur moet gemeten worden met een temperatuursensor (dit kan bv. met een TMP36), we moeten de comforttemperatuur kunnen instellen (bv. met een potentiometer) en ten slotte willen we kunnen tonen of het te warm of te koud is, dit kan bv. met een vijftal LED's die de toestand aangeven (figuur B.1).

- 1 groene LED: temperatuur is max. 5 graden hoger of lager dan comfortwaarde
- 2 gele LED's: een afwijking tussen de 5 en 10 graden
- 2 rode LED's: meer dan 10 graden afwijking



Figuur B.1: Temperatuurmonitor

Aangezien we enkel de huidige temperatuur willen controleren hoeven we de data niet bij te houden. Verder is er geen netwerkverbinding nodig omdat we enkel ter plaatse de toestand willen kunnen zien.

B.2 Benodigde interfaces

De Arduino beschikt over een beperkt aantal pinnen en interfaces: het aantal interfaces van het te ontwerpen project moet dus kleiner zijn dan het aantal beschikbare interfaces op de Arduino.

Analoge ingangen kunnen gebruikt worden om analoge sensors in te lezen, digitale pinnen kunnen zowel als input en output gebruikt worden als voor analoge output (PWM).

De protocollen I²C, SPI en Serieel gebruiken vaste pinnen op de Arduino en kunnen indien het protocol gebruikt wordt niet meer voor iets anders gebruikt worden.

In het voorbeeld van de temperatuurmonitor hebben we twee analoge ingangen nodig (1 voor de temperatuursensor en 1 voor de potentiometer) en 5 digitale uitgangen voor de LED's. Een Arduino UNO volstaat dus prima.

B.3 Lijst met componenten

Om de juiste lijst met benodigde componenten op te stellen kunnen we een lijstje nagaan:

- sensoren
- display onderdelen: LED's, LCD, ...
- knoppen
- motoren
- extra onderdelen: weerstanden, transistoren, ...
- voeding

Zo hebben we om een motor aan te sturen een extra transistor of een H-brug nodig. Voor een LED is dan weer een serieweerstand noodzakelijk.

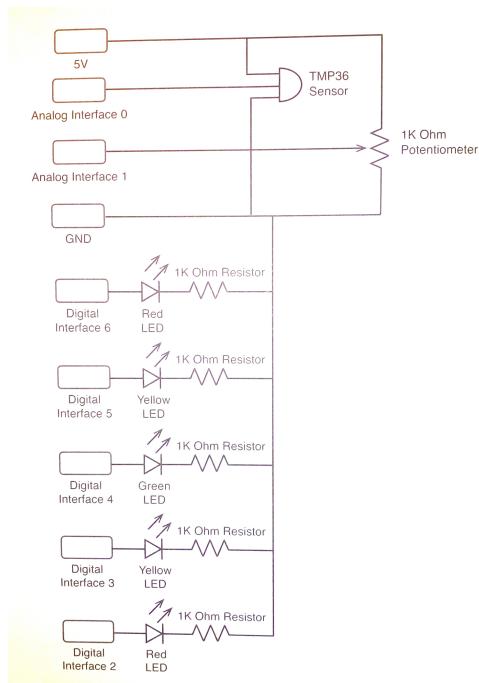
B.3.1 Temperatuurmonitor

Voor het voorbeeld hebben we de volgende componenten nodig:

- 5 LED's
- weerstanden voor de LED's
- TMP36 temperatuursensor
- potentiometer
- Arduino UNO bordje
- adapter (voeding)

B.4 Schema tekenen

Samen met het bepalen van de componenten kunnen we ook het schema tekenen, zie figuur B.2.



Figuur B.2: Schema temperatuurmonitor

B.4.1 Breadboard opstelling

Het is onnodig om een print te maken bij een prototype van een (klein) project. We kunnen de opstelling eerst testen op een breadboard. Dit is gemakkelijk om eventuele wijzigingen aan te brengen.

Na een succesvolle test kan er een print ontwerpen worden indien nodig. De uiteindelijke schakeling en de Arduino kunnen eventueel gevoed worden via een adapter i.p.v. de USB-aansluiting.

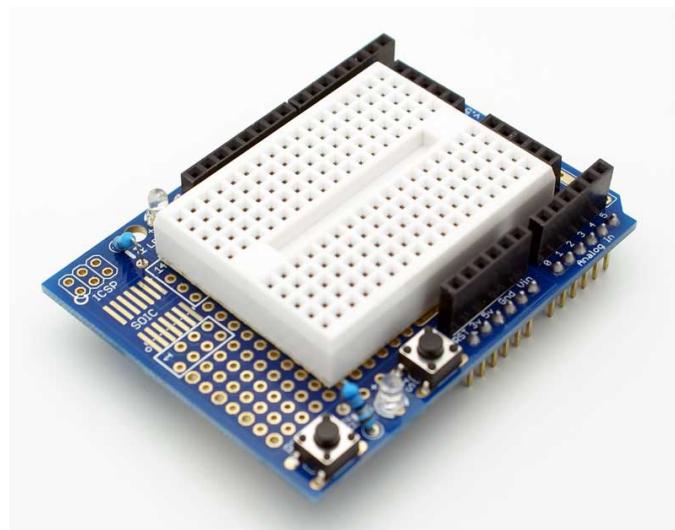
Kleine opstellingen kunnen gebruik maken van een breadboard-shield zoals te zien is in figuur B.3.

B.5 De sketch schrijven

Vooraleer we starten met het schrijven van code moet er bepaald worden welke bibliotheken er nodig zijn.

Ook globale variabelen en constanten kunnen min of meer op voorhand gedefinieerd worden.

Ten slotte kan je bepalen welke code er éénmalig moet uitgevoerd worden (in `setup()`) en wat telkens herhaald moet worden (in `loop()`).



Figuur B.3: Breadboard-shield

Voor grotere projecten zijn er meestal ook extra functies en/of bibliotheken nodig.

B.5.1 Temperatuurmonitor

De functies `setup()` en `loop()` kunnen als volgt geïmplementeerd worden:

`setup():`

- seriële poort initialiseren (seriële monitor wordt gebruikt voor het debuggen)
- de gebruikte digitale interfaces definiëren als input of output (analoge inputs moeten niet op voorhand ingesteld worden)

`loop():`

- de temperatuursensor uitlezen en de temperatuur berekenen
- de potentiometer uitlezen en omrekenen naar de gewenste temperatuurswaarde
- de ingelezen waarde vergelijken met de ingestelde en de LED's aansturen

De temperatuur opvragen en de LED's instellen kan in dit voorbeeld bv. in aparte functies gecodeerd worden.

```
1 #define LED1 2  
2 #define LED2 3
```

```
3 #define LED3 4
4 #define LED4 5
5 #define LED5 6
6 #define tempsensor A0
7 #define potmeter A1
8
9 int temp;
10 int setting;
11
12 void setup() {
13     pinMode(LED1, OUTPUT);
14     pinMode(LED2, OUTPUT);
15     pinMode(LED3, OUTPUT);
16     pinMode(LED4, OUTPUT);
17     pinMode(LED5, OUTPUT);
18     Serial.begin(9600);
19 }
20
21 void loop() {
22     int scale;
23     scale = map(analogRead(potmeter), 0, 1023, 0, 254);
24     temp = getTemp(analogRead(tempsensor));
25     setting = getTemp(scale);
26     Serial.print("Temp: ");
27     Serial.print(temp);
28     Serial.print(" setting: ");
29     Serial.println(setting);
30     checkTemp(temp, setting);
31     delay(1000);
32 }
33
34 int getTemp(int value) {
35     float voltage, tempC, tempF;
36     voltage = value * (5000.0 / 1024.0);
37     tempC = (voltage - 500) / 10;
38     tempF = (tempC * 9.0 / 5.0) + 32.0;
39     return int(tempC);
40 }
41
42 void checkTemp(int temp, int setting)
43 {
44     if (abs(temp - setting) < 5)
45     {
```

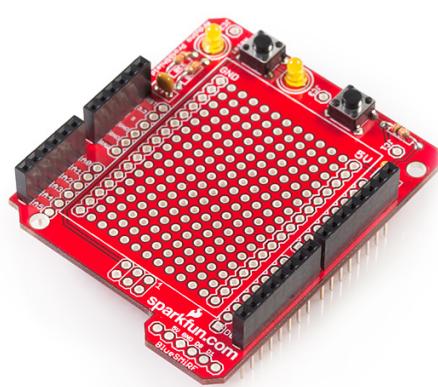
```
46 // temperatuur in comfortzone
47 digitalWrite(LED5, LOW);
48 digitalWrite(LED4, LOW);
49 digitalWrite(LED3, HIGH);
50 digitalWrite(LED2, LOW);
51 digitalWrite(LED1, LOW);
52 }
53 else if (((temp - setting) > 0) && ((temp - setting) < 10))
54 {
55 // temperatuur te warm
56 digitalWrite(LED5, LOW);
57 digitalWrite(LED4, HIGH);
58 digitalWrite(LED3, LOW);
59 digitalWrite(LED2, LOW);
60 digitalWrite(LED1, LOW);
61 }
62 ...
63 }
```

B.6 Testen

Als alles klaar is kan de software en de hardware uitgetest worden. De seriële monitor kan daarbij gebruikt worden als debugger. Eventueel kunnen bepaalde componenten bij twijfel uitgemeten worden met een multimeter.

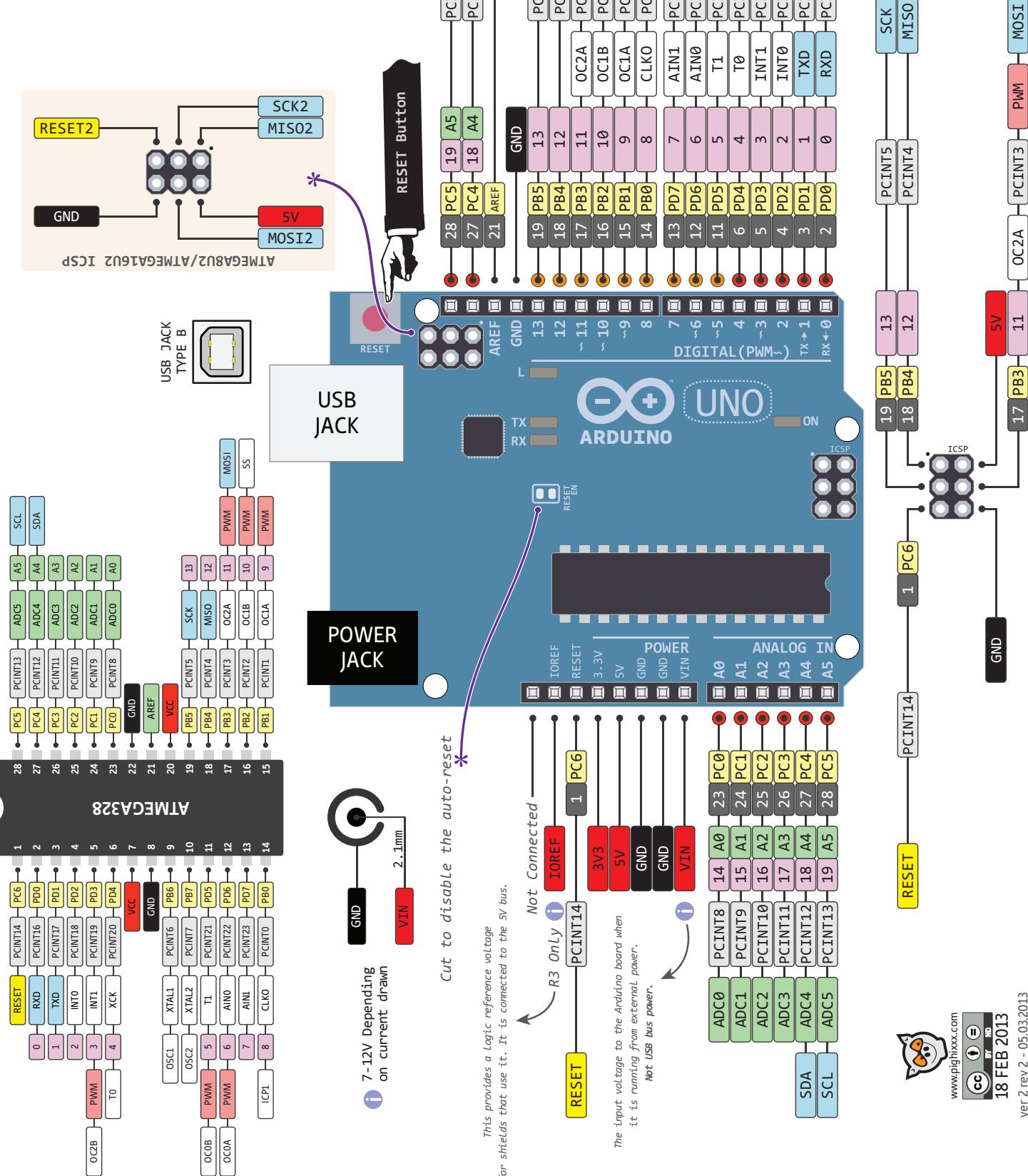
Als alles correct werkt op het breadboard kan men een print maken of een prototype board (*proto-shield*) gebruiken.

Een voorbeeld van een *proto-shield* zie je in figuur B.4.



Figuur B.4: Proto-shield

THE DEFINITIVE ARDUINO UNO PINOUT DIAGRAM



ARDUINO CHEAT SHEET

Content for this Cheat Sheet provided by Gavin from Robots and Dinosaurs.
For more information visit: <http://arduino.cc/en/Reference/Extended>