



Inhoud:

Wie zijn wij ?	5
Introductie [Module 0]	6
WAT IS EEN ARDUINO ?	7
WAT GAAN WIJ GEBRUIKEN OM TE LEREN PROGRAMMEREN MET ARDUINO ?	8
ESP32 CHIP	8
Arduino programmeren - Opzet - [Module 1]	9
WAT IS EEN “IDE” EN WAT IS EEN “COMPILER” ?	9
ARDUINO PORTABLE versie	9
HOE INSTALLEREN WE ARDUINO IDE OP JE COMPUTER ?	11
INITIËLE CONFIGURATIE VAN DE ARDUINO SOFTWARE	13
INSTALLEREN VAN DE BIBLIOTHEEK “E4SBoard”	17
TESTEN VAN ONZE ESP32 DEVICE	18
Arduino programmeren - Uitvoer - [Module 2]	20
EEN KORTE VERKENNING VAN DE ARDUINO IDE	20
COMMUNICATIE MET DE ESP32	21
WAT HEEFT USB TE BIEDEN BIJ ARDUINO PROGRAMMEREN ?	21
1. USB (Universal Serial Bus) kan de ESP32 voorzien van stroom.	21
2. USB als middel om programma's naar de ESP32 te sturen.	21
3. USB kan data ontvangen van de ESP32.	21
SERIËLE COMMUNICATIE OVER USB	21
SERIËLE MONITOR – GEGEVENS VAN DE ESP32 ONTVANGEN	23
DE BASIS ARDUINO PROGRAMMA LAY-OUT	25
Setup() en Loop()	25
ESP32 PROGRAMMEREN MET OPMERKINGEN	27
INSTRUCTIES EN CODE BLOKKEN	28
Een code blok start en eindigt met accolades “{}”	28
Een instructie eindigt altijd met een puntkomma “,”	28
Stroomkring - [Verdieping 1]	29
WAT IS EEN STROOMKRING ?	29
STROOM METEN IN ONZE STROOMKRING	33
WEERSTAND METEN	34
CODERING VAN WEERSTANDEN EN HOE AF TE LEZEN	35
Opbouw e-4S ESP32 device [Module 3]	38
De “ESP32 NodeMCU-32S”	40
Blink [Module 4]	43
Knipperende LED	43
OPDRACHTEN :	49
We willen een “Vermoeidheids Indicator” maken.	49
We willen een “SOS signaal” laten zien.	49
Spoorwegovergang	50

Relais - [Verdieping 3]	52
Definitie :	52
Constructie :	52
Werkingsprincipe :	52
Variabelen [Module 5]	54
Wat is een variabele	55
Naam van een variabele	55
DATATYPE	55
Hoe gaan we tewerk om variabelen te gebruiken ?	55
Bereik van een variabele (scope)	58
Variabelen die niet veranderd kunnen worden.	59
Overzicht van alle typen variabelen	60
Boolean	60
Char	60
byte	60
int	60
unsigned int	61
long	61
unsigned long	61
float	61
double	61
string	61
array	61
Met variabelen werken	62
Oefening 1	62
Oefening 2	63
Oefening 3	64
Oefening 4	65
Bewerkingen uitvoeren met variabelen	66
Oefening 5	66
Oefening 6	67
Operators [Module 6]	68
Rekenkundige Operators.	68
Vergelijking Operators	69
Boolean Operators	70
AND (&&)	70
OR ()	72
NOT (!)	73
Samengestelde (Compound) Operators.	73
Beslissingen [Module 7]	75
If ... else ...	75
Switch ... case ...	81
FLOWCHART	83

Lussen [Module 8]	85
FOR - loop	85
WHILE – loop	87
“DO ... WHILE ...” LOOP	88
Digitale input en output [Module 9]	90
Drukknop vs schakelaar	90
LED in- en uitschakelen.	91
Gestructureerd programmeren	95
Shift Register uitgangen	100
Multiplexer ingangen	102
Ingangen van het type SINK	102
Ingangen van het type SOURCE	103
Machten [Verdieping 4]	107
Getalstelsels [Verdieping 5]	109
Bit en Byte [Verdieping 6]	114
Flowchart [Verdieping 7]	115
Digitale poorten [Verdieping 8]	117
Extra uitbreiding	125
Wat is dat allemaal? GPIO, SPI, I2C, en UART communicatie	125
GPIO / General Purpose Input Output	125
SPI / Serial Peripheral Interface	125
Meer info over het gebruik van SPI voor verschillende platformen:	126
I2C / Inter-Integrated Circuit	127
UART / Universal Asynchronous Receiver / Transmitter	128
Ditjes en datjes Analooog vs Digitaal	130
Digitaal en analooog	130
Digitale systemen	131
Getalsystemen	131
Decimaal getal systeem	131
BINAIR getal systeem	131
HEXADECIMAAL getal systeem (h of \$ of 0x)	132
Rekenen met binaire getallen	134
Verband binaire getallen en digitale signaalverwerking	134
Binaire codesystemen	135
BCD-code	135
GRAY-code	135
ASCII-code	136
Ditjes en datjes over	137
Microprocessor, Micro Computer, Microcontroller	137

Ditjes en datjes over Programmeertalen	138
Een IDE (Integrated development environment)	138
Analoog wordt digitaal (en omgekeerd)	139
Communicatie tussen digitale apparaten	141
PARALLEL	141
SERIEEL	141
Communicatiesystemen en protocollen	142
Parallelle communicatie	142
Seriële communicatie (asynchroon)	142
Synchrone seriële communicatie: I2C	142
PROTOCOL	143
Voorbeeldprogramma gebruik LCD met e-4S board	144
Notities	145



Kennis delen en van elkaar leren versterkt de samenwerking!

Wie zijn wij ?

Wij zijn mensen die zich door zelfstudie, onderlinge communicatie en technische onderzoeken, aan de radiotechniek interesseren en dit voor strikt persoonlijke motieven zonder financieel belang. Over de ganse wereld zijn er meer dan 3 miljoen radioamateurs met een zendmachtiging.

In België zijn er ongeveer 3.000 mensen aangesloten bij de UBA, die ons vertegenwoordigt bij de Internationale Telecommunicatie Unie, de ITU. De radioamateurs zijn zowat over de ganse wereld erkend voor hun nuttige dienstverlening, bvb. bij natuurrampen, hun ernst, pedagogische en didactische waarde.

Het BIPT (Belgisch Instituut voor Post en Telecommunicatie) is het regulerend orgaan voor deze diensten, en o.a. ook voor de toepassing van de wetgeving voor radioamateurs. Het bezit en uiteraard ook het gebruik van een radiozender zonder vergunning, (dus zonder machtiging) is bij wet verboden.

Elk jaar richt het BIPT examens in voor het bekomen van een radioamateur certificaat waarmee een vergunning kan worden aangevraagd. Het radioamateurstation, het type van uitzendingen (modulatiemethode), en de aard ervan moeten voldoen aan de wettelijke eisen., zo is bvb. het uitzenden van muziek verboden, evenals commentaar over politiek enz... De frequentiebanden die door de amateurs mogen gebruikt worden zijn nationaal en internationaal vastgelegd, en omvatten praktisch het ganse frequentiespectrum die nuttig zijn voor dit doel. Zo proberen radioamateurs over gans de wereld een "QSO" te voeren, d.i. een verbinding te maken. , In dergelijk qso wordt informatie doorgestuurd over gebruikte toestellen, antennes, het weer en over de voortplanting condities. Radioamateurs maken zich bekend door het melden van hun callsign, hun naam en woonplaats. Een qso gebeurt altijd in een vriendschappelijke sfeer los van politieke en religieuze opvattingen.

Ter bevestiging van de verbinding worden er QSL kaarten uitgewisseld. er worden ook Contesten ingericht (wedstrijden met puntenverdeling).

Een CB'er hoeft geen examen af te leggen, maar is beperkt tot het gebruiken van de 11 meter band, Veel radioamateurs zijn evenwel als CB'er begonnen. Ieder land heeft zijn eigen roepletters (Call), voor België is dit ON, gevolgd door een nummer en 2 à 3 letters, bvb ON4IPR. (bij speciale gelegenheden ook nog OP, OQ, OR, OS en OT)

Interesse neem contact op met onze voorzitter:



[ON4JES]

Jess Versailles

E-mail : on4jes@uba.be

Gsm +32 (0)487 309927

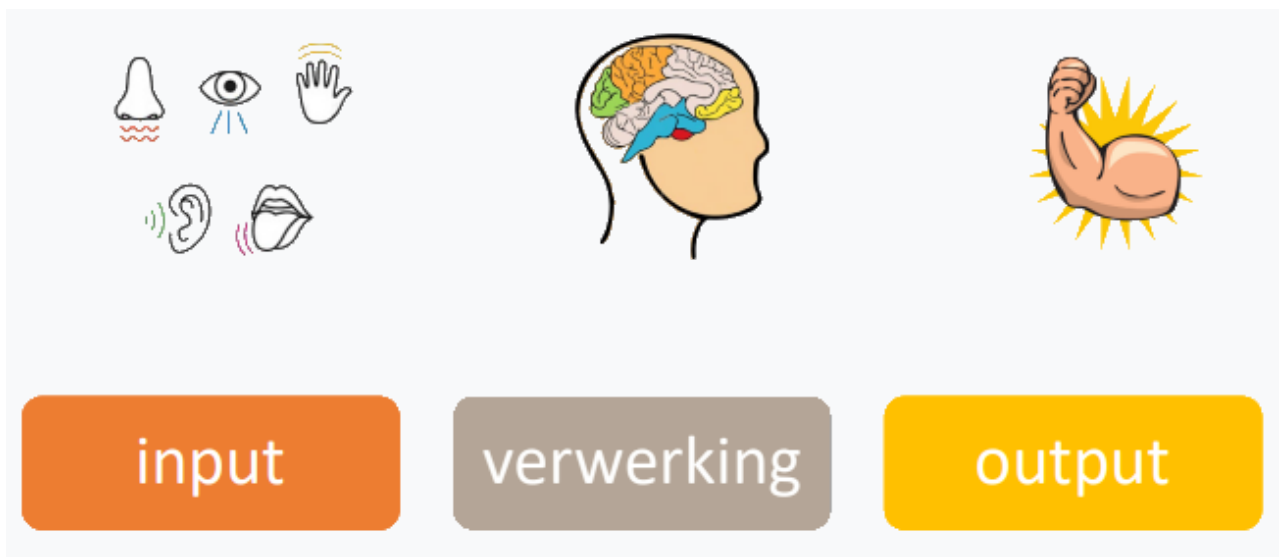
Introductie [Module 0]

Je bent begonnen in een cursus waarin uitgelegd wordt hoe je je eigen aansturingen kunt maken! Je leert in een snel tempo niet alleen hoe je een lampje kunt laten knipperen, maar ook hoe je displays en verschillende typen motoren kunt besturen.

Je staat op het punt een nieuwe wereld aan mogelijkheden te ontdekken met een heel moderne microcontroller. We zouden een microcontroller gestuurd systeem best kunnen vergelijken met een levend wezen ...

*Via de zenuwen (**draden**) worden signalen naar de hersenen (**microcontroller**) gebracht. Dit noemen we voelen (**INPUT**). Na verwerking in de hersenen (**microcontroller**) worden signalen via de zenuwen (**draden**) naar onder andere spieren (**actuatoren**) gestuurd. De spieren (**actuatoren**) kunnen dan bewegingen uitvoeren (**OUTPUT**).*

*De manier van sturen wordt door de hersenen (**microcontroller**) geregeld. De hersenen (**microcontroller**) reageren op een bepaalde manier (**verwerking**) op signalen van buitenaf*



Het is natuurlijk zo dat menselijke hersenen veel complexer ineen steken, dus deze vergelijking is nogal oppervlakkig. Toch is het zo dat naarmate de techniek zich verder ontwikkeld, de microcontroller steeds meer op ons menselijk brein zal lijken. Microcontrollers en hun programmeeromgeving zullen steeds geavanceerder en krachtiger worden, waardoor ze het gedrag van het menselijk brein zullen kunnen benaderen, evenaren of zelfs voorbijstreven ! Het is een kwestie van tijd !

WAT IS EEN ARDUINO ?

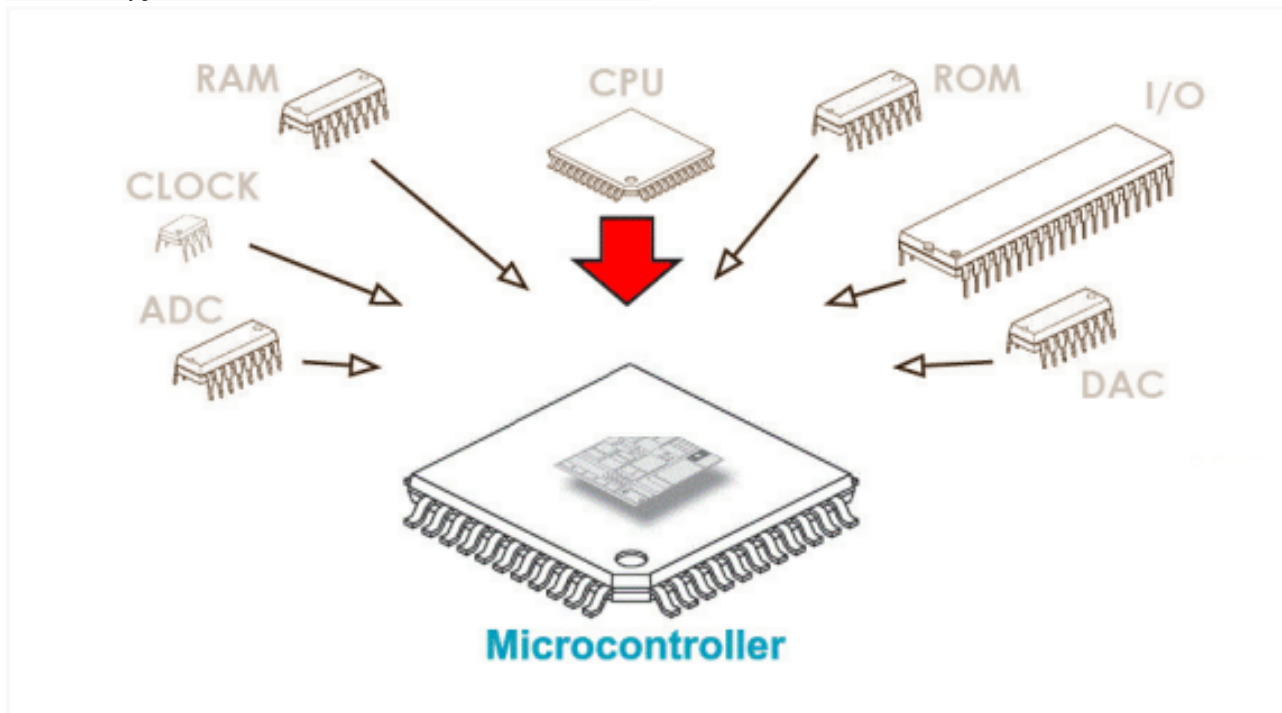
Simpel uitgelegd is de Arduino een klein printplaatje met wat elektronica onderdelen opgezet rond een zogenaamde microcontroller. Een microcontroller is eigenlijk een simpele, maar relatief complete mini computer.

Het is stukken minder krachtig in vergelijking met een desktop of laptop computer of zelfs een mobiele telefoon. Je vindt ze overal in huis, zoals in een wekker, wasmachine, magnetron enz...

Arduino is een de-facto standaard geworden voor hobbyisten die microcontrollers willen (leren) gebruiken en toepassingen er willen voor ontwikkelen. Arduino is volledig open source: de software is gratis te gebruiken en de hardware is betaalbaar. Compatibele hardware is goedkoop tot zeer goedkoop.

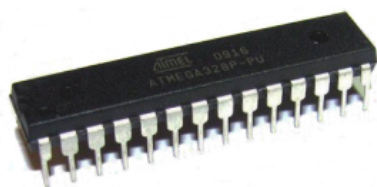
Een specifieke microcontroller die op de Arduino gebruikt wordt, is de zogenaamde Atmel AVR. Atmel is de fabrikant en AVR is het serie model.

Deze chip bevat onder andere een microprocessor (CPU), geheugen (ROM, RAM) en programmeerbare in-en uitgangen (I/O) die het mogelijk maken om allerlei zaken aan te sturen zoals lampjes, schakelaars, motoren, relais enz...

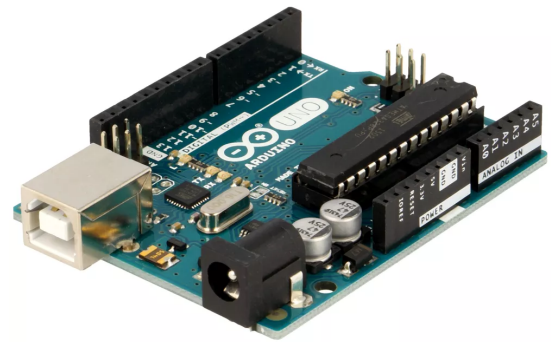
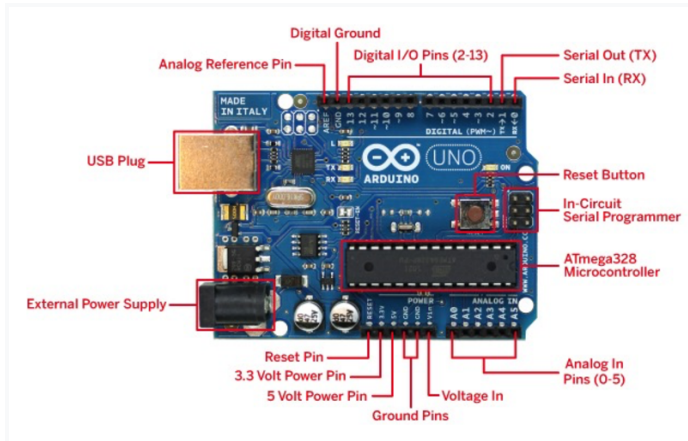


De microcontroller is slechts een enkele chip, wat het aansluiten en experimenteren een beetje lastiger maakt.

Voorbeelden van een Atmel AVR chip



Om het aansluiten wat gemakkelijker te maken heeft Arduino een printje gemaakt, waarop de microcontroller geïntegreerd is.

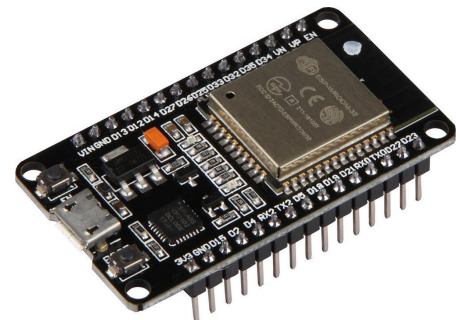


[Arduino Uno]

Over de jaren heen hebben ze een aantal modellen en varianten ontwikkeld met verschillend geheugen, smd uitvoeringen, snelheid, prijs enz ...

WAT GAAN WIJ GEBRUIKEN OM TE LEREN PROGRAMMEREN MET ARDUINO ?

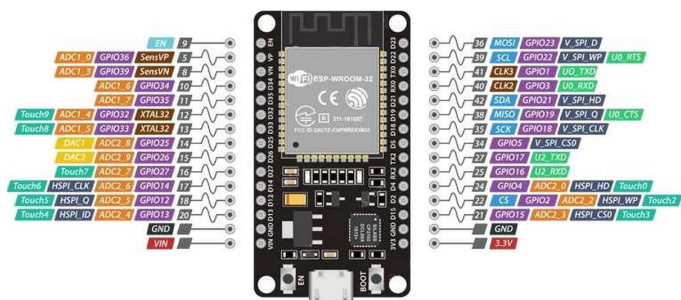
Wij zullen werken met de ESP32. Dit is een microprocessor systeem die volledig vervat is in een chip met onder andere WIFI en Bluetooth aan boord. De ESP32 is ontwikkeld door de firma Espressif Systems en is volledig compatibel met Arduino IDE. We zullen "over the air" (OTA) kunnen programmeren. (draadloos) Wellicht zullen een aantal termen en benamingen nog niet direct duidelijk zijn ! Maak je hierover geen zorgen ! Dit wordt wel duidelijk naarmate je verder vordert in deze cursus !



ESP32 CHIP

De ESP32 is een vernieuwde en uitgebreide versie van de ESP-chips. De voornaamste verschillen zijn:

- Double-core 32-bit CPU LX6
- Kloksnelheid: 80 of 160 MHz
- 36 GPIO's (digitale input/output pinnen)
- 2x SPI, 2x I2C, 2x I2S, 2x UART, 1x CAN
- 10 Touch sensoren, temperatuur sensor
- Bluetooth 4.2
- 18ADC- kanalen, 12 bit



ESP32 Dev. Board Pinout

Arduino programmeren - Opzet - [Module 1]

WAT IS EEN “IDE” EN WAT IS EEN “COMPILER” ?

Een **IDE** is een geïntegreerde ontwikkelomgeving of wel een “Integrated Development Environment”. Wat eigenlijk wil zeggen dat alle programma’s en software, die we nodig hebben om te programmeren bij elkaar zitten in een handig pakketje. Je hoeft dus niet op zoek te gaan naar allerlei onderdelen en in ons geval is dat pakketje gewoon gratis. Je kunt de **desktop versie** van de Arduino website **downloaden** op volgend adres: <https://www.arduino.cc/en/software>

Opgelet: Er bestaat een **online versie van de Arduino IDE**. Je gebruikt die zonder iets te installeren. Alle software die je maakt wordt bewaard in de cloud en je beschikt steeds over de allerlaatste versies van de software. Deze versie ondersteunt de ESP-chips (nog) niet. **We gebruiken deze dan ook niet.**

Downloads



Arduino IDE 1.8.19

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

SOURCE CODE

Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

DOWNLOAD OPTIONS

Windows Win 7 and newer
Windows ZIP file

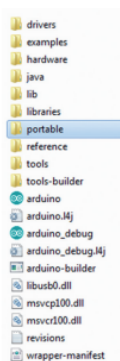
Windows app Win 8.1 or 10 [Get](#) 

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Mac OS X 10.10 or newer

[Release Notes](#)

[Checksums \(sha512\)](#)



ARDUINO PORTABLE versie

De portable versie van de IDE installeert alle bestanden en folders op één plaats zonder andere folders van de computer te beïnvloeden. De portable IDE kan gebruikt worden door studenten die niets kunnen installeren op de schoolcomputers of radioamateurs die niet houden van rommel op hun computer: installeer alles op een externe schijf, de gemaakte sketches komen ook op die schijf terecht. Dit systeem kan je echter op verschillende computers gebruiken.

Het IDE “pakket” heeft een programma om tekst in te voeren (een zogenaamde tekst editor) waarmee we de **code (instructies)** van ons programma kunnen invoeren. Daarnaast hebben we **hulpmiddelen** om verbinding met onze ESP32 te maken en een aantal **bibliotheken (libraries)**. Dit zijn **verzamelingen met code** om ons werk te besparen en een zogenaamde **compiler**. Vertalen ...

De compiler is een van de belangrijkste onderdelen, omdat we ons programma in een hogere taal schrijven. Daarmee bedoelen we dat de “taal” of “programmeertaal” dichter bij de menselijke taal zit dan bijvoorbeeld machinetaal. In ons geval gebruiken we de **programmeertaal “C”**.

“C” is een taal die wat weg heeft van de Engelse taal, waardoor het wat makkelijker te lezen is.

De programmeertaal “C” bestaat al erg lang en er zijn reeds heel wat varianten in omloop.

Het probleem met computers is dat ze de menselijke taal niet spreken. Een programmeertaal zoals “C” komt al dichterbij, maar ook die taal begrijpt de microcontroller helemaal niet. We hebben daarom ook een tolk vertaler nodig.

Dit vertalen kan op twee manieren gedaan worden.

INTERPRETER – Vertalen tijdens gebruik

Het vertalen terwijl we het programma “draaien” of aflopen wordt gedaan door een zogenaamde **interpreter**.

Zie het als het lezen van een boek geschreven in een taal die jij niet kent. Om dat boek te kunnen lezen, moet er een vertaler naast je zitten. Die eerst wat tekst leest en deze dan vertaalt en hardop uitsprekt, zodat jij het verhaal kunt volgen.

Het voordeel is dat we meteen aan de slag kunnen met het lezen van dat boek in die vreemde taal. Het nadeel is dat we dus wel even een tolk in huis moeten hebben, die dan ook nog steeds een stukje moet lezen, vertalen en hardop uitspreken ... dus zeg maar regel voor regel.

Dit is dus langzamer dan wanneer het hele boek al in het Nederlands zou zijn.

Dus snelle start, maar langzaam lezen.

COMPILER – Vertalen voor gebruik

Hierbij komt de zogenaamde **compiler** om de hoek kijken.

Een compiler kun je zien als een vertaler die het hele boek gelezen en vertaald heeft.

Je hebt dus een Nederlandse versie van het boek in handen.

Het voordeel is dus dat we geen tolk in de voorraadkast hoeven te hebben en dat we meteen kunnen lezen als we dat zouden willen zonder dat er iets vertaald hoeft te worden.

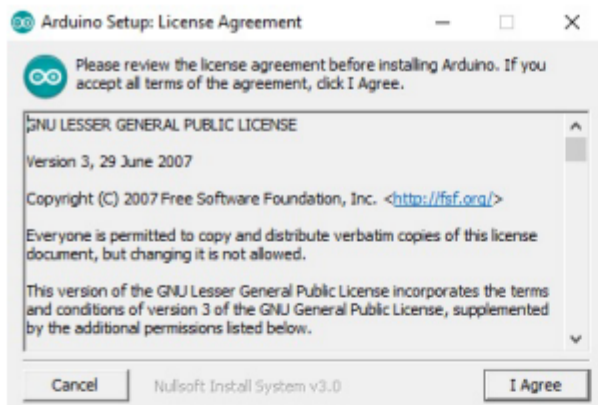
Het nadeel is echter dat het boek eerst vertaald dient te worden, in z’n geheel, vóór dat we het kunnen gaan lezen. Dus langzame start, maar snel lezen.

Dit geldt ook voor een computer. Het vertalen van het hele programma voor we het starten, kost wat extra tijd, maar eenmaal vertaald hebben we geen hulp meer nodig. Daarbij komt dan nog eens dat het “lezen” sneller gaat, wat zich laat zien door een programma die veel sneller is

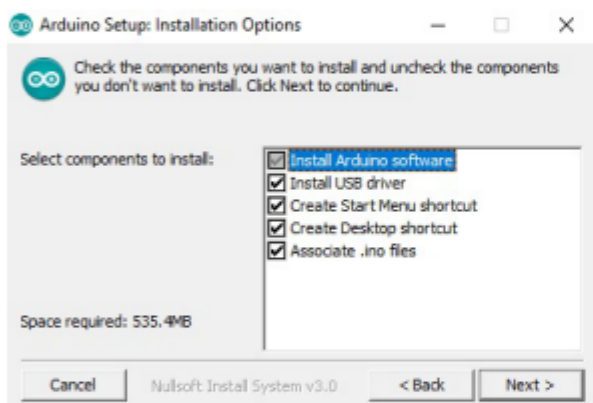
De Arduino IDE gebruikt dus een COMPILER

HOE INSTALLEREN WE ARDUINO IDE OP JE COMPUTER ?

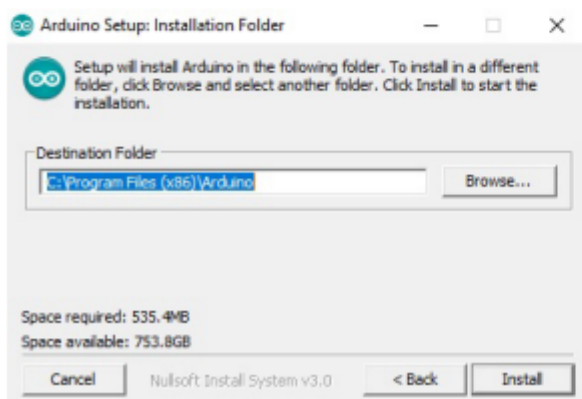
Na het downloaden van de Arduino IDE op jouw computer is het tijd om te installeren. Je doet dit door dubbel te klikken met de linkermuisknop (LMK) op het EXE bestand. (De bestandsextentie "EXE" staat voor "executable" en betekent "uitvoerbaar".) Vervolgens volg je de instructies op je scherm.



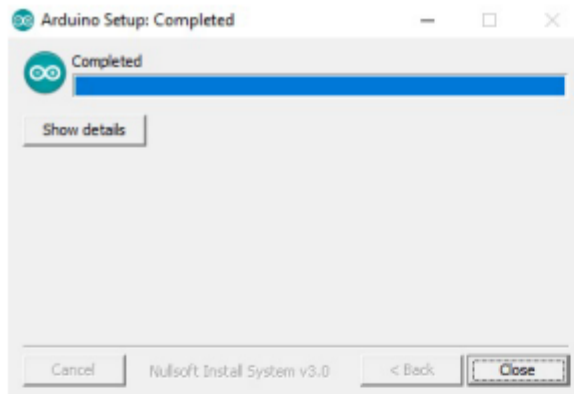
Klik (LMK) op "I Agree"



Klik (LMK) op "Next >"

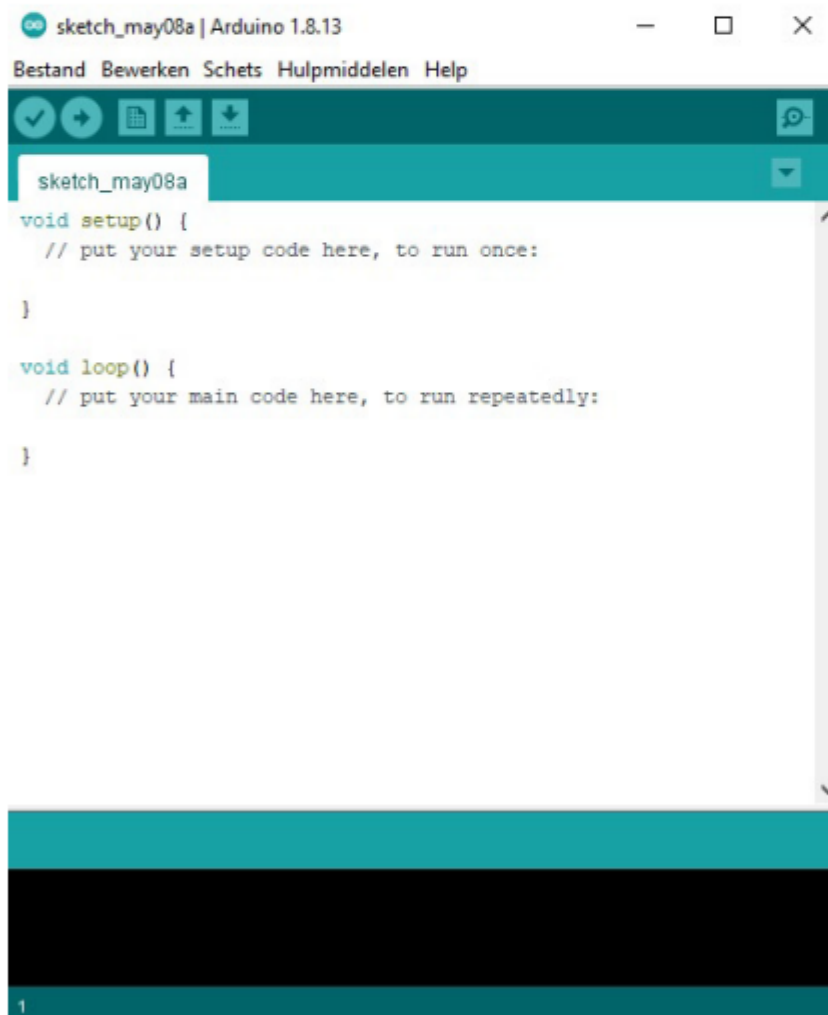


Klik (LMK) op "Install"



Klik (LMK) op "Close"

Eenmaal geïnstalleerd en opgestart ziet het beeld van je Arduino IDE er als volgt uit :

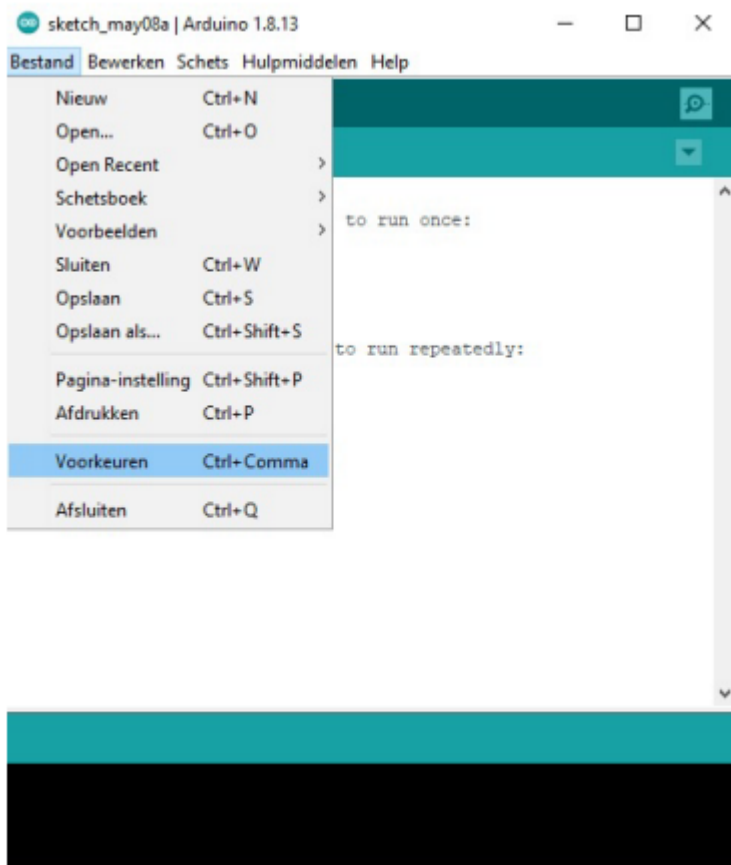


INITIËLE CONFIGURATIE VAN DE ARDUINO SOFTWARE

Na installatie moeten we even kijken naar enkele initiële instellingen zodat onze ESP32 correct gekozen wordt.

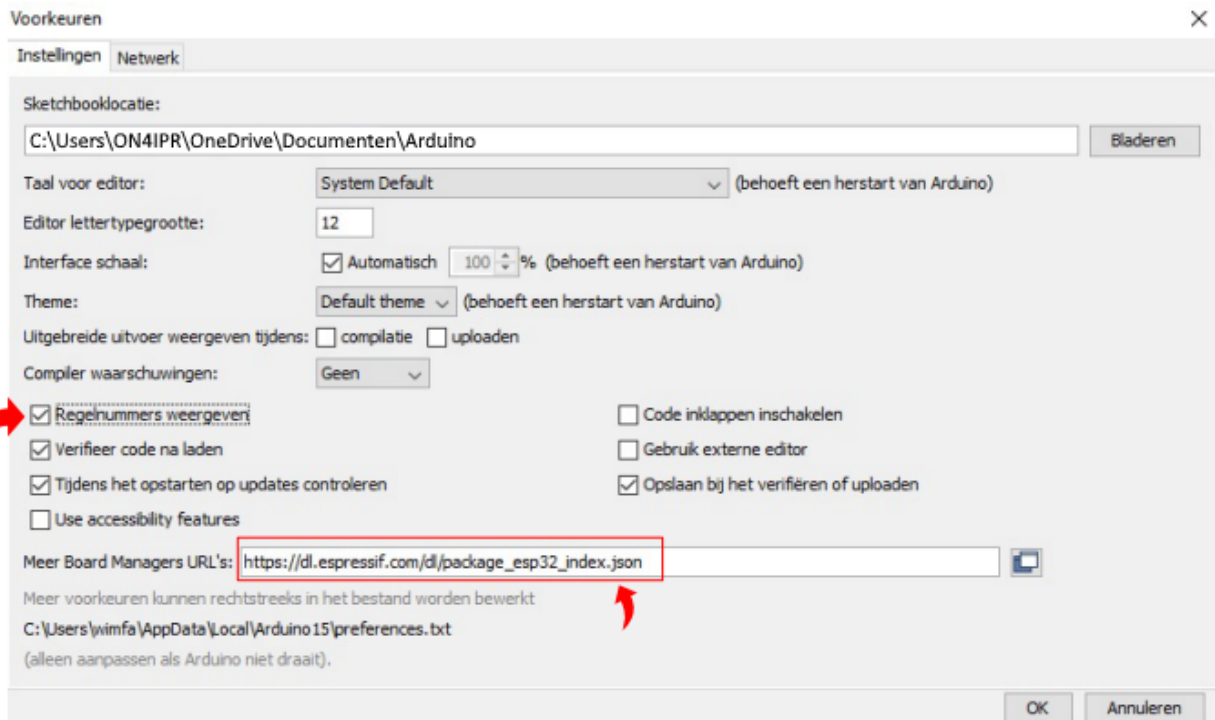


Sluit de ESP32 device aan op de computer met het USB kabeltje.

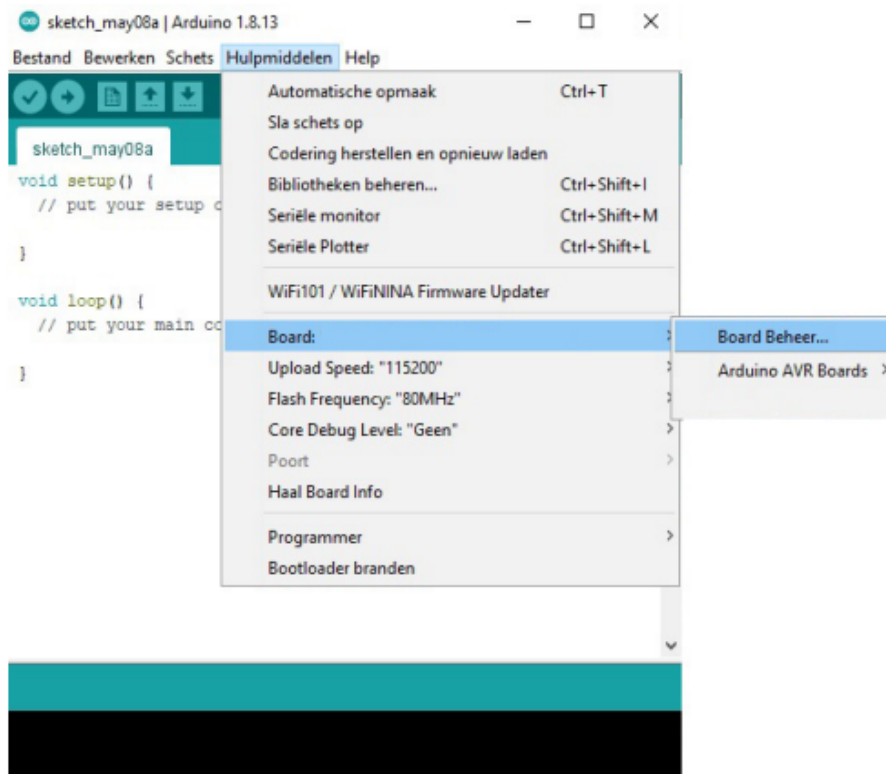


In het menu bovenaan selecteren we “Bestand” en klikken (LMK) op “Voorkeuren”

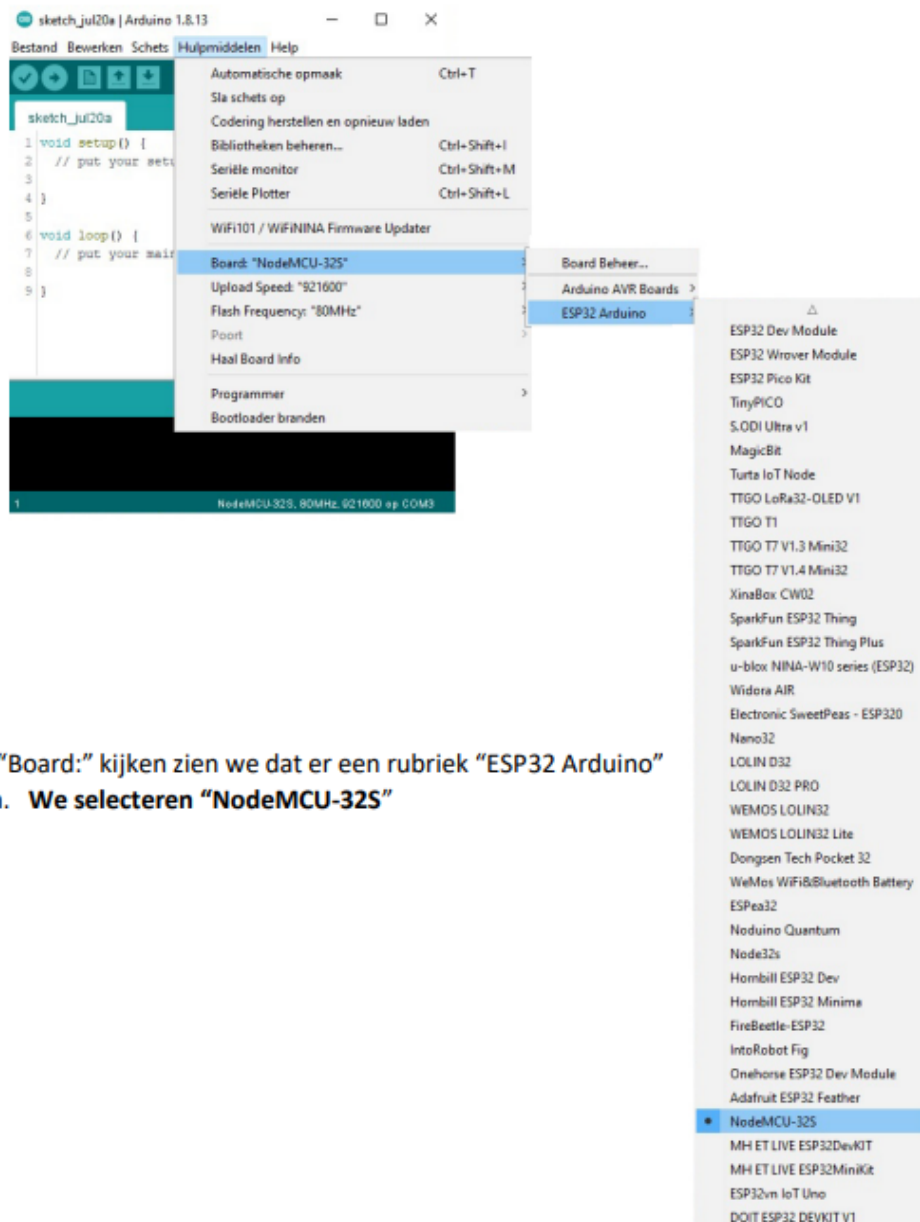
We vervolledigen het scherm met de voorkeuren :



In het menu
bovenaan
selecteren we
"Hulpmiddelen"
en klikken (LMK)
op "Board"
en dan op
"Board Beheer..."

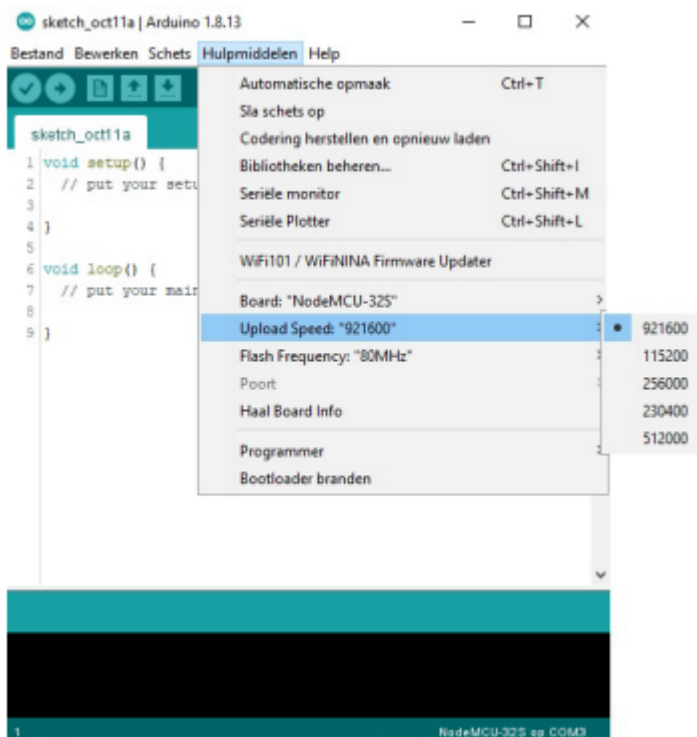


Vervolgens typen we ESP32 in, druk op “enter” en daarna klikken (LMK) op “installeren”

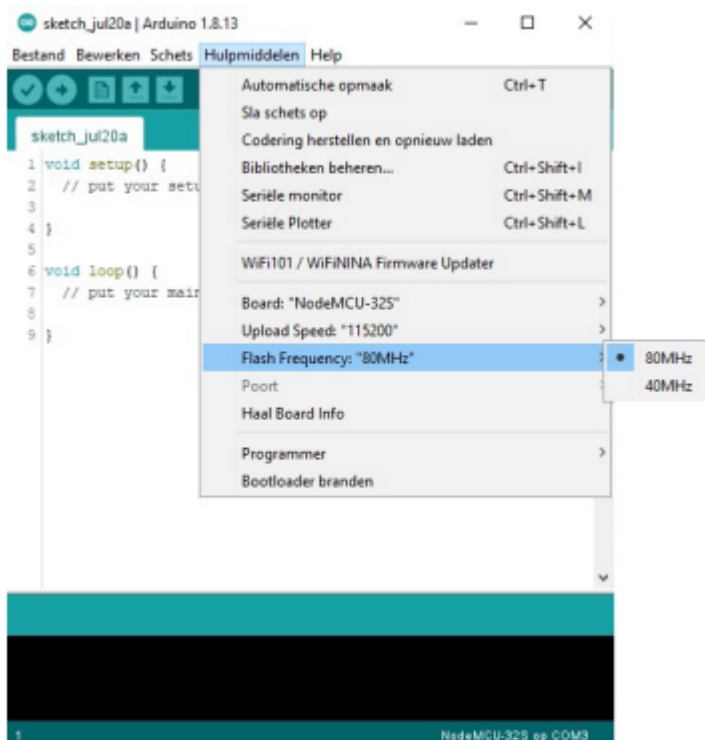


Als we nu bij “Board:” kijken zien we dat er een rubriek “ESP32 Arduino” is bijgekomen. **We selecteren “NodeMCU-32S”**

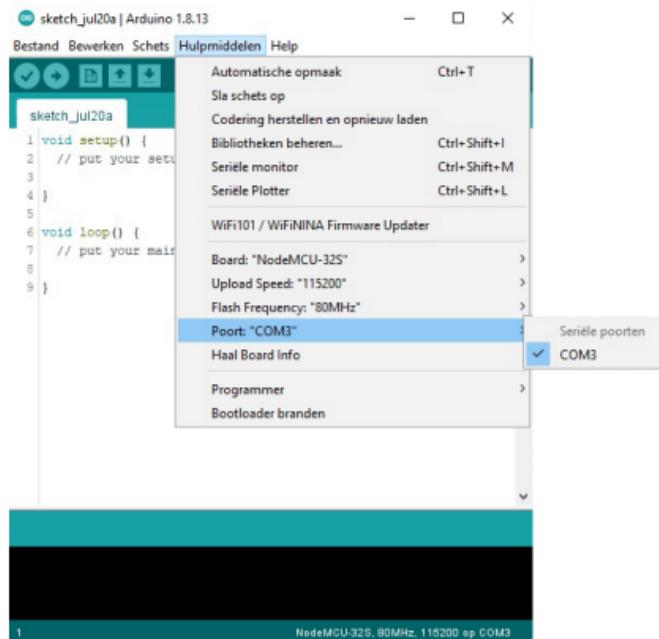
Vervolgens selecteren we voor **“Upload Speed”** : **921600**



Voor **“Flash Frequency”** selecteer je : **80 MHz**

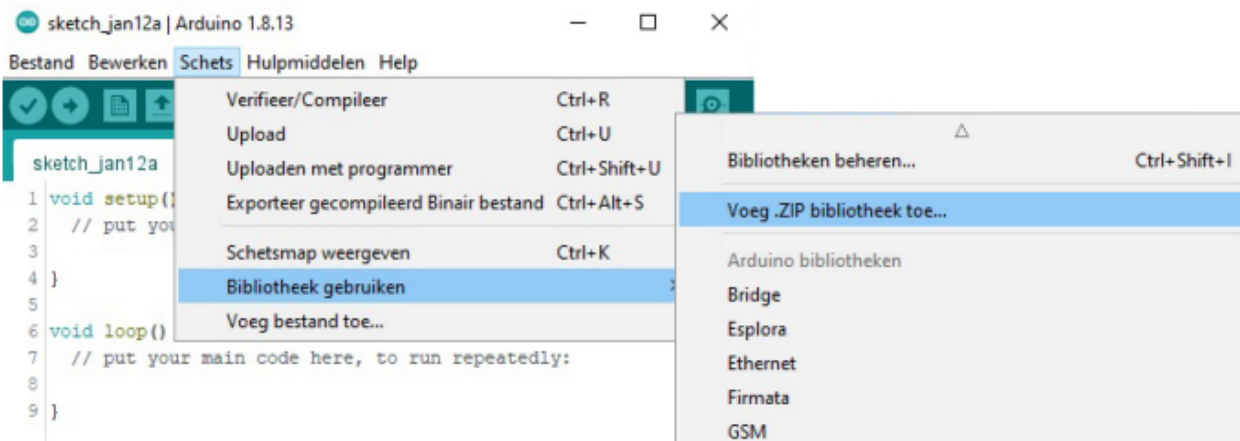


Tenslotte moeten we nog de COM poort selecteren die onze ESP32 zal gebruiken. Bij mij is dit COM3, bij jou computer kan dit ook een ander COM poort zijn.



INSTALLEREN VAN DE BIBLIOTHEEK “E4SBoard”

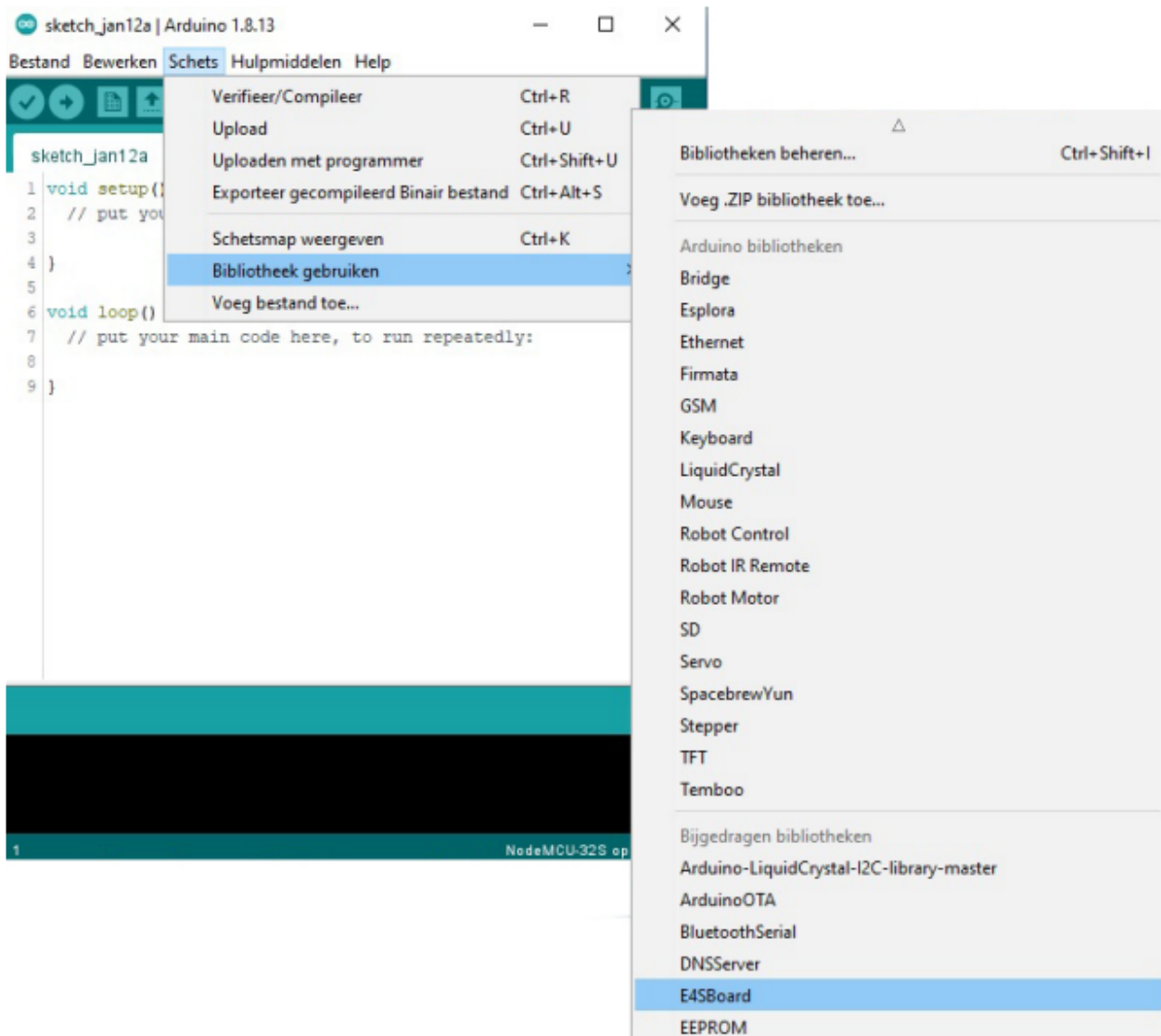
Kopieer het bestand “**E4SBoard.zip**” naar het bureaublad van je computer.
In het menu bovenaan selecteren we “Schets” en klikken (LMK) op “Bibliotheek gebruiken” daarna selecteren we “Voeg .ZIP bibliotheek toe...”



Selecteer het bestand “**E4SBoard.zip**” die op het bureaublad van je computer staat.

Klik (LMK) op “**Open**”

Als de bibliotheek “**E4SBoard**” goed geïnstalleerd is dan moet ze **in de lijst** van “**Bijgedragen bibliotheken**” staan.



TESTEN VAN ONZE ESP32 DEVICE

Nu dat we de Arduino IDE hebben geconfigureerd (alle instellingen zijn in orde gemaakt) ... is het tijd om te kijken of het werkt. We gaan alles even testen met een klein programma dat alle functies van onze ESP32 device achtereenvolgens laat werken. Dit programma vinden we terug in de map “ESP32_TestProg”. We downloaden deze map en vinden hem terug als een zip-file in de map “Downloads” op onze computer. Vervolgens verplaatsen we deze map naar de desktop van onze computer. We klikken met onze rechtermuisknop op deze map en selecteren “Alles uitpakken ...”. We kijken in de uitgepakte map en klikken op de onderliggende mappen tot we de file “ESP32_BB_test02” zien. We doen dubbelklik op deze file en openen hem in onze IDE.

De volgende stap is dat we dit programma moeten vertalen (dit doet de compiler en wordt ook wel “compileren” genoemd), en vervolgens moet het vertaalde resultaat naar de ESP32 gestuurd worden. Gelukkig kan de Arduino IDE dit allemaal voor ons regelen met een enkele klik. Zie je de 5 knoppen, boven in de Arduino IDE, zoals weergegeven in onderstaande figuur? We klikken nu op de “B” knop. (Het pijltje die naar rechts wijst.)



In de bovenstaande tabel zie je dat knop “A” gebruikt kan worden om jouw code te verifiëren op fouten.

Knop “B” doet dit ook, maar vertaalt het daarna nog eens (compileren) en als er geen fouten gevonden werden, stuurt hij de code naar onze ESP32.

De knoppen “C”, “D” en “E” worden gebruikt om met bestanden te werken, zoals het maken van een nieuw document, opslaan (Save) en inlezen (Open).

Je ziet dat we het woord “Sketch” een paar keer hebben gebruikt.

Dit woord gebruiken Arduino gebruikers om **source code (broncode)** aan te geven.

Met andere woorden : ons programma geschreven in de **programmeertaal C**.

Nadat je op de “B” knop hebt geklikt, zal jouw programma (source code) gecontroleerd worden, vertaald worden en naar de ESP32 gestuurd worden (uploaden) , mits alles goed ging natuurlijk.

Tijdens dit gebeuren zie je onderaan op de Arduino IDE tekst verschijnen, die aangeeft waar de Arduino IDE mee bezig is.

Vervolgens start het programma : 1 seconde aan, 1 seconde uit, 1 seconde aan, 1 seconde uit enz...

Gefeliciteerd ! Je hebt een ESP32 aangesloten op jouw computer, de Arduino IDE geïnstalleerd en een eerste test programma naar de ESP32 gestuurd.



Klaar voor Module 2 ...

Als je vragen hebt, blijf er niet mee zitten.
Bedenk dat er geen domme vragen zijn,
behalve dan natuurlijk de vraag die niet gesteld is.
We zijn allemaal ooit bij nul begonnen !

Every
pro
was
once an
amateur,
every
expert
was once
a
beginner.

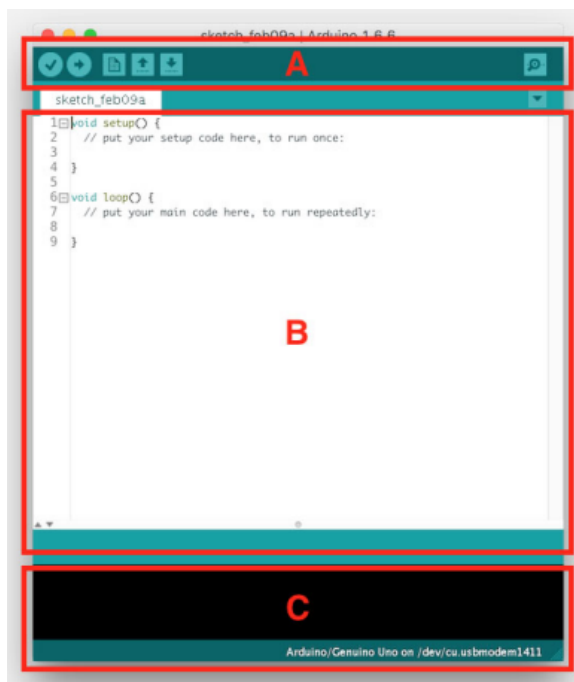
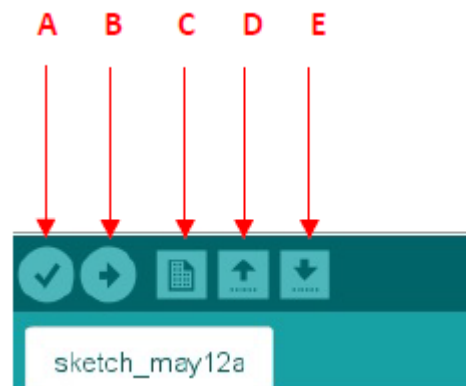
Arduino programmeren - Uitvoer - [Module 2]

EEN KORTE VERKENNING VAN DE ARDUINO IDE

In Module 1 leerden we reeds een aantal knoppen kennen van onze Arduino IDE.

Klik op de knop “C” zoals in onderstaande afbeelding wordt weergegeven om een nieuw “project” starten.

Na het klikken op de knop “C” zie je een venster zoals hieronder :



We zien 3 “belangrijke zones” :

In zone “A” zien we de knoppen, die we eerder besproken hebben.

Helemaal rechts zien we een extra knop

die op een vergrootglas lijkt



Dit is de “**Serial Monitor**” knop, die soms ook wel eens “Debug Window” knop genoemd wordt.

In zone “B” kunnen we onze Source Code (Broncode) bewerken. Het is eigenlijk een simpele tekstverwerker met zogenaamde “Code highlighting”. Code highlighting zorgt ervoor dat bepaalde trefwoorden, tekens of karakter reeksen een bepaalde kleur krijgen, wat ons beter helpt bij het lezen van onze broncode.

Zone “C” is het gebied waar de Arduino IDE ons belangrijke meldingen zal geven. Bijvoorbeeld als er iets fout gaat als we onze broncode compileren, of als er iets anders fout of goed gaat. In het begin zullen deze meldingen nog wat onduidelijk zijn voor ons, maar naarmate we meer leren en ervaring opdoen worden ze gemakkelijk om te volgen.

De Arduino IDE heeft natuurlijk nog veel meer functies, die we zelden gebruiken maar toch handig zijn. Naarmate we meer werken met Arduino zul je merken wat een aantal functies voor ons kunnen doen. Nu zijn ze niet zo van belang.

COMMUNICATIE MET DE ESP32

Zoals eerder gezegd en gezien: de ESP32 heeft geen beeldscherm, toetsenbord of muis. Dat maakt communicatie tussen ons mensen en de ESP32 wat lastig en aan een knipperend lampje hebben we niet al teveel. Dus hoe kunnen we zien (uitvoer!) wat de ESP32 aan het uitspoken is ? Daarvoor gaan we onze "Serial Monitor" gaan gebruiken. Later hierover meer.

WAT HEEFT USB TE BIEDEN BIJ ARDUINO PROGRAMMEREN ?

1. USB (Universal Serial Bus) kan de ESP32 voorzien van stroom.

Zoals met de meeste elektronica heeft de ESP32 natuurlijk stroom nodig om te werken. Onder normale omstandigheden biedt een USB-aansluiting 5V, wat genoeg is voor de ESP32 om te werken.

Wij hebben ervoor gekozen om onze ESP32 niet te voeden via USB maar via een externe adapter omwille van twee redenen :

- Dit maakt het mogelijk om de ESP32 programma's te laten uitvoeren zonder dat hij verbonden is met de computer.
- Uit veiligheidsoverwegingen. Stel dat je tijdens het verbinden van draden de voedingsspanning van de ESP32 zou kortsluiten, dan is er geen gevaar dat de USB-poort van je computer stuk zou gaan.

2. USB als middel om programma's naar de ESP32 te sturen.

We schrijven ons programma op een computer en dat programma (na vertaling) moet dus naar de ESP32 overgezet worden. Het zogenaamde "**UPLOADEN**".

Sommige USB-kabeltjes zijn alleen geschikt om een toestel van voeding te voorzien...

We moeten er dus op letten, dat ons USB-kabeltje ook geschikt is om data over te brengen ! Het uploaden gebeurt dus over de USB verbinding.

Later zullen we leren hoe we dit "over the air" (draadloos) kunnen doen.

3. USB kan data ontvangen van de ESP32.

De USB-verbinding kan ook gebruikt worden om gegevens (data) van de ESP32 naar de PC terug te sturen. Dan kunnen we deze data op het beeldscherm van onze computer bekijken.

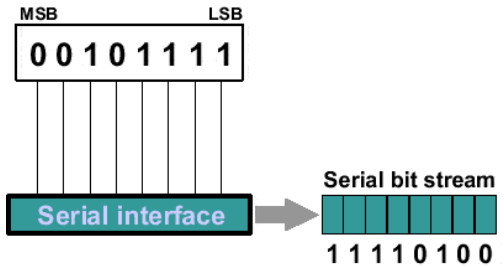
SERIËLE COMMUNICATIE OVER USB



Zoals je ziet, is de belangrijkste taak van de USB-verbinding communicatie. Nu moet je weten dat een microcontroller een tijdje geleden nog communiceerde via een zogenaamde seriële poort. Dat doet ze nog altijd, maar men heeft een "omzetter" op de meeste arduino modellen geplaatst die dit omzet naar een USB-aansluiting. De seriële poort zoals we ze vroeger kenden is dus eigenlijk geëvolueerd naar een USB-poort, die de data aan veel hogere snelheden kan overbrengen. Bij onze ESP32 vinden we dus ook dergelijke USB-aansluiting. Een dergelijke seriële poort wordt ook wel "COM poort" genoemd.



Serial interfaces—one bit at a time



Een seriële poort, of COM-poort verstuurd gegevens (data) met één bit per keer, achter elkaar dus.
Een bit kan de waarde van een “één” (1) of een “nul” (0) aannemen.

Aan één enkele bit hebben we niet zo veel ... Het kan alleen maar twee toestanden aangeven.
vb. aan/uit, ja/nee, waar/niet-waar (dit noemen we “boolean”, iets wat we later bespreken)
Om dit op te lossen heeft men de “**byte**” bedacht.

Een byte bestaat uit 8 bits in een groep, die 256 verschillende waarden kunnen aannemen.

Waarom nu juist 256 waarden hoor ik je vragen ?

Eén bit kan twee waarden aannemen, een nul (0) of een één (1).

Met 8 bits kunnen we wiskundig gezegd 2^8 (= 2 tot de macht 8 = $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 256$) combinaties maken.

Mocht je in de wiskundig nog niet over machten geleerd hebben, geen probleem ...

Dit komt zeker nog aan bod in volgende lessen.

Nu is het belangrijk, dat je weet dat een **byte 256 waarden** kan hebben : **van 0 tot 255**.

Een seriële poort “praat” met een bepaalde snelheid, dit noemt men de “**baudrate**”.

De baudrate geeft aan hoe snel bits verstuurd en/of ontvangen worden. We spreken van

“**bits per seconde**”. Er zijn een aantal standaard snelheden zoals bijvoorbeeld **9600**.

Dat wil zeggen dat gegevens (data) met een snelheid van 9600 bits per seconde verstuurd worden.

Natuurlijk moet men de snelheid aan beide kanten hetzelfde hebben, dus zowel de computer als de ESP32 moeten met dezelfde snelheid praten anders snappen ze elkaar niet.

Een aantal standaard “baudrate”waarden zijn :

110, 300, 600, 1200, 2400, 4800, **9600**, 14400, 19200, 38400, 56000, 115200

OK, even genoeg geschiedenisles voor vandaag ... We weten nu dat via de USB-verbinding ons programma naar de ESP32 kan gestuurd worden. Tijd om eens te kijken hoe de ESP32 nu gegevens terug kan sturen naar de computer.

SERIËLE MONITOR – GEGEVENS VAN DE ESP32 ONTVANGEN

Als we een programma starten, dan willen we natuurlijk graag zien wat het programma doet. Al is het maar om eventuele fouten te zoeken in ons programma.

Het zoeken van fouten noemen we “**DEBUGGING**”

Debugging is het proces waarbij men fouten (“bugs”) probeert te vinden en probeert te verhelpen in programma’s.

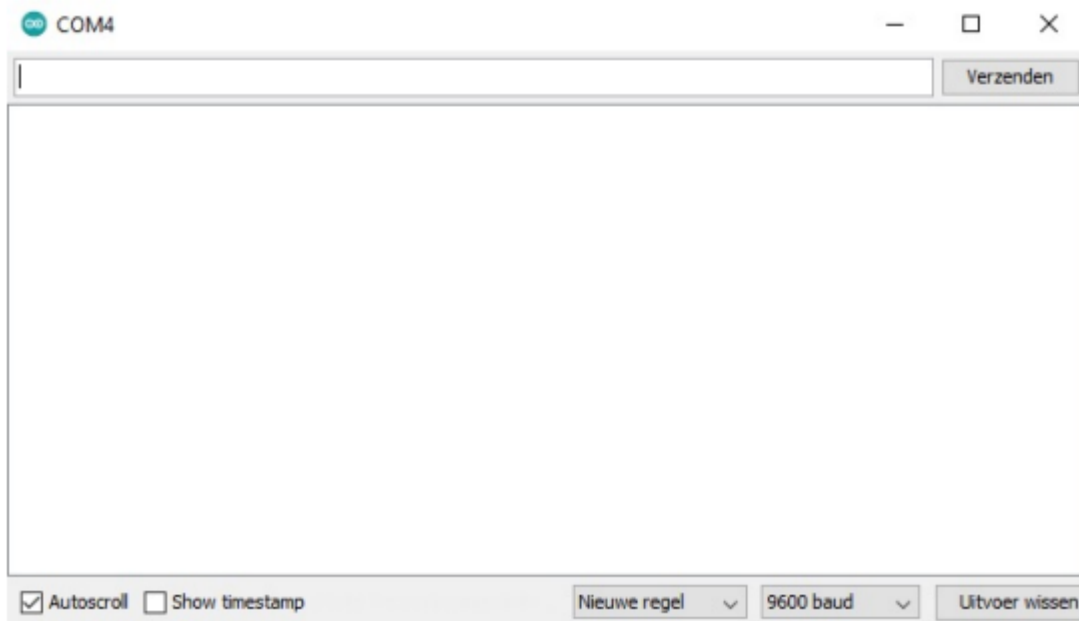
Dit is waar de “**Serial Monitor**” erg handig voor is ...

Om er nu voor te zorgen dat de ESP32 ons informatie stuurt, moeten we zowel de Seriële Monitor als de ESP32 vertellen met welk tempo we gaan communiceren. Voor de ESP32 wil dat zeggen, dat we in ons programma moeten opgeven welke baudrate we willen gaan gebruiken. Meestal is de Arduino IDE slim genoeg om dat vanzelf te ontdekken, maar soms moeten we dit handmatig doorgeven.

Laten we onze seriële monitor eens gebruiken.

Hiervoor moeten we de Arduino IDE open hebben en de ESP32 moet verbonden zijn met de computer. Als dit in orde is, kunnen we drukken op de knop “Serial monitor”. (het vergrootglas-knopje rechtsboven in de Arduino IDE)

Nadat je op de “Serial Monitor” knop hebt gedrukt, krijg je als het goed is een venster op je computer te zien zoals op onderstaande afbeelding :



In dit venster kunnen we gegevens (data) ontvangen van de ESP32, maar ook gegevens naar de ESP32 toe sturen. (later meer hierover) Laten we onze eerste “praat” programma bekijken ...

In onderstaande code heb ik 3 regels aan de standaard code toegevoegd : regels 3 t.e.m. 5.

```
void setup()

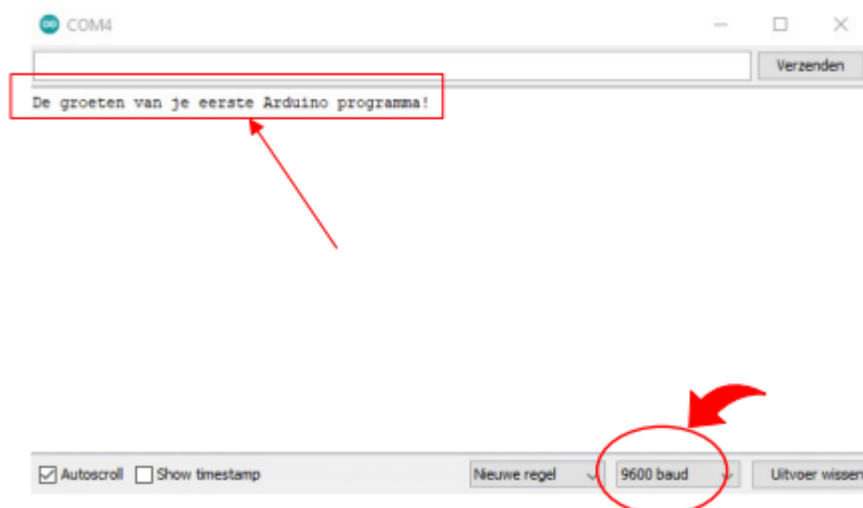
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(500);
  Serial.println("De groeten van je eerste Arduino programma!");
}

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

De regel “**Serial.begin**(9600);” vertelt de ESP32, dat het over de seriële verbinding moet praten met een snelheid van 9600 baud. Als je nu in de onderstaande afbeelding kijkt, dan zie je “9600 baud” rechts onderin. (Dezelfde snelheid die we opgeven in het programma.)

Met de regel “**delay**(500);” laten we het programma 500 milliseconden (= 0,5 seconden) wachten.

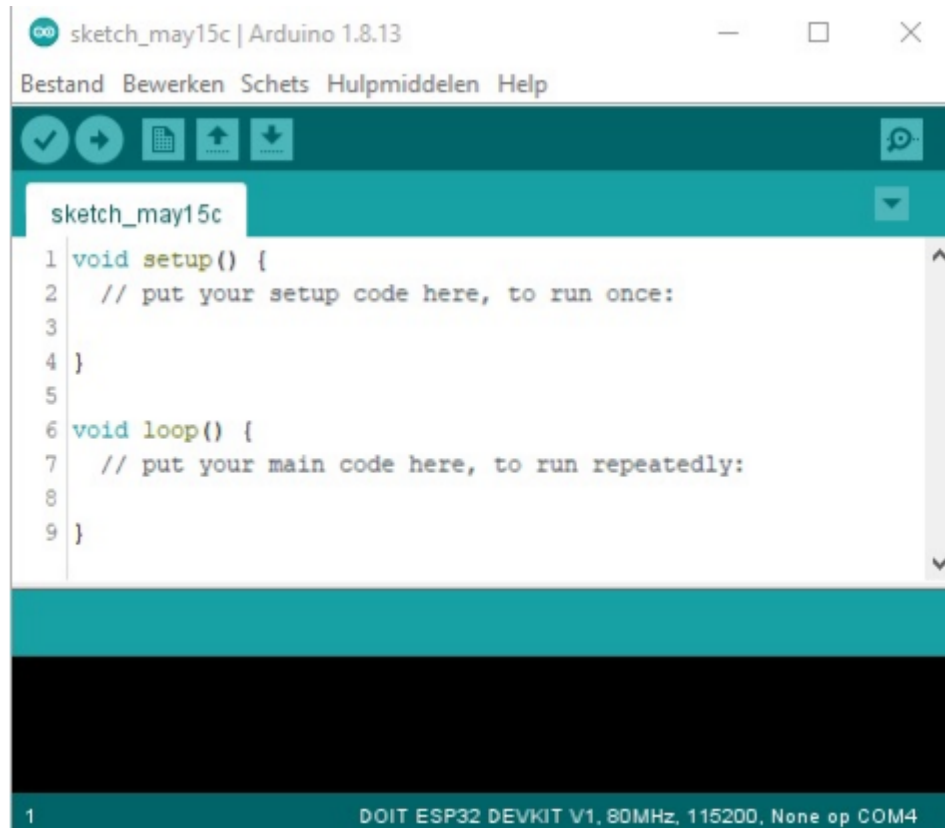
De daarop volgende regel “**Serial.println**(“De groeten van je eerste Arduino programma!”);” vertelt de ESP32 dat het de tekst moet versturen over de seriële verbinding. OK, neem nu de code over in de Arduino IDE, waarbij alle bestaande tekst vervangen wordt. Klik vervolgens op de “**Compileer en Upload**” knop. (knop **B**, zie afbeelding pagina 20 bovenaan) Na het uploaden zal je het volgende zien in het venster van de seriële monitor :



DE BASIS ARDUINO PROGRAMMA LAY-OUT

Setup() en Loop()

Ieder Arduino programma heeft dezelfde basis lay-out en dat zie je meteen als je een nieuw project start. (klik op de “C” knop, zie afbeelding pagina bovenaan [Module 2])



Hier zien we twee basiselementen “**setup()**” en “**loop()**”, en beiden, ook al laat je ze leeg, moeten bestaan ! Als de ESP32 start, dan zal het eerst naar “**setup()**” gaan zoeken en deze slechts één keer doorlopen. Daarna zal de ESP32 “**loop()**” uitvoeren en deze “**loop()**” oneindig blijven herhalen tot de ESP32 uitgezet wordt.

Vandaar de naam “loop” (= lus).

Om dit te illustreren zouden we de code die de tekst uitvoert naar de seriële verbinding, in de “**loop()**” kunnen zetten i.p.v. in de “**setup()**”. Het gevolg zou zijn, dat het venster van de seriële monitor herhaaldelijk dezelfde tekst blijft weergeven tot we de ESP32 uitschakelen.

Probeer het maar eens :

We laten "**Serial.begin(9600);**" in de "setup()" staan, want instellen van de snelheid hoeft natuurlijk maar 1 keer.

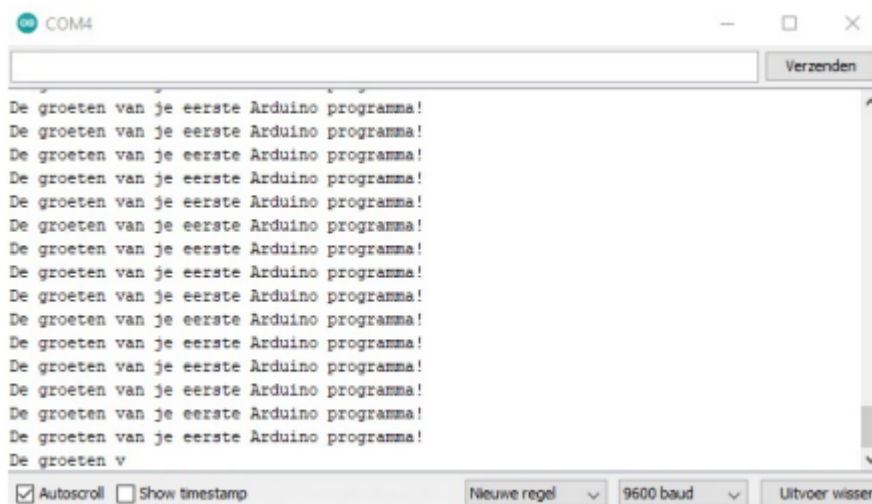
De regel "**Serial.println("De groeten van je eerste Arduino programma!");**" verplaatsen we nu naar "loop()"

wat dus als gevolg heeft dat de tekst zich blijft herhalen.

```
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(9600);
}

void loop()
{
    // put your main code here, to run repeatedly:
    delay(500);
    Serial.println("De groeten van je eerste Arduino programma!");
}
```

Het resultaat op de seriële monitor ziet er als volgt uit:



ESP32 PROGRAMMEREN MET OPMERKINGEN

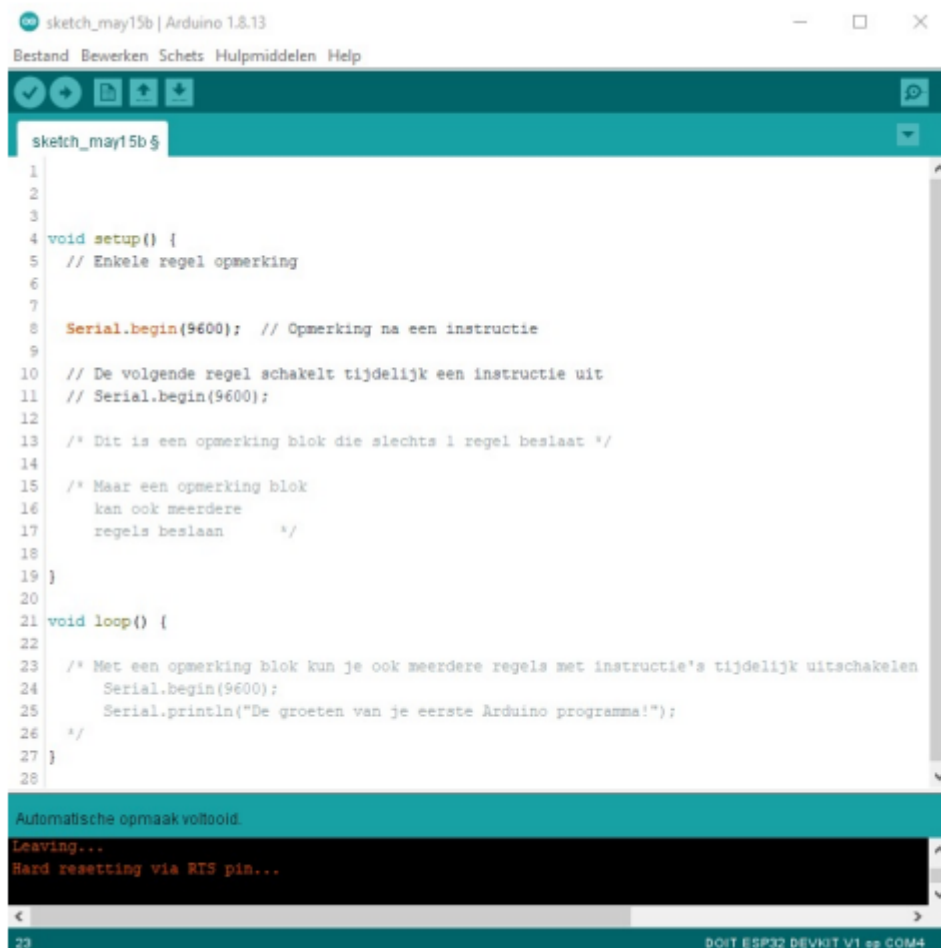
Het toevoegen van opmerkingen in de code van ons programma is zeker sterk aan te raden !

Waarom ?

- Het maakt onze code aanzienlijk duidelijker, wat handig is als we op een later tijdstip iets willen veranderen in onze code.
- Nog belangrijker is dat opmerkingen uitleg bieden aan anderen die misschien jouw code willen bewerken of aanpassen.

We kunnen dit op twee manieren doen :

Je moet weten dat opmerkingen beginnen met **2 schuine strepen "//"**, dat wil zeggen dat alles achter deze tekens (in deze regel) als commentaar of opmerking moet gezien worden. Dat wil dus ook zeggen, dat de compiler alles na de 2 schuine strepen negeert op de betreffende regel. Dat wil ook zeggen dat we opmerkingen kunnen plaatsen achter commando's. Opmerkingen kunnen ook tussen **"/*"** en **"*/"** geplaatst worden en dat mag zelfs over meerdere regels.



```
1
2
3
4 void setup() {
5   // Enkele regel opmerking
6
7
8   Serial.begin(9600); // Opmerking na een instructie
9
10  // De volgende regel schakelt tijdelijk een instructie uit
11  // Serial.begin(9600);
12
13  /* Dit is een opmerking blok die slechts 1 regel beslaat */
14
15  /* Maar een opmerking blok
16     kan ook meerdere
17     regels beslaan */
18
19 }
20
21 void loop() {
22
23  /* Met een opmerking blok kun je ook meerdere regels met instructie's tijdelijk uitschakelen
24     Serial.begin(9600);
25     Serial.println("De groeten van je eerste Arduino programma!");
26  */
27 }
28
```

Automatische opmaak voltooid.
Leaving...
Hard resetting via RTS pin...

23 DOIT ESP32 DEVKIT V1 op COM4

INSTRUCTIES EN CODE BLOKKEN

Een code blok start en eindigt met accolades “{}”

De accolade open, start de code blok. Alles wat hierna komt, tot het punt waar we de accolade sluiten, wordt als één blok gezien.

Deze accolades zien we altijd na een functie vb. “void setup() {}”

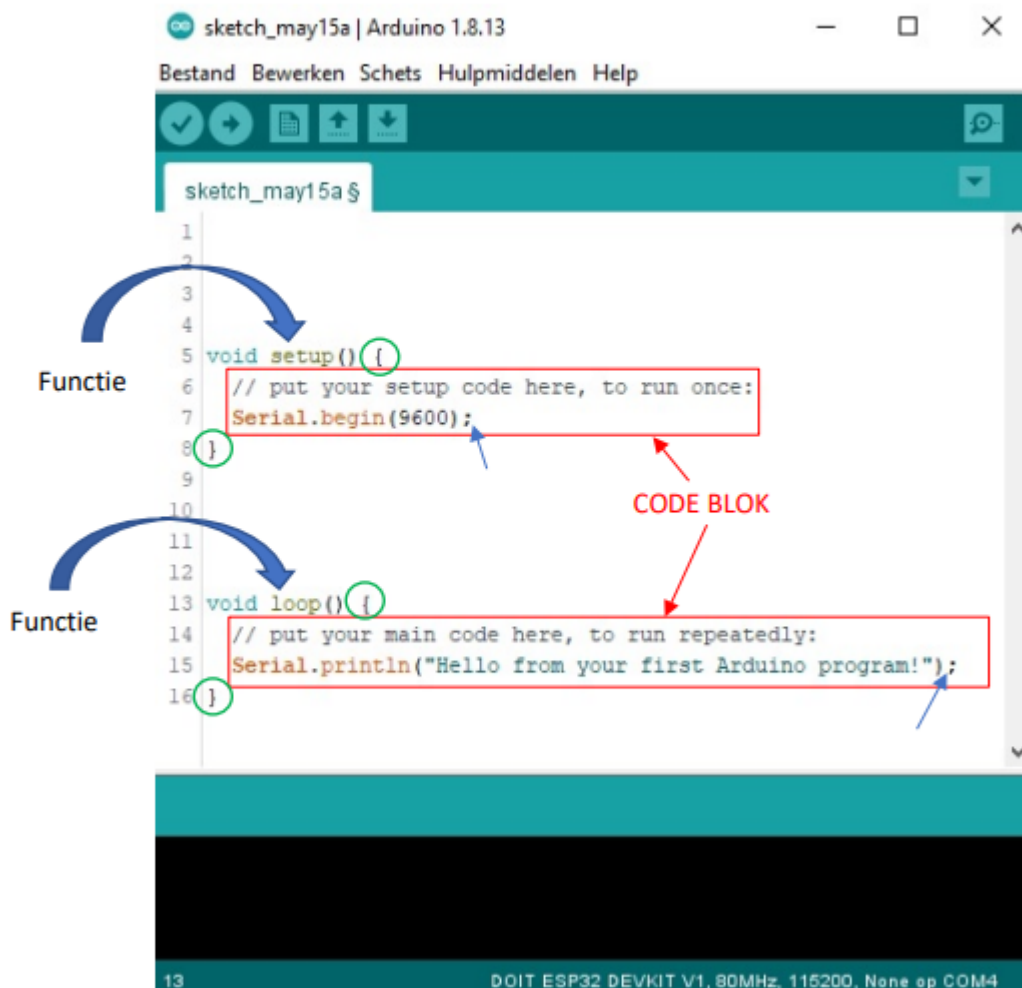
Hier is de naam van de functie “setup”.

Je vraagt je wellicht af wat “void” en de dubbele haakjes () betekenen, daarvoor is het nu nog te vroeg en daar komen we later nog op terug.

Een instructie eindigt altijd met een puntkomma “;”

Voorbeeld :

`Serial.begin(9600);`



Voor je verder gaat naar Module 3 is het een goed idee om “Verdieping 1” door te nemen.

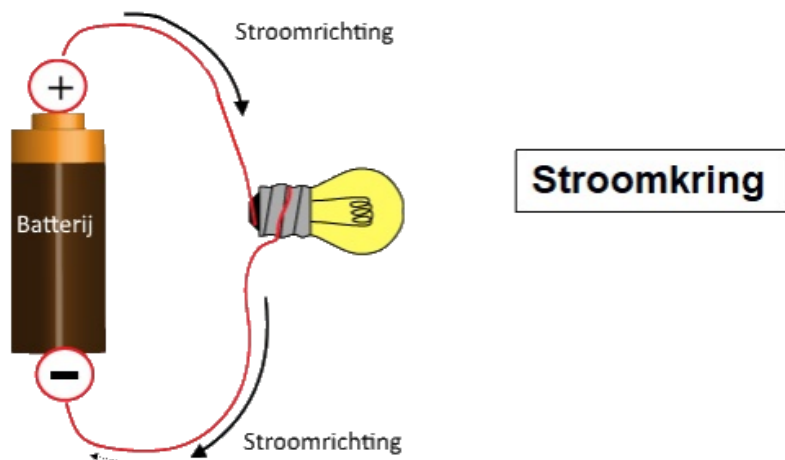
Stroomkring - [Verdieping 1]

Voor we verder gaan met programmeren zullen we eerste enkele basisbegrippen uit de elektronica van naderbij bekijken.

WAT IS EEN STROOMKRING ?

Een stroomkring bestaat uit een aantal onderdelen, die er samen voor zorgen dat er een elektrische stroom kan vloeien. Bijvoorbeeld een batterij (bron), een lamp (verbruiker) en enkele geleiders.

De batterij levert **STROOM**, die via de geleiders van de pluspool van de batterij door de lamp naar de minpool van de batterij loopt. Hierdoor gaat het lampje licht en warmte afgeven. We spreken hier in dit geval van een gesloten stroomkring

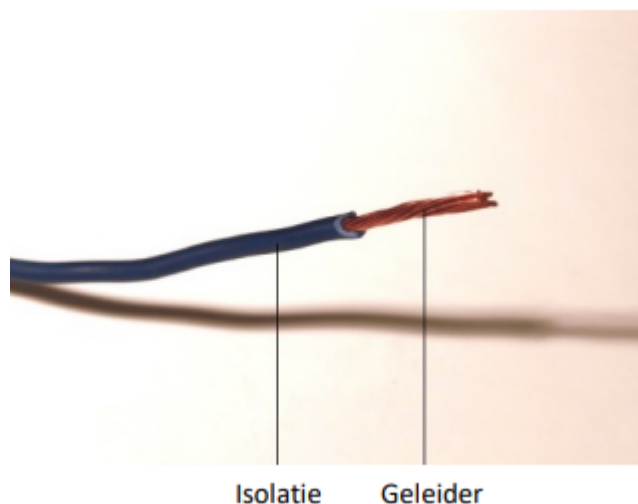


De draden (**geleiders**) waarmee de lamp verbonden is aan de batterij zijn van een materiaal gemaakt waar de stroom gemakkelijk doorheen kan.

Vb. Koperdraad

Materialen waardoor de stroom niet kan, noemen we **isolatoren**.

Vb. PVC, rubber, glas enz ...



Je kunt elektrische stroom goed vergelijken met het stromen van water door een buis.



De waterpomp lijkt op de batterij in onze elektrische stroomkring. De waterpomp zorgt voor een zekere druk op het water zodat het gaat stromen in onze waterleiding. Op dezelfde manier zorgt de batterij ervoor, dat er stroom vloeit in onze elektrische stroomkring. De batterij zorgt als het ware voor een “druk” die ervoor zorgt, dat er stroom kan vloeien in onze elektrische stroomkring. We zullen

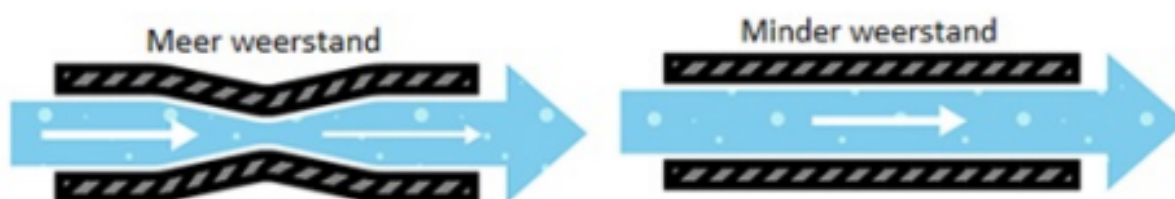
deze “druk” vanaf nu **“SPANNING”** noemen.

De spanning van de batterij zorgt ervoor dat er elektrische stroom kan vloeien in onze stroomkring.

We hebben reeds gezegd, dat de elektrische stroom gemakkelijk door koperen geleiders kan vloeien. Er bestaan ook materialen waar de elektrische stroom minder goed door kan.

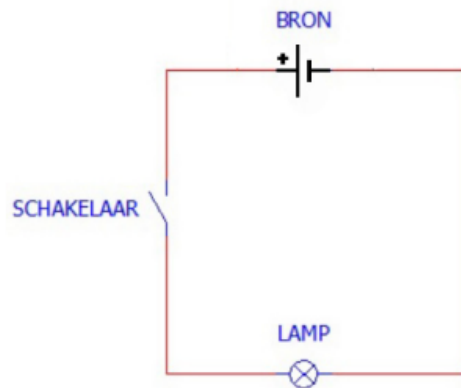
Een component dat tegenstand biedt aan de elektrische stroom noemen we een **“WEERSTAND”**

Als we dit vergelijken met onze waterleiding, dan zal het water tegenstand ondervinden door een vernauwing in de buis



We kunnen alle componenten die we gebruiken ook tekenen in een stroomkringschema :

Met de schakelaar kunnen we de stroomkring onderbreken.



De bron (batterij) levert een gelijkspanning (DC) die je kan meten met een "voltmeter".

We plaatsen de selector van de V-meter op de stand "DC" :



We verbinden de "COM"-klem (zwart) van de V-meter met de "-" klem van de bron.

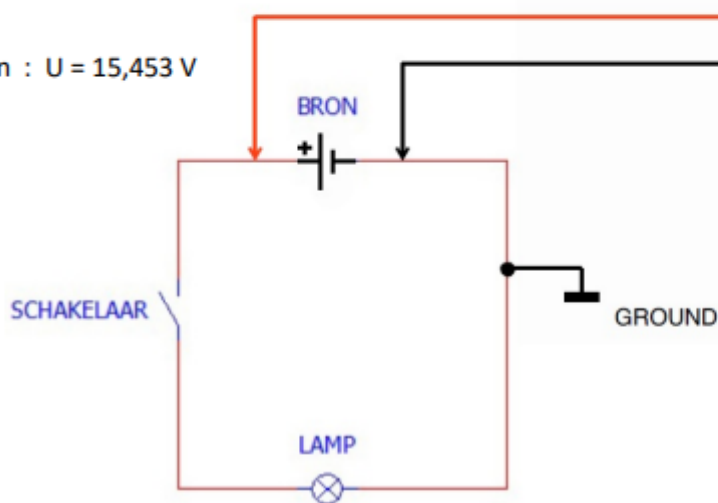
We verbinden de "V"-klem (rood) met de "+" klem van de bron.



De spanning krijgt als symbool de letter "U"

De spanning wordt uitgedrukt in Volt (V).

De spanning van deze bron : $U = 15,453 \text{ V}$



Spanning gaan we altijd meten ten opzichte van een vaste referentie, de “**GROUND**”

De “ground” ligt aan de “-” klem van de bron. We meten in ons voorbeeld aan de “+” klem van de bron een potentiaal van + 15,453 V ten opzichte van de “ground”.
(De ground heeft in ons voorbeeld een potentiaal van 0 V)

We kunnen onze stroomkring nu ook anders gaan tekenen :



STROOM METEN IN ONZE STROOMKRING

Als we de schakelaar dicht doen, vloeit er een stroom van de “+” klem van de bron door de lamp naar de “-” klem van de bron.

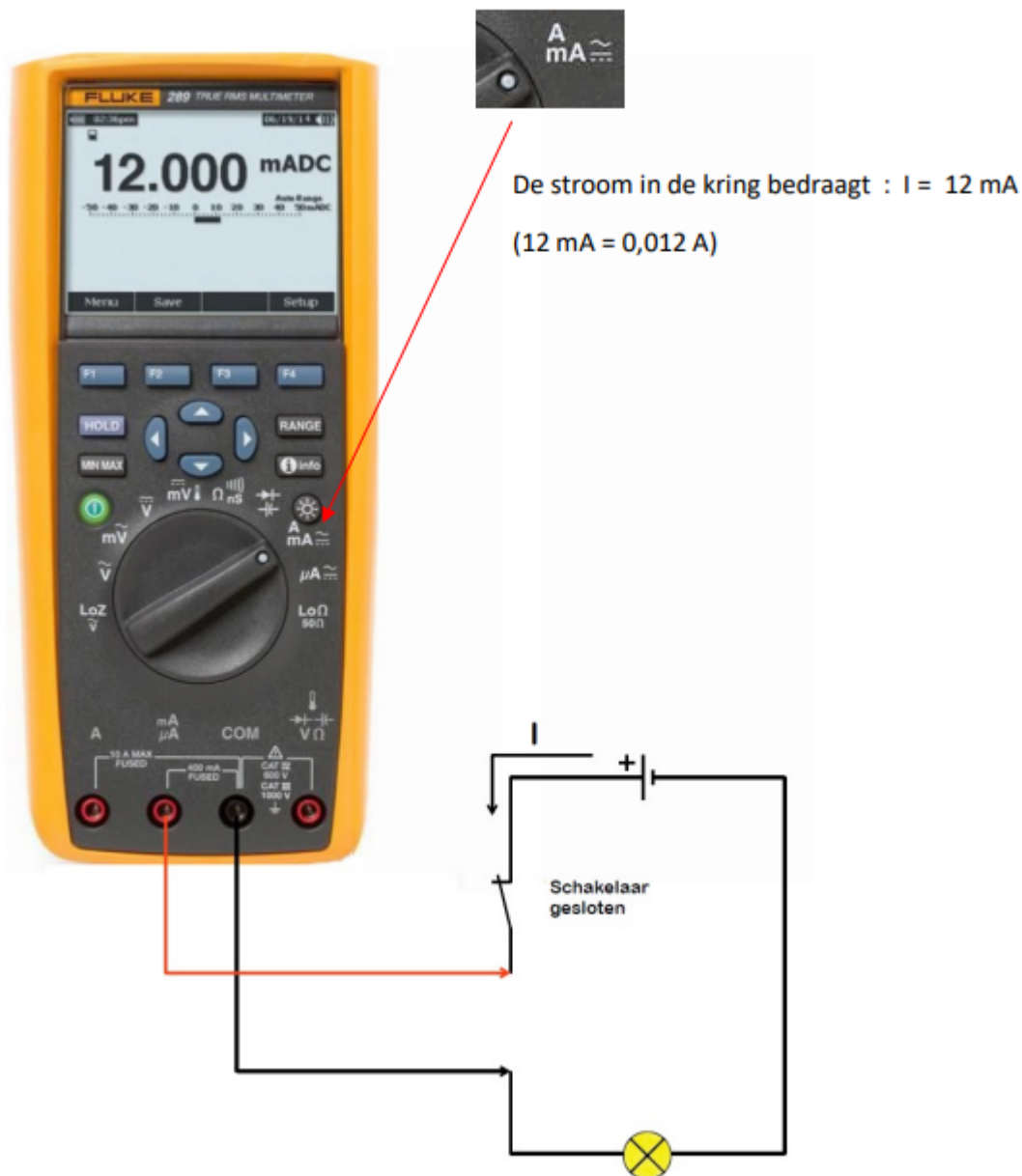
De gelijkstroom in onze kring gaan we meten met een “ampèremeter”.

De A-meter staat **altijd in serie met de verbruiker**. In ons geval is de verbruiker hier een lampje.

De stroom krijgt als symbool de letter “I”

De stroom wordt uitgedrukt in Ampère (A).

We plaatsen de selector van de V-meter op de stand “DC”



WEERSTAND METEN

De weerstand kan gemeten worden met een **Ohmmeter**.

De Ω - meter werkt met een inwendige batterij en mag dus nooit gebruikt worden als er spanning aanwezig is op de kring.

We sluiten de Ω - meter **parallel** aan op de te meten weerstand.

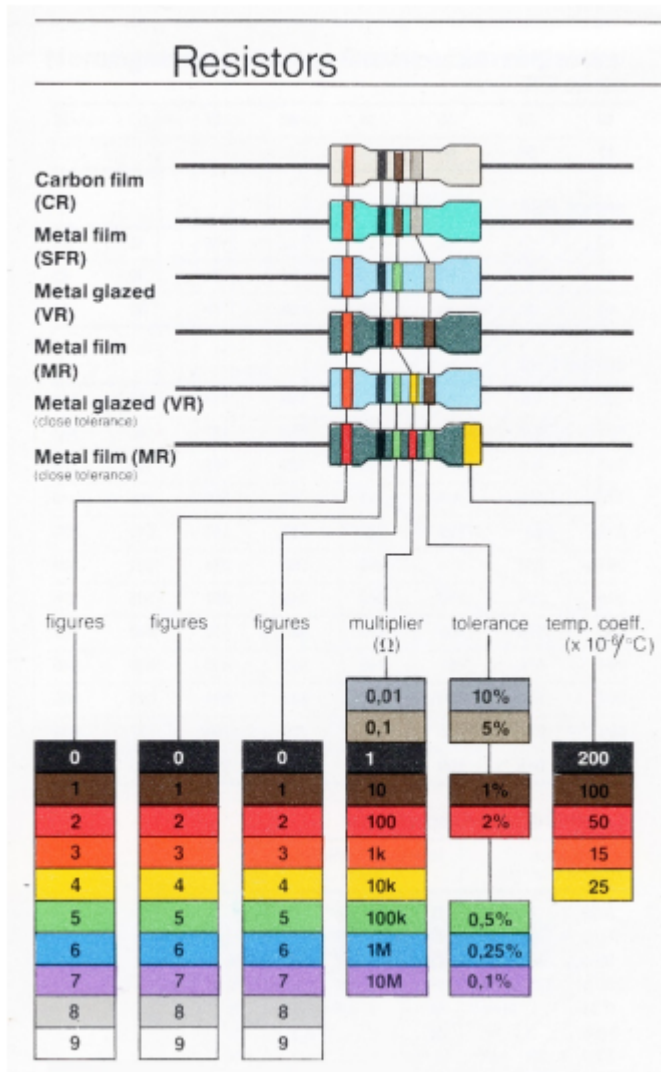


De **weerstand** krijgt als **symbool** de letter “R”

De **weerstand** wordt **uitgedrukt** in Ohm “ Ω ”.

De waarde van de weerstand die we hier meten bedraagt : $R = 1,0108 \text{ k}\Omega = 1010 \Omega$

CODERING VAN WEERSTANDEN EN HOE AF TE LEZEN



De gekleurde ringen die we zien op de weerstand in ons voorbeeld zijn : **Bruin**, **Zwart**, **Rood**, **Goud**

Eerste ring -> Bruin : 1

Tweede ring -> Zwart : 0

Derde ring -> Rood : x 100

Vierde ring -> Goud : tolerantie 5 %

Dus $10 \times 100 = 1000 \Omega$

Tolerantie 5% => **De weerstandswaarde ligt tussen de 1000Ω - 5% en 1000Ω + 5%**

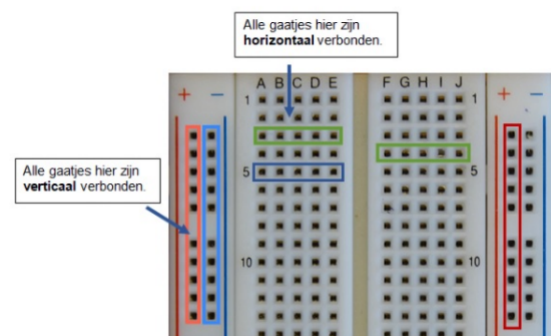
De weerstandswaarde ligt dus tussen 950Ω en 1050Ω

Wij meten 1010Ω

Tijd om zelf eens een schakeling te maken en enkele metingen te doen :

Wat hebben we nodig ?

Een "**Breadboard**" : Dit is een bordje waar je elektronica componenten kan op prikken, zodat je elektronica experimenten kunt uitvoeren. Aan de zijkanten van het bordje zijn al de (+) gaatjes verticaal met elkaar verbonden. De (-) gaatjes zijn ook allemaal verticaal met elkaar verbonden. De rijen in het middenveld zijn horizontaal met elkaar verbonden. Zo kan je gemakkelijk elektrische stroomkringen maken. Controleer dit maar eens met je Ohm-meter.

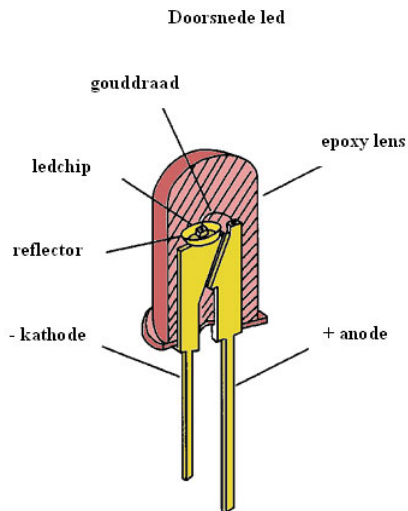


Een "LED"



Een LED is een speciaal soort lampje.

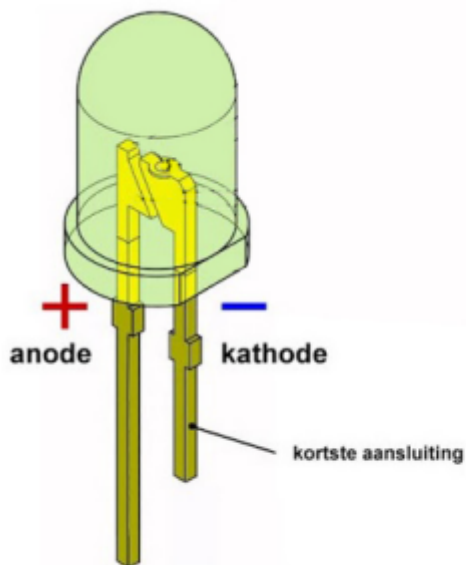
De stroom kan er maar in één richting door. De LED zal branden als de stroom door de LED kan vloeien. **De stroom die door de LED vloeit mag niet groter zijn dan 20 mA** (0,020 A), anders gaat hij stuk. Om ervoor te zorgen dat de stroom door de LED niet groter wordt dan 20 mA zullen we een weerstand in serie plaatsen met de LED.



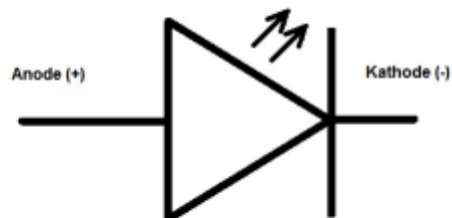
Een LED heeft **twee aansluitingen** : een "**anode**" en een "**kathode**".

De **kathode** heeft het **kortste pootje**.
(de "**k**" van **kathode** is ook de "**k**" van **kort**)

De LED heeft een afgeplatte kant aan de zijde van de kathode.



Symbool :



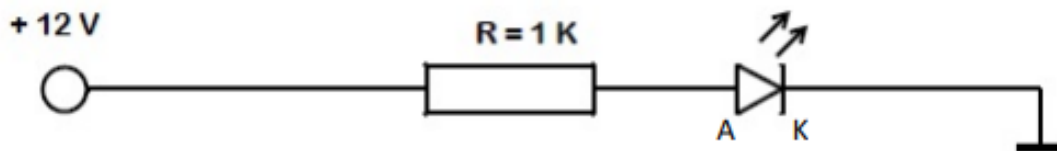
De stroom kan alleen maar vloeien in de richting van de pijl.

Een "weerstand"



We gebruiken een weerstand van 1 k Ω (= 1000 Ω) om de stroom door de LED te beperken

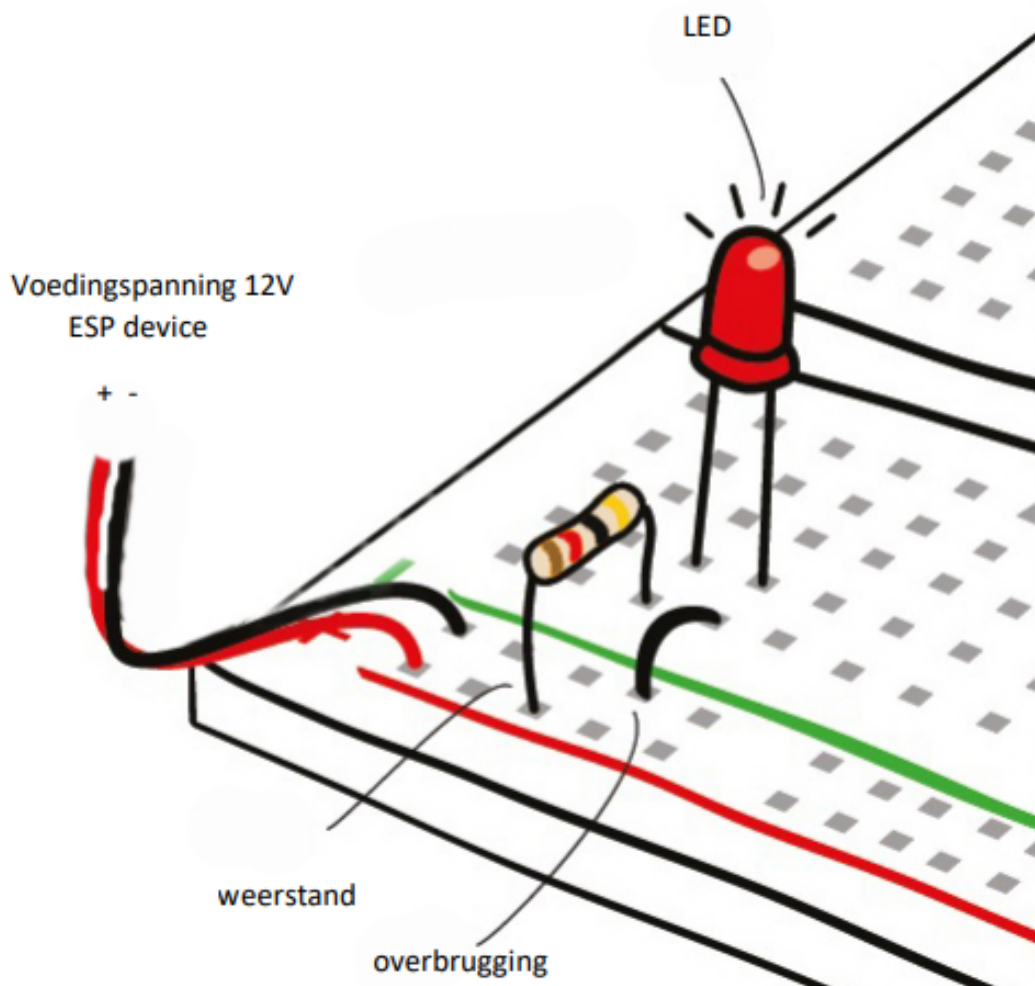
Het schema van onze schakeling zal er als volgt uitzien :



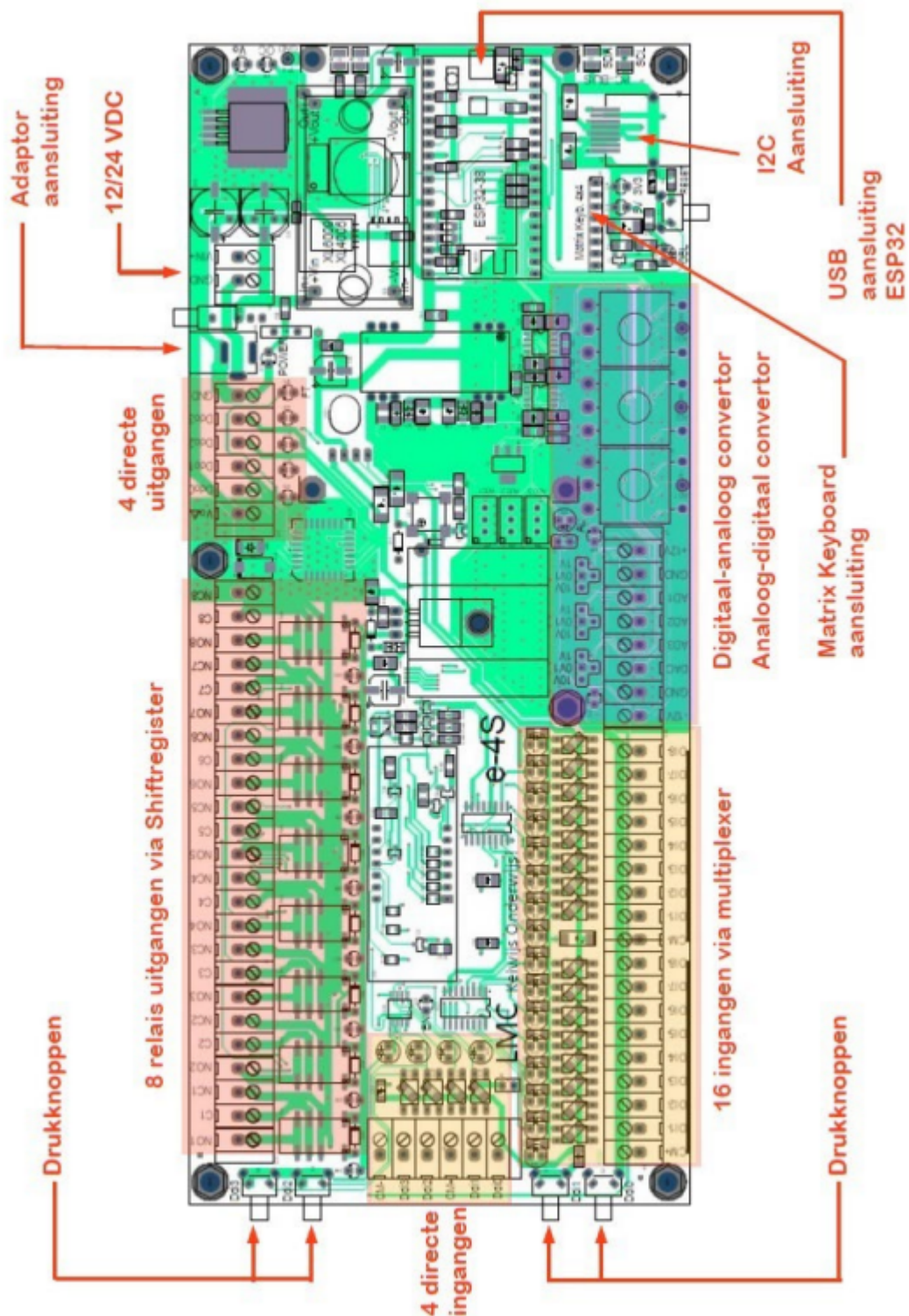
De componenten op het breadbord zijn als volgt opgesteld :

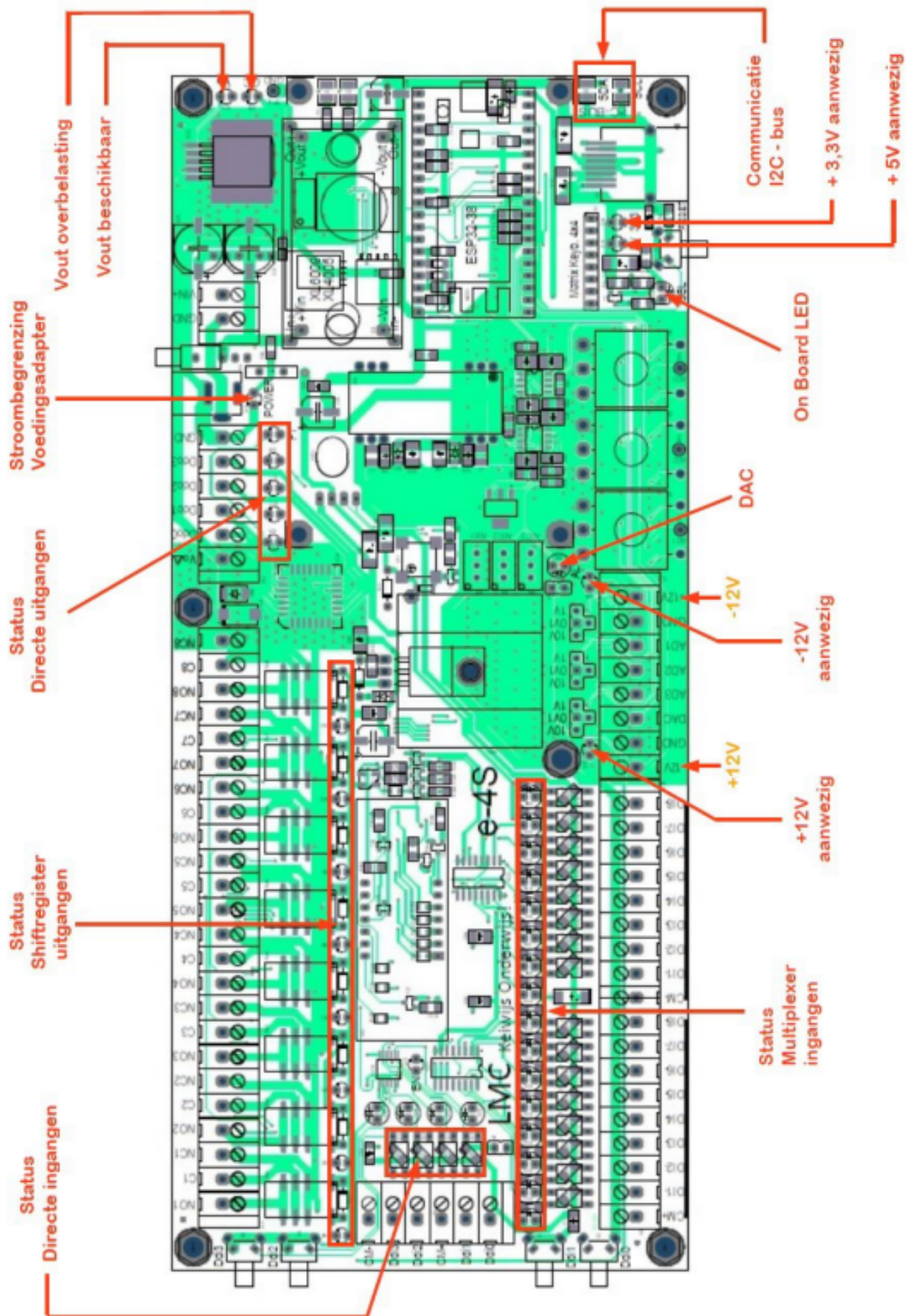
Maak de schakeling en meet volgende zaken :

- de voedingspanning :
- de spanning over de LED :
- de spanning over de weerstand :
- en de stroom door de LED :



Opbouw e-4S ESP32 device [Module 3]



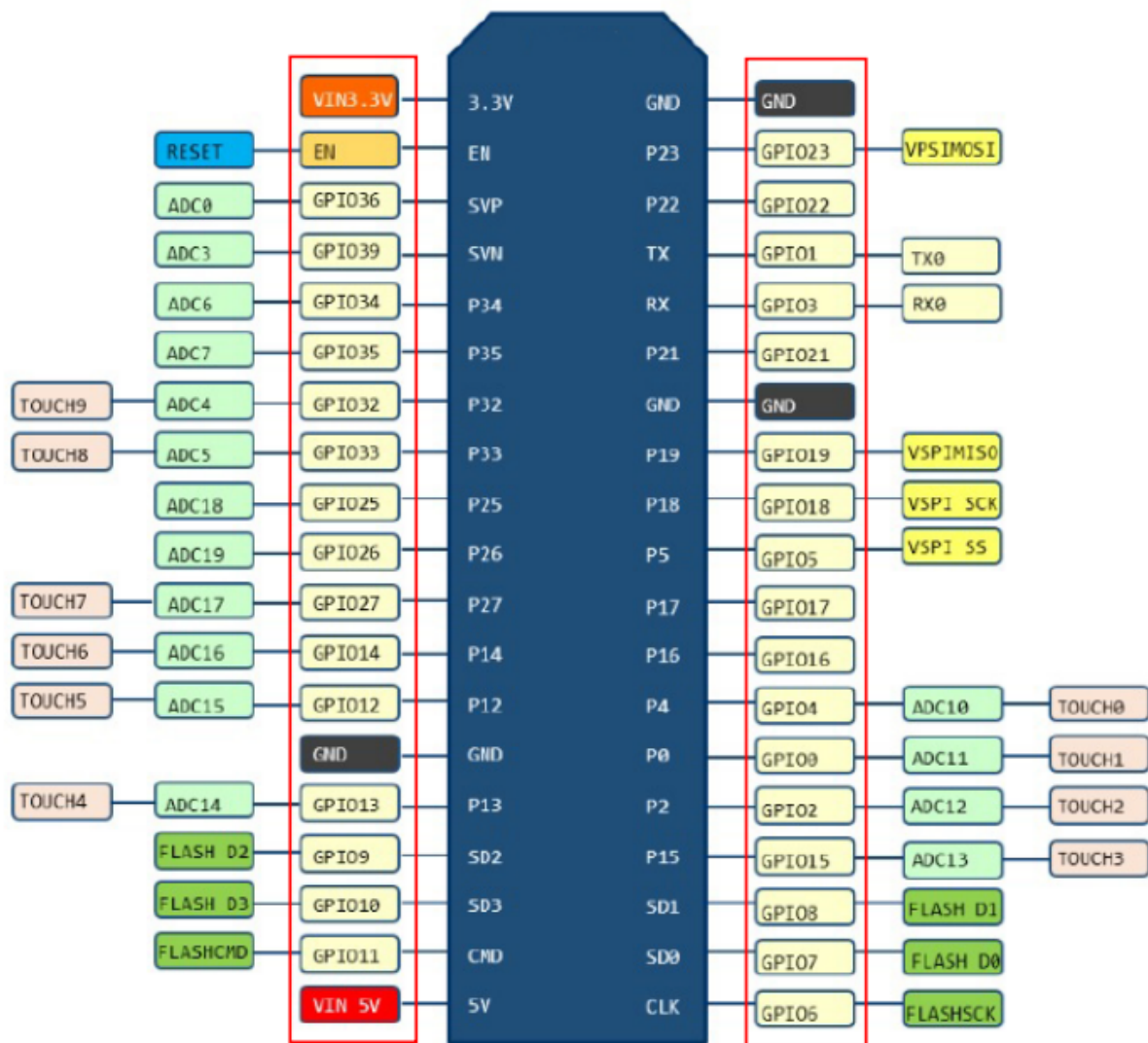


De “ESP32 NodeMCU-32S”

In module 0 hebben we geleerd dat een microcontroller systeem een input, een verwerking en een output bevat



Op het printje van de “ESP32 NodeMCU-32S” zijn er een aantal aansluit pinnetjes die we zowel voor “input” als voor “output” kunnen gebruiken. Alle aansluitingen hebben van de fabrikant een naam gekregen.



“GPIO” betekent “General Purpose Input Output”

Via de ESP-device waar de “ESP32” deel van uitmaakt hebben we volgende aansluitpinnen GPIO16 - GPIO17 – GPIO18 – GPIO19 – GPIO2 (verbonden met onboard LED) gereserveerd als transistor output.

Directe uitgang	GPIO - nummer
OBL	2
Ddo0	16
Ddo1	17
Ddo2	18
Ddo3	19

De aansluitingen GPIO25 – GPIO27 – GPIO32 – GPIO33 hebben we gereserveerd als directe input. (Zij kunnen ieder geactiveerd worden met een drukknopje voorzien op de ESP32 device.)

Directe ingang	GPIO - nummer
Ddi0	32
Ddi1	33
Ddi2	25
Ddi3	27

Volgende aansluitingen zijn gereserveerd als output voor de selectie ingangen van de interne multiplexer GPIO15 – GPIO0 – GPIO5 – GPIO14 (zie module 13) De aansluiting GPIO39 is gereserveerd als input en is verbonden met de uitgang 'y' van de interne multiplexer. (zie module 13)

De overige aansluitingen zijn gereserveerd voor andere doeleinden, maar daar gaan we ons nu nog niets van aantrekken. De ESP32 weet natuurlijk niet welke aansluitingen wij gereserveerd hebben als input of als output !!

Daarom zullen wij dit duidelijk maken aan de ESP32 via het programma dat we zullen schrijven. In module 4 schrijven we ons eerste programma waarin we de ESP32 duidelijk maken dat er een aantal aansluitingen als output zijn voorzien.

De uitgangen van de interne shift registers zijn als volgt gereserveerd :(zie module 15)

Uitgang	Functie
0	RGB LED blauw
1	RGB LED groen
2	RGB LED rood
3	7-segment DEC. DIV.
4	7-segment Digit 4
5	7-segment Digit 3
6	7-segment Digit 2
7	7-segment Digit 1
8	7-segment dp
9	7-segment g
10	7-segment f
11	7-segment e
12	7-segment d
13	7-segment c
14	7-segment b
15	7-segment a
16	Relais 8
17	Relais 7
18	Relais 6
19	Relais 5
20	Relais 4
21	Relais 3
22	Relais 2
23	Relais 1

Blink [Module 4]

Knipperende LED

Op de ESP32 device zitten acht LED's en acht relais. Voor ieder relais die geactiveerd wordt zal er een LED oplichten. Deze acht relais met hun LED's zullen aangestuurd worden via een schuifregister. Later meer hierover.

De ESP32 device heeft ook vier transistor uitgangen waarmee we rechtstreeks een uitwendig relais of/en een LED kunnen aansturen. In het programma dat we nu zullen schrijven, laten we eerst de "onboard" LED die verbonden is met GPIO 2 knipperen.

ACTIE

- ☐ Zorg dat er geen losse onderdelen rond de ESP32 device liggen. Als die per ongeluk onder de ESP32 device terechtkomen, kunnen ze kortsluiting veroorzaken en de elektronica beschadigen.
- ☐ Verbind de ESP32 device via de USB-kabel met je pc.
- ☐ Start de Arduino-software.
- ☐ Kies een nieuwe sketch : open het menu Bestand > Nieuw
- ☐ Selecteer de inhoud van deze sketch en verwijder het.
- ☐ Neem onderstaande code volledig over in je sketch.
- ☐ Bewaar dit op je computer onder de naam : "sketch_blink_ESP32_without_var"
(Bestand > Opslaan Als ...)

```
/*
  Blink
  Op je ESP32 device zie je een LED voor één seconde oplichten,
  vervolgens voor één seconde uitgaan
  en zo verder ...
  Je hoort ook een relais op het zelfde ritme aan en uit gaan.
*/

#include <E4SBoard.h>
E4SClass E4SBoard;

void setup()
{
  E4SBoard.Begin();
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);
}

void loop()
{
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```


In de groene menubalk van het venster zie je een paar knoppen. Beweeg er met je muis overheen om de betekenis van de knoppen te lezen. Klik op de knop Upload.

- Een groene voortgangsbalk geeft het compileren en uploaden van de sketch naar de ESP32 device weer.
- Het programma is ingeladen en de sketch draait nu: de LED die aangesloten is op GPIO 2 knippert 1 seconde aan, 1 seconde uit enzovoort. Tijdens het uploaden wordt de sketch in het geheugen van de ESP32 geplaatst. Dit geheugen is flash-geheugen dus de sketch blijft bewaard, ook als de voeding wegvalt.

Probeer dit uit:

- ☐ Trek de voedingsadapter uit de ESP32 device. Gevolg : De LED dooft.
- ☐ Trek de USB-kabel uit de ESP32 device.
- ☐ Wacht een paar seconden en steek de stekker van de voedingsadapter er weer in.

Gevolg : De sketch start opnieuw en de LED knippert weer.

Eventjes herhalen :

Een sketch bestaat uit minimaal twee functies : “**setup**” en “**loop**”.
tussen de accolades staan statements.

Een **statement** is een regel met code **en aan het eind een puntkomma**.

Verklaring van de statements die we gebruikt hebben in ons programma :

In de “**setup**” zie je bijvoorbeeld volgende statements:

pinMode(2, OUTPUT);

Hier gaan we GPIO 2 als output gaan initialiseren.

Met initialiseren bedoelen we het volgende :

De ESP32 weet de toestand niet van GPIO 2.

Hij kan deze zowel als input of als output gebruiken. Door GPIO 2 als output te initialiseren, zeggen we eigenlijk : Vanaf nu is GPIO 2 een output tot we dit eventueel terug veranderen.

digitalWrite(2, LOW);

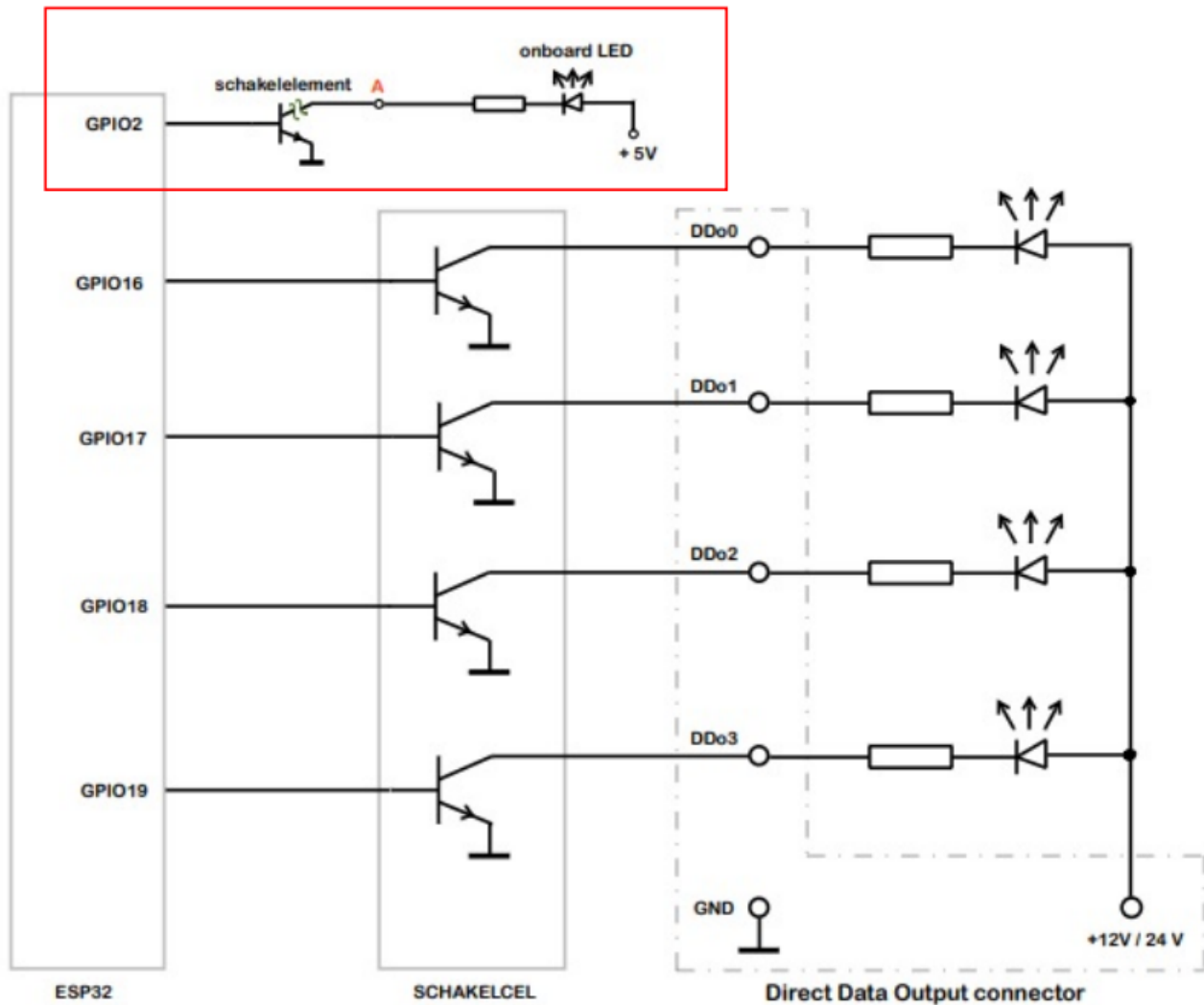
Hier gaan we GPIO 2 om te beginnen op “LOW” zetten.

De ESP32 zal ervoor zorgen dat er een potentiaal van 0V aanwezig is op GPIO 2.

Hierdoor zal het zijn alsof punt ‘A’ niet meer verbonden is met het schakelelement.

Het potentiaal op punt ‘A’ zal zwevend zijn. (geen potentiaal aanwezig)

Hierdoor zal de LED **NIET** branden. (reden : Er kan geen stroom vloeien door de onboard LED.)



In de "loop" zie je volgende statements:

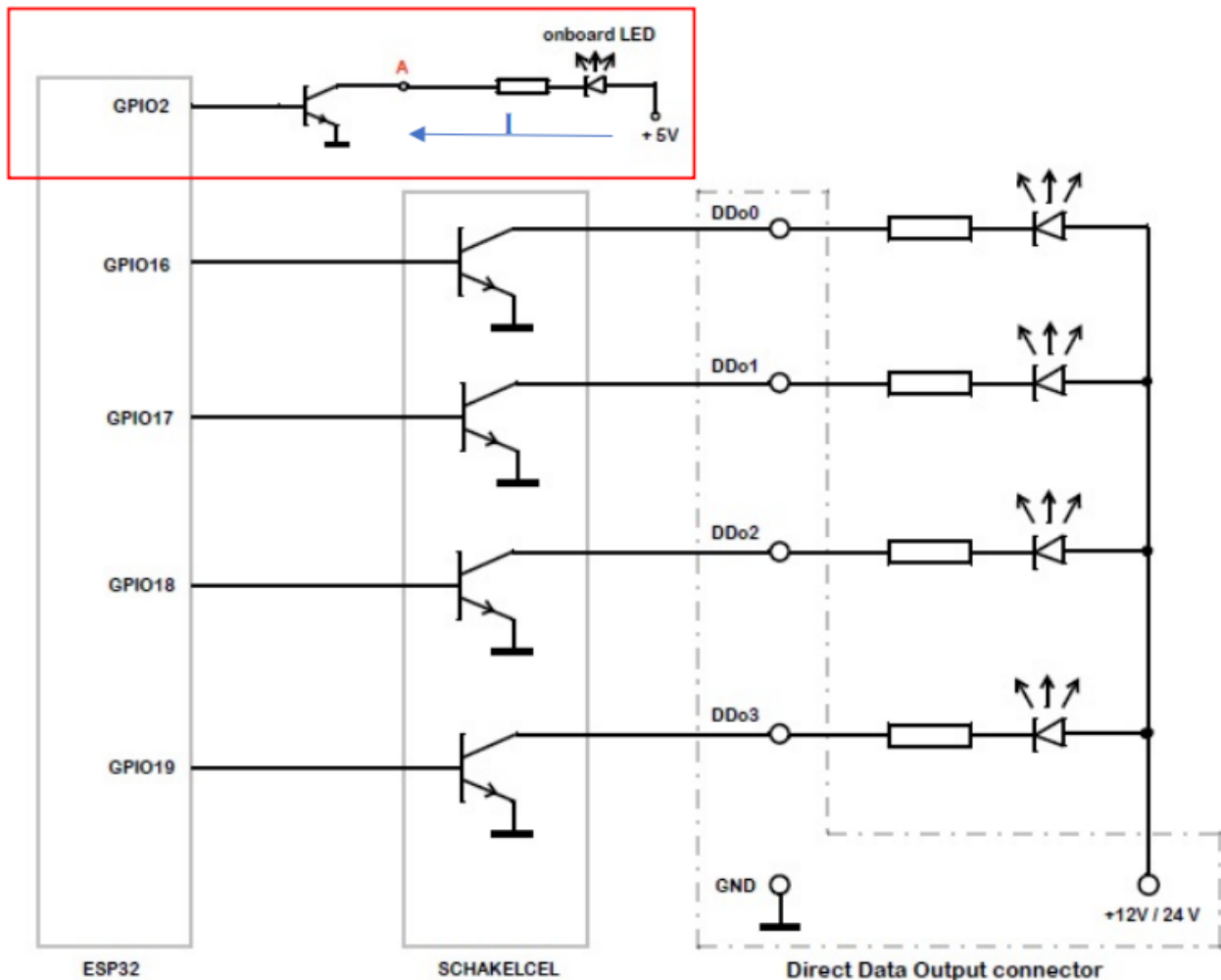
`digitalWrite(2, HIGH);`

Hier gaan we GPIO 2 op "HIGH" zetten.

De ESP32 zal ervoor zorgen dat er een potentiaal van 3,3V aanwezig is op GPIO 2.

Hierdoor zal het potentiaal op punt 'A' nu 0V zijn.

Hierdoor zal de LED branden. (reden : Nu is er een potentiaalverschil aanwezig.)



Delay(1000);

Met dit statement lassen we een pauze in van 1000 milliseconden.

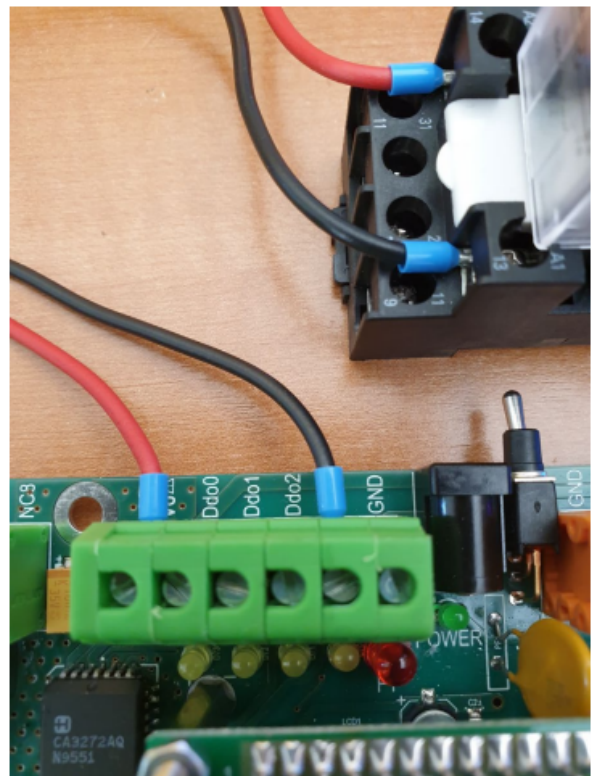
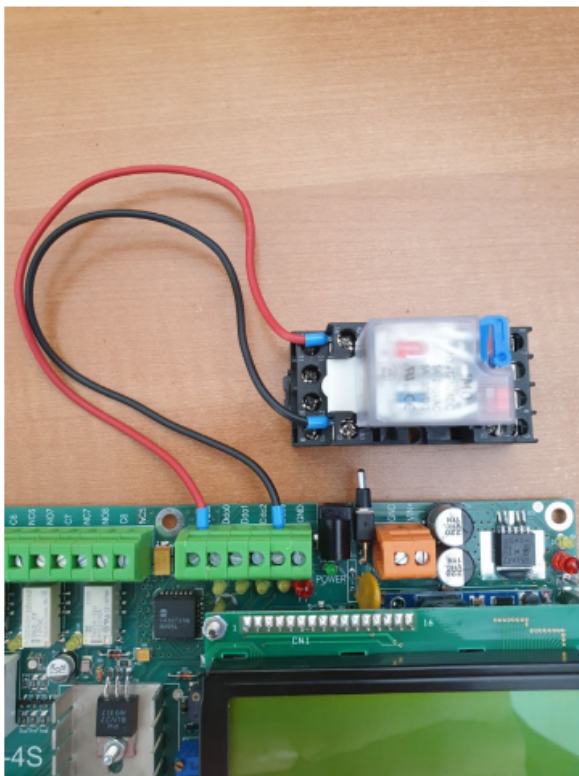
1000 milliseconden = 1 seconde

Zodra de ingestelde pauze om is, gaat de ESP32 verder naar het volgende statement.

We kunnen ook een uitwendig relais activeren met een transistor uitgang.

ACTIE

- ☐ Verander je programma en gebruik GPIO 19 in plaats van GPIO 2
- ☐ Klik op de knop Upload.
- ☐ Verwijder de stekker van de voedingsadapter uit de ESP32 device.
- ☐ Lees eerst de module verdieping 3.
In deze module leer je hoe een relais werkt en hoe je hem moet aansluiten.
- ☐ We sluiten nu een relais aan op DDo3 (Direct Data Output 3) die hoort bij GPIO 19 Zie onderstaande tekening.



- We sluiten de stekker van onze voedingsadapter terug aan op de ESP32 device. Als alles goed werkt zal het uitwendig relais nu voortdurend aan en uit gaan.

Samengevat werkt ons programma als volgt :

De sketch begint met het initialiseren van GPIO 19 als digitale uitgang.
We zetten deze op "LOW" zodat het relais uitgeschakeld is om te beginnen.

Onder de codeblok van de "setup" begint de "loop".

Alles wat in deze "loop" staat, wordt continu herhaald.

We zorgen ervoor dat GPIO 19 op "HIGH" staat zodanig dat het relais geactiveerd wordt.
Vervolgens wachten we 1 seconde en dan zetten we GPIO 19 terug op "LOW" zodat het relais terug uitgaat. Daarna wachten we terug 1 seconde. Om vervolgens het proces in de "loop" te herhalen, enzovoort ...

```
/*  
Blink  
Op je ESP32 device zie je een LED voor één seconde oplichten,  
vervolgens voor één seconde uitgaan en zo verder ...  
Je hoort ook een relais op het zelfde ritme aan en uit gaan.  
*/  
  
#include <E4SBoard.h>  
E4SClass E4SBoard;  
  
void setup()  
{  
    E4SBoard.Begin();  
    pinMode(19, OUTPUT);  
    digitalWrite(19, LOW);  
}  
  
void loop()  
{  
    digitalWrite(19, HIGH);  
    delay(1000);  
    digitalWrite(19, LOW);  
    delay(1000);  
}
```

*Tijd voor een paar opdrachten
met wat we tot nu toe al geleerd hebben*

OPDRACHTEN :

We willen een “**Vermoeidheids Indicator**” maken.

Als de LED één maal brandt per seconde, dan kunnen we zeggen dat de LED knippert met een frequentie van één Hertz. ($f = 1\text{Hz}$)

Schrijf een programma waarbij je test welke knipperfrequentie je nog kan waarnemen. Hoe meer je vermoeid bent, hoe lager de knipperfrequentie die je kunt waarnemen.

We willen een “**SOS signaal**” laten zien.

Morsecode bestaat uit korte en lange signalen met daartussen een korte pauze.

Elke letter en elk cijfer heeft zijn eigen combinatie (zie onderstaande tabel).

De code werd uitgevonden door Samuel Morse (1835) en is een soort voorloper van de latere digitale communicatie met enen en nullen. Je kunt de signalen op allerlei manieren doorgeven, bijvoorbeeld door op tafel te kloppen, tegen een tralie of verwarmingsbuis te tikken (wat in gevangenissen veel werd gedaan), met lampen (scheepvaart), met telegrafie signalen enz.... Een bekend morsecode voorbeeld uit de scheepvaart en luchtvaart is SOS.

A	·—	J	·— — —	S	...·	1	·— — — —
B	—...·	K	—·—	T	—·—	2	··— — —
C	—·—·	L	·—...·	U	··—	3	...— —
D	—...·	M	— —	V	...—	4	—
E	·	N	—·	W	·— —	5	
F	··—·	O	— — —	X	—...—	6	—....
G	— — ·	P	·— — ·	Y	—·— —	7	— — ...
H	·	Q	— — · —	Z	— — ...	8	— — — ·
I	··	R	· — ·	0	— — — — —	9	— — — — ·

Het morse alfabet.

Ga aan de slag met de ESP32 device en zorg dat de LED die aangesloten is op GPIO 19 het SOS signaal laat zien. Het signaal moet zich voortdurend herhalen.

Een kort lichtsignaal duurt 100 ms, een lang lichtsignaal duurt 300 ms, de pauze na elke letter is 500 ms en de pauze na elk woord is 1000 ms.

Spoorwegovergang

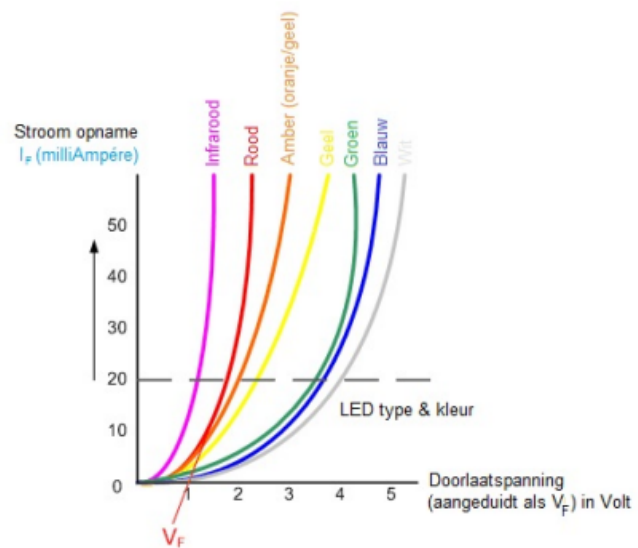
Twee LED's laten knipperen die extern aangesloten zijn op GPIO18 en GPIO19.

Uit Module 3 weet je inmiddels, dat je een voorschakelweerstand nodig hebt om een LED te laten oplichten. Deze voorschakelweerstand dient om de stroom door de LED te beperken. Je mag dus nooit een LED zonder voorschakelweerstand op een spanningsbron aansluiten. De LED zal onmiddellijk stuk gaan als gevolg van de veel te grote stroom die erdoor gaat. De maximum stroom (I_F) die we door een LED mogen sturen is 20 mA (0,020 A)

LED's bestaan in verschillende kleuren. De spanningsval (V_F) over de LED als hij oplicht is vrijwel constant en verschillend naargelang het kleur. (zie onderstaande tabel)

KLEUR	GELEIDINGSSPANNING
Infrarood	kleiner dan 1,6 V
Rood	1,6 V tot 2,0 V
Orange	2,0 V tot 2,1 V
Geel	2,1 V tot 2,2 V
Groen	1,9 V tot 4,0 V
Blauw	2,5 V tot 3,7 V
Paars	2,7 V tot 4,0 V
Ultraviolet	3,0 V tot 4,1 V

De geleidingsspanning over een LED.



- ☐ We gaan nu de voorschakelweerstand berekenen die we nodig hebben respectievelijk voor een rode LED en een witte LED.
- ☐ We sluiten de rode LED met zijn voorschakelweerstand aan op GPIO18 en de witte LED met zijn voorschakelweerstand op GPIO19.

Niet alle weerstanden die we berekenen zijn zomaar in de handel verkrijgbaar. Weerstand worden opgedeeld in E-reeksen.

Een tabel met praktische weerstandswaarden vind je op volgende link :

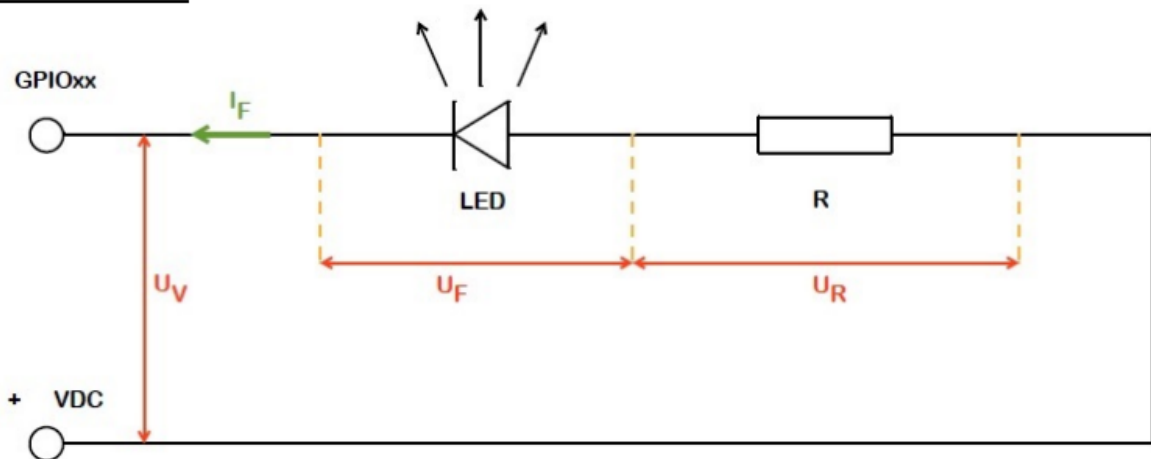
<https://verstraten-elektronica.blogspot.com/p/weerstandswaarden-volgens-de-e-reeksen.html>

We gebruiken een weerstand van 1200Ω uit de E12 reeks.

Meer bijzonderheden over E-reeksen bij weerstanden vind je in de module "Verdieping 2".

Bereken de stroom die door de LED vloeit bij een voorschakelweerstand van 1200Ω .

Situatieschema :



Voorschakelweerstand Rode LED :

$$R = \frac{U_R}{I_F} = \frac{U_V - U_F}{I_F} = \frac{24 - 1,6}{0,020} = 1120 \, \Omega$$

$$I_F = \frac{U_V - U_F}{R} =$$

Bereken de voorschakelweerstand voor de Witte LED :

De geleiding spanning over een witte LED is gelijk aan deze van een ultraviolet LED.
Je mag hiervoor een waarde van 3,6 V nemen.

$R =$

Wat is de waarde van de voorschakelweerstand die we gaan gebruiken voor de witte LED ?

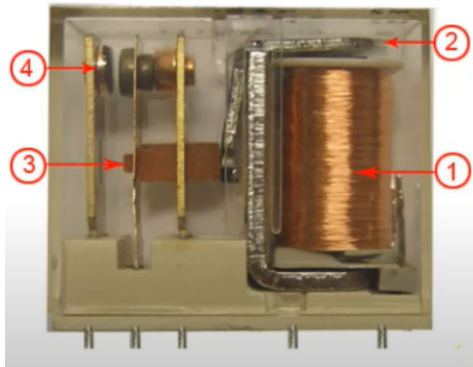
$R =$

Schrijf een programma om de beide LED's om-en-om te laten knipperen.
Je laat ze 1 seconde oplichten en 1 seconde uit enz...

Relais - [Verdieping 3]

Definitie :

Een relais is een door een magnetisch veld bediende mechanische schakelaar. Het wordt vooral gebruikt als eind component in digitale schakelingen.

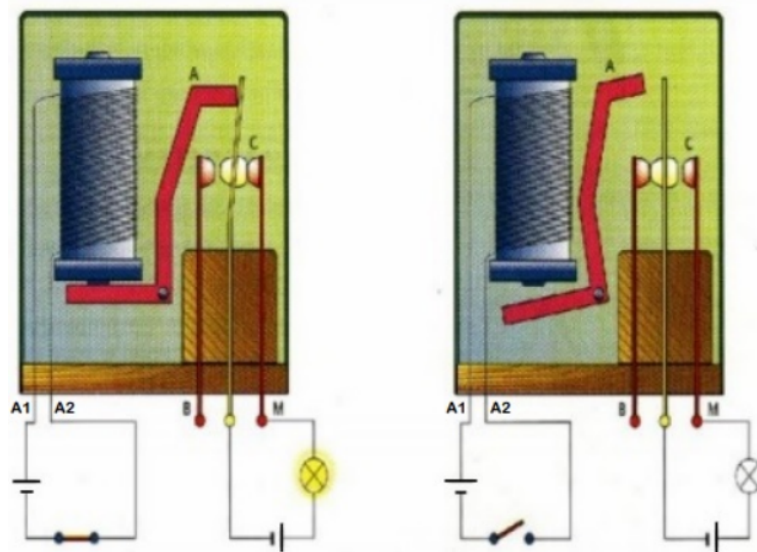


Constructie :

Een relais bestaat uit een elektromagneet, de spoel (1) met hierin een (U-vormige) kern, het anker (2), een plaatje "weekijzer" (A) dat aangetrokken wordt door de elektromagneet met hieraan bevestigd één of meerdere contacten, één of meerdere vaste contacten en een veermechanisme.

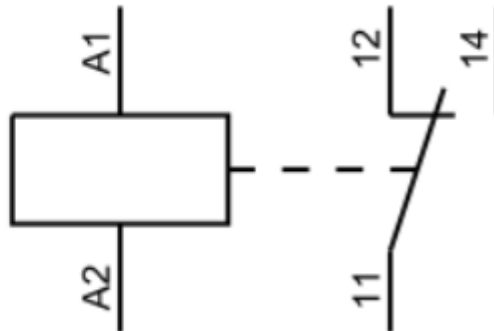
Weringsprincipe :

Door een passende spanning op de spoel aan te sluiten (A1-A2) gaat er een stroom lopen door de elektromagneet en wordt een magnetisch veld opgewekt, waardoor het ijzeren anker (A) aangetrokken wordt en de contacten bediend worden. Daardoor zal de contact overgang B-C openen en M-C sluiten. Valt de spanning weg, dan verdwijnt het magnetisch veld en zorgt het veermechanisme ervoor dat het ijzeren anker (A) met de contacten terugkeert in de oorspronkelijke stand (de onbekerachtigde of ruststand). Daardoor zal de contact overgang B-C sluiten en M-C openen.



De meeste relais hebben ten minste één set van drie contacten.
moedercontact (C), maakcontact (M) en verbreekcontact (B)

Symbol :



Voor gelijkspanning spoelen maakt het niet uit welke polariteit de aangelegde spanning heeft. In sommige relais is echter een vrijlooptiode over de spoel geïntegreerd, waardoor de polariteit wel vastligt. Later meer hierover.

Voor wisselspanning spoelen maakt de polariteit nooit uit.

Contactoren hebben dezelfde werking als relais maar hun contacten zijn veel zwaarder uitgevoerd. Op deze manier kunnen ze verbruikers aansturen die een veel grotere stroom trekken.



Variabelen [Module 5]

ACTIE

Open het bestand “*sketch_blink_ESP32_without_var*” dat je hebt opgeslaan op je computer.
(Bestand > Open ...)

```
/*
Blink
Op je ESP32 device zie je een LED voor één seconde oplichten,
vervolgens voor één seconde uitgaan en zo verder ...
Je hoort ook een relais op het zelfde ritme aan en uit gaan.
*/

#include <E4SBoard.h>
E4SClass E4SBoard;

void setup()
{
    E4SBoard.Begin();
    pinMode(19, OUTPUT);
    digitalWrite(19, LOW);
}

void loop()
{
    digitalWrite(19, HIGH);
    delay(1000);
    digitalWrite(19, LOW);
    delay(1000);
}
```

- Verander het programma zodanig dat de LED die aangesloten is op GPIO 19 niet meer geprogrammeerd is, maar dat er nu wel een LED knippert die aangesloten is op GPIO 16.
- Upload het programma.

Wat heb je gedaan ?

Je hebt op 4 plaatsen het getal 19 moeten veranderen naar 16. Stel dat het programma veel groter zou geweest zijn en dat je op 100 plaatsen dit getal zou moeten wijzigen... Stel dat je een getal vergeet te wijzigen... Dat zou wel vervelend zijn en mogelijks ten koste van de goede werking van je programma. Om deze situaties te vermijden, gaan we werken met **variabelen**.

Wat is een variabele

Een variabele kunnen we vergelijken met een doos. Er zijn verschillende soorten dozen, waar verschillende inhoud inpassen. (vb. speelgoed_auto, brieven, cursussen enz...)

Op de doos plak je een label met een naam om deze later terug te vinden.

Waar je de doos ook mee naartoe neemt, de inhoud blijft erin zitten.

Tijdens het verhuizen kan je bijvoorbeeld de inhoud (vb. gele_auto) uit de doos halen om er een andere gele speelgoed auto in te stoppen.

Naam van een variabele

De naam van de variabele mag je zelf verzinnen. Je bent daar redelijk vrij in, maar er zijn wel enkele beperkingen waar je rekening mee moet houden :

- De naam mag alleen letters, cijfers en de underscore “_” bevatten.
- De naam mag niet beginnen met een cijfer.
- De naam mag geen gereserveerd woord zijn van C.
- Namen van variabelen mogen geen spaties bevatten.

Het is van groot belang dat je zinvolle namen kiest voor je variabelen, dat maakt het lezen van je broncode een stuk makkelijker. Met zinvolle namen bedoel ik namen die betrekking hebben op het doel of de functie van de variabele.

Als de **naam** bestaat uit meerdere woorden, dan schrijven we deze in een “**CAMELCASE**”.

We beginnen **elk woord met een hoofdletter** en we plakken de **woorden aan elkaar**.

vb. myFirstName

DATATYPE

Wanneer je de computer vraagt iets met die variabele (doos) te doen, dan verricht het een handeling

met wat er op dit moment in de variabele zit. De variabelen in onze programmeertaal C zijn ontworpen om specifieke soorten waarden te bevatten. Als C een programmeertaal zou zijn die zich bezighoudt met levensvormen, zouden katten en honden deel uitmaken van het variabele type “dier”. Bomen en varens zouden dan deel uitmaken van het variabele type “plant”.

De variabelen die we gebruiken in C werken op een soortgelijke wijze, waarbij voor specifieke waarden bijbehorende variabelen typen worden toegekend. Dus als we een variabele willen gebruiken, dan moeten we de compiler vertellen wat voor soort gegevens we in de variabele willen opslaan. Het soort gegevens noemen we “**datatype**” of kortweg “**type**”.

Voorbeelden van “datatype” zijn : int, long, float, string enzovoort....

Het laatste blad van deze module heeft een overzicht van alle data types die er bestaan.

Hoe gaan we tewerk om variabelen te gebruiken ?

DECLARATIE en INITIALISATIE

Voordat we een variabele kunnen gebruiken, moeten we kenbaar maken hoe de variabele heet en van welk type hij is. Dit noemen we een “**declaratie**”.

Hieronder zie je een aantal voorbeelden van “**declaratie**”

```
// reserveer geheugen om tekst op te slaan
string myText;

// reserveer geheugen om een geheel getal op te slaan
int myInteger;

// reserveer geheugen om een gebroken getal op te slaan
float myFloatingPoint;
```

Datatype →

De **declaratie** van een variabele moet altijd gebeuren, voordat de variabele voor het eerst gebruikt wordt.

We kunnen een gedeclareerde variabele een waarde geven zoals in onderstaand voorbeeld.

```
// declareer een variabele om tekst in op te slaan
string myText;

// geeft de variabele een waarde
myText = "Hello, world!";
```

Met getallen werkt het op dezelfde manier dit is te zien hieronder

```
// declareer een variabele om een geheel getal in op te slaan

int myInteger;

// geef de variabele een waarde
myInteger = 101;
```

Omdat het heel vaak voorkomt, dat je een variabele een beginwaarde wilt meegeven, kun je dit doen tijdens de declaratie. We zeggen dan dat je de variabele initialiseert. Dat ziet er als volgt uit :

```
// initialiseer een tekst variabele met de tekst Hello, world!
string myText = "Hello, world!";

// initialiseer een geheel getal met de waarde 101
int myInteger = 101;
```

Type → Naam → Inhoud

OK, terug naar onze sketch die we reeds geschreven hadden. We zullen deze nu aanpassen met gebruik van variabelen. Als ik nu een andere GPIO wil gebruiken, dan hoef ik alleen maar één maal de inhoud van variabele Do te veranderen.

```
/*
  Blink
  Op je ESP32 device zie je een LED voor één seconde oplichten,
  vervolgens voor één seconde uitgaan en zo verder ...
  Je hoort ook een relais op het zelfde ritme aan en uit gaan.
*/

#include <E4SBoard.h>
E4SClass E4SBoard;

byte Do = 16;

void setup()
{
  E4SBoard.Begin();
  pinMode(Do, OUTPUT);
  digitalWrite(Do, LOW);
}

void loop()
{
  digitalWrite(Do, HIGH);
  delay(1000);
  digitalWrite(Do, LOW);
  delay(1000);
}
```

Bereik van een variabele (scope)

Als je een variabele definieert in een programma blok (routine) dan is die variabele alleen bekend binnen die programma blok. Dat klinkt ingewikkeld dus geef ik je hier een voorbeeld.

```
void setup()
{
  int a=5;
}
void loop()
{
  a=6;
}
```

Dit bovenstaande programma zal een foutmelding geven.

Reden :

De variabele a wordt in de setup() routine gedefinieerd en is daarom niet bekend in de loop() routine.

De variabele a is een zogenaamde “**local variabele**” (plaatselijke variabele). **Hij is alleen bekend in de routine waar hij aangemaakt wordt.** Om een variabele in het hele programma beschikbaar te maken, moet je hem aan het begin van het programma, nog voor de setup, definiëren.

```
int a=5;
void setup()
{
  a=3;
}
void loop()
{
  a=4;
}
```

Dit programma zoals hierboven zal **geen** foutmelding geven.

Reden :

De variabele a wordt **aan het begin van het programma gedefinieerd** en daardoor kan hij overal in het **programma gebruikt en veranderd worden**. Dit heet dan ook een “**global variabele**” (globale variabele). De eindwaarde van de variabele a in ons voorbeeld is 4.

Variabelen die niet veranderd kunnen worden.

Soms heb je **waarden** nodig **die door de rest van het programma niet veranderd mogen worden**. Daarvoor kunnen we een “**constante variabele**” aanmaken met het commando “**const**”.

```
const int a=5;
void setup()
{
}
void loop()
{
  a=4;
}
```

Dit programma zal **absoluut een foutmelding** veroorzaken.

Reden :

Je hebt de variabele a als een const (vaste waarde) gedefinieerd en je probeert hem in de “loop” te veranderen.

Alle typen variabelen (int, float, String, bool enz...) kunnen als local en global variabelen gedefinieerd worden en kunnen ook als “const” ingezet worden.

#define

Een alternatieve manier van constanten gebruiken is met de “**define**” compiler instructie.

#define AantalLichten 5

We zien hier een aantal belangrijk verschillen.

Als eerste, starten we deze regel met een hash (#). We hebben dat nog niet eerder gezien en het geeft de compiler (vertaler) specifieke opdrachten die uitgevoerd worden voor de vertaling start. In principe zegt het “voor je gaat vertalen, vervang over het gehele programma alle “AantalLichten” in de code met het nummer 5”.

Dus in tegenstelling met “const” methode, wijst deze dus niet naar een geheugen locatie! Er zijn nog meer compiler instructies maar we laten het even bij deze. In tegenstelling tot gewone instructies zien we ook dat de puntkomma ontbreekt (;) Dit is alleen correct voor compiler instructies en commentaar. En als laatste typen we gewoon “5” in plaats van “= 5”

Overzicht van alle typen variabelen

Boolean

Een Boolean variabele kan slechts twee waarden aannemen: TRUE of FALSE of "1" of "0" in het volgende voorbeeld wordt variabele "DeZonSchijnt" gedeclareerd als variabele van het type Boolean en krijgt hij de beginwaarde TRUE. Voor **dit type variabele** reserveert de ESP32 **1 byte geheugenruimte**.

```
boolean DeZonSchijnt = true;
```

```
boolean DeZonSchijnt = 1;
```

Char

Char is de afkorting van character value of tekenwaarde. Een variabele van het type "char" is een reeks met combinatie van acht enen en nullen (byte), dat een teken van bijvoorbeeld het toetsenbord vertegenwoordigd. Druk je bijvoorbeeld op de toets 'A' dan wordt dit aan een char variabele toegewezen als "10000001" (zie ASCII-tabel in Appendix 1, achteraan deze cursus) Dit komt overeen met ASCII-symbool 'A' en decimale waarde 65.

Opmerking : **Met variabelen van het type char kan je geen rekenkundige bewerkingen uitvoeren.**

```
char myChar = 'A';
```

ofwel

```
char myChar = 65;
```

Beide manieren van schrijven komen op hetzelfde neer.

byte

Een variabele van het type "byte" vertegenwoordigd 8-bits. (zie Verdieping 6)

Een byte kan een decimale getalswaarde hebben tussen 0 en 255. In het volgende voorbeeld wordt variabele myPocket gedeclareerd als variabele type byte met de beginwaarde B10011. Hierin staat "B" voor binair en is 10011 gelijk aan (decimaal) 19.

```
byte myPocket = B10011;
```

int

Een variabele van het type "int" (integer) vertegenwoordigt een positief of negatief getal zonder komma tussen 32.797 en -32.768. Wordt de variabele groter dan 32767 dan wordt hij negatief. Voor **dit type variabele reserveert de ESP32 16 bits (= 2 bytes) geheugenruimte**.

Het volgende voorbeeld declareert variabele Do als integer met de beginwaarde 5.

```
int Do = 5;
```

unsigned int

Een variabele van het type “unsigned int” vertegenwoordigt een positief decimaal getal zonder komma tussen 0 en 65535. Voor dit type variabele reserveert de ESP32 2 bytes geheugenruimte.

unsigned int Inwoners1 = 27423;

long

Een variabele van het type “long” vertegenwoordigt een positief of negatief decimaal getal zonder komma tussen 2.147.483.647 en -2.147.483.648.

Voor dit type variabele reserveert de ESP32 32bits (= 4 bytes) geheugenruimte.

long langeVariabele = 90000;

unsigned long

Zelfde als long, maar voor positieve decimale getallen tussen 0 en 4.294.967.295.

float

Een variabele van het type “float” vertegenwoordigt een positief of negatief decimaal getal met cijfers achter de komma tussen $3,4028235 \times 10^{38}$ tot $-3,4028235 \times 10^{38}$.

Voor dit type variabele reserveert de ESP32 32 bits (= 4 bytes) geheugenruimte.

In plaats van een komma tussen de cijfers wordt een punt gebruikt, net als in de Engelse taal.

float tweepiVariabele = 6.28318;

double

Een variabele van het type “double” vertegenwoordigt een zeer groot decimaal getal met cijfers achter de komma. De maximale waarde is $1,7976931348623157 \times 10^{308}$.

Voor dit type variabele reserveert de ESP32 4 bytes geheugenruimte.

string

Een variabele van het type “string” vertegenwoordigt een reeks ASCII-tekenen.

Per teken reserveert de ESP32 8bits (= 1 byte) geheugenruimte.

array

Een variabele van het type “array” vertegenwoordigt een serie waarden.

Elke waarde is aan te wijzen door middel van een index die vanaf 0 begint te tellen.

In het onderstaande voorbeeld bevat de array met de naam koelFan acht integer waarden.

De waarde “20” heeft index “0”, de waarde “40” heeft index “1” enzovoort.

int koelFan[8] = {20, 40, 80, 120, 160, 200, 240, 255};

i = koelFan[4] // dus i = 160

Met variabelen werken

ACTIE

Oefening 1

Maak een nieuw programma zoals in onderstaand voorbeeld.

```
#include <E4SBoard.h>
E4SClass E4SBoard;

char c='a';
int i=1;
float f=19.0;
double d=20000.009;

void setup()
{
    E4SBoard.Begin();
    Serial.begin(9600);
    Serial.print("De inhoud van variabele c = ");
    Serial.println(c);
}

void loop()
{
}
```

- ☐ Sla dit programma op onder de naam “Variabelen zichtbaar maken”.
- ☐ Zorg dat de seriële monitor zichtbaar is op het beeldscherm van je computer.
- ☐ Load het programma op.
- ☐ Pas het programma aan zodat alle variabelen zichtbaar zullen zijn op de seriële monitor.
- ☐ Load het programma op en controleer of alle variabelen zichtbaar zijn op de seriële monitor.
- ☐ Sla deze aanpassing op.

Oefening 2

Bekijk volgende broncode waar we een globale variabele "A" aanmaken en zijn waarde in de beide routines "setup" en "loop" aanpassen. We brengen voor iedere aanpassing van "A" de inhoud van "A" op de seriële monitor. Wat verwacht je te zien op de seriële monitor ?

```
#include <E4SBoard.h>
E4SClass E4SBoard;

int A = 0;

void setup()
{
    E4SBoard.Begin();
    Serial.begin(9600);
    Serial.print("Setup: Eerste A = ");
    Serial.println(A);

    A = 10;
    //print de waarde naar de seriële monitor
    Serial.print("Setup: Tweede A = ");
    Serial.println(A);
}

void loop()
{
    Serial.print("Loop: Eerste A = ");
    Serial.println(A);

    A = 20;
    //print de waarde naar de seriële monitor
    Serial.print("Loop: Tweede A = ");
    Serial.println(A);

    delay(5000);
}
```

Schrijf dit programma en controleer je antwoord

Oefening 3

Bekijk onderstaand programma waar we een globale variabele "B" aanmaken en een locale variabele "B" aanmaken zowel in de routine "setup" als in de routine "loop". Wat verwacht je te zien op de seriële monitor ?

```
#include <E4SBoard.h>
E4SClass E4SBoard;

int B = 0;

void setup()
{
    E4SBoard.Begin();
    int B;
    B = 1;
    Serial.begin(9600);
    //print de waarde naar de seriële monitor
    Serial.print("Setup: B = ");
    Serial.println(B);
}

void loop()
{
    int B;
    B = 2;
    //print de waarde naar de seriële monitor
    Serial.print("Loop: B = ");
    Serial.println(B);

    delay(5000);
}
```

Schrijf dit programma en controleer je antwoord.

Oefening 4

Om het effect van LOCAAL en GLOBAAL te zien, gaan we nu even in de “setup” routine twee regels uitschakelen met behulp van commentaar-tekens (//) Bekijk de aanpassing. Wat verwacht je te zien op de seriële monitor ?

```
#include <E4SBoard.h>
E4SClass E4SBoard;

int B = 0;

void setup()
{
    E4SBoard.Begin();
    //int B;
    //B = 1;
    Serial.begin(9600);
    //print de waarde naar de seriële monitor
    Serial.print("Setup: B = ");
    Serial.println(B);
}

void loop()
{
    int B;
    B = 2;
    //print de waarde naar de seriële monitor
    Serial.print("Loop: B = ");
    Serial.println(B);

    delay(5000);
}
```

Pas het programma aan en controleer je antwoord.

De globale variabele is nu in gebruik in de “setup” routine. In de “loop” routine wordt deze globale variabele niet gebruikt omdat er een lokale variabele is met dezelfde naam.

Bewerkingen uitvoeren met variabelen

Oefening 5

```
#include <E4SBoard.h>
E4SClass E4SBoard;

void setup()
{
    E4SBoard.Begin();
    // snelheid voor de seriële monitor
    Serial.begin(9600);

    // variabelen definiëren met datatype "int"
    int ZakGeld;
    int SpaarGeld;
    int AlMijnGeld;

    // variabelen waarden geven
    ZakGeld = 4;
    SpaarGeld = 12;
    AlMijnGeld = ZakGeld + SpaarGeld;

    // variabelen weergeven op de seriële monitor
    Serial.print("Geld op zak = ");
    Serial.println(ZakGeld);

    Serial.print("Spaargeld = ");
    Serial.println(SpaarGeld);

    Serial.print("Al mijn Geld = ");
    Serial.println(AlMijnGeld);
}

void loop()
{
    // even leeg laten
}
```

Bekijk volgend programma:

Wat verwacht je te zien op de seriële monitor ?

Oefening 6

```
#include <E4SBoard.h>
E4SClass E4SBoard;

int D = 13;
float DD = 13;
int E;
float F;
float G;

void setup()
{
  E4SBoard.Begin();
  // snelheid voor de seriële monitor
  Serial.begin(9600);

  E = D / 2; //E is D gedeeld door twee
  F = D / 2; //F is D gedeeld door twee
  G = DD / 2; //G is DD gedeeld door twee

  Serial.print("E = ");
  Serial.println(E);
  Serial.print("F = ");
  Serial.println(F);
  Serial.print("G = ");
  Serial.println(G);
}

void loop()
{
  // even leeg laten
}
```

Bekijk volgende programma:

Wat verwacht je te zien op de seriële monitor ? Verklaar je antwoord !

Schrijf dit programma en controleer je antwoord.

Operators [Module 6]

Nu dat we geleerd hebben over “Datatypes”, “Variabelen” en hoe we ze kunnen weergeven, wordt het tijd om eens te gaan kijken, naar wat men noemt “Operators”. “Operator” is het Engelse woord voor “Bewerker”. Dit is een beetje een rare vertaling, maar het geeft toch wel weer wat ze doen.

Een operator is een symbool dat iets doet met data.

Rekenkundige Operators.

Deze **operators** kunnen gebruikt worden om berekeningen te maken. De meeste van deze symbolen ken je wel. Sommige van deze symbolen zijn net even anders op de computer.

“Delen door” ziet er normaal als volgt uit: $12 : 4 = 3$

In onze programmeertaal **C** schrijf je dit als volgt: $12 / 4 = 3$

Ook **vermenigvuldigen** ziet er een beetje anders uit.

Normaal schrijven we: $4 \times 3 = 12$

In onze programmeertaal **C** schrijven we dit als volgt: $4 * 3 = 12$

Symbool	Doel
=	Toewijzen
+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen
%	Modulo

De vreemde eend in de bijt kan de “**modulo**” operator zijn.

Stel dat we 5 delen door 2 is 2,5. We nemen het gedeelte na de komma, dit is 0,5.

We vermenigvuldigen dit met 2. Daaruit volgt $0,5 \times 2 = 1$.

Dus ... de “**modulo**” van 5 gedeeld door 2 is: $5 \% 2 = 1$

Nog een paar voorbeelden:

$10 \% 5 = 0$ omdat $(10 : 5 = 2,0$ en het gedeelte na de komma $0 \times 5 = 0$)

$33 \% 7 = 5$ omdat $(33 : 7 = 4,71429$ en het gedeelte na de komma $0,71429 \times 7 = 5$)

Een andere rare jongen is het “=” teken.

Dit wil niet zeggen “gelijk aan” maar “toegewezen aan”.

Voorbeeld :

$b = b + 1$

Rekenkundig zou dit niet kunnen.

Maar we willen eigenlijk zeggen dat we de waarde van $b + 1$ toewijzen aan de variabele b

Vergelijking Operators

Symbol	Doel
==	is gelijk aan
!=	is niet gelijk aan
<	is kleiner dan
>	is groter dan
<=	is kleiner dan of gelijk aan
>=	is groter dan of gelijk aan

Als we bezig zijn met programmeren in Arduino, dan zul je snel merken dat het belangrijk is dat we beslissingen kunnen maken in een programma. vb. Het licht aansteken als het donker is. Hiervoor moeten we gaan vergelijken en daarvoor dienen deze groep operators.

Een vergelijking levert altijd een "true" (waar) of een "false" (onwaar) als antwoord. Dit is dus een boolean antwoord.

Voorbeeld :

5 > 2 (5 is groter dan 2) levert een "true" op.

5 < 2 (5 is kleiner dan 2) levert een "false" op.

Schrijf onderstaand programma, compileer het en stuur het door naar de ESP32. Vergeet niet je seriële monitor aan te zetten.

```
#include <E4SBoard.h>

E4SClass E4SBoard;

boolean A;

void setup()
{
  E4SBoard.Begin();
  Serial.begin(9600);

  A = 5 < 2;
  Serial.print("A = ");
  Serial.println(A);
}

void loop()
{
}
```

We kijken als vijf kleiner is dan twee.
Dit is dus onwaar (false).

We krijgen op de seriële monitor een nul te zien.
Sla het programma op als
"sketch_vergelijking_operator".

Meer hierover in Module 7

Boolean Operators

Boolean operators zijn operators die we **in de gewone taal** uitspreken als “**en**”, “**of**” en “**niet**”.

Symbol	Doel
&&	AND (en)
	OR (of)
!	NOT (niet)

AND (&&)

Laten we de AND operator uitleggen aan de hand van een voorbeeld.

Stel we hebben twee boolean waarden : “ZietErUitAlsEenAap” en “KlinktAlsEenAap”.

We gaan deze twee waarden gebruiken om te bepalen of we te maken hebben met een “Aap”, en daarvoor moeten beiden WAAR (true) zijn. Dus we hebben te maken met een Aap als ZietErUitAlsEenAap AND (en) KlinktAlsEenAap beiden waar (true) zijn.

Aap = ZietErUitAlsEenAap && KlinktAlsEenAap

Bekijken we eens de waarheidstabel

Waarde1	Waarde2	Resultaat van Waarde1 AND Waarde2
false	false	false
false	true	false
true	false	false
true	true	true

Schrijf onderstaand voorbeeldprogramma, compileer het en voer het uit.

Bekijk het resultaat op de seriële monitor.

Sla het programma op als "sketch_boolean_and_operator".

```
#include <E4SBoard.h>
E4SClass E4SBoard;

boolean ZietErUitAlsEenAap = true; //Waarde1
boolean KlinktAlsEenAap = true;    //Waarde2
boolean Aap;

void setup()
{
    E4SBoard.Begin();
    Serial.begin(9600);

    Aap = ZietErUitAlsEenAap && KlinktAlsEenAap;
    Serial.print("Het is een Aap = ");
    Serial.println(Aap); //Toon het resultaat van regel 9 op de seriële monitor
}

void loop()
{
}
```


OR (||)

Terug een voorbeeld om de OR operator uit te leggen :

We bekijken als voorbeeld de volgende twee boolean waarden : “DonkerBuiten” en “Na21Uur”

Om te bekijken of de buitenverlichting aan moet (variabele : “LichtenAan”) moet minstens één van deze twee variabelen waar (true) zijn. Dus LichtenAan als het DonkerBuiten OR (of) Na21Uur is.

LichtenAan = DonkerBuiten || Na21Uur

Bekijken we nu de waarheidstabel voor de OR operator

Waarde1	Waarde2	Resultaat van Waarde1 AND Waarde2
false	false	false
false	true	true
true	false	true
true	true	true

Schrijf onderstaand voorbeeldprogramma, compileer het en voer het uit.

Bekijk het resultaat op de seriële monitor.

Sla het programma op als “*sketch_boolean_or_operator*”.

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  boolean DonkerBuiten = true; //Waarde1
5  boolean Na21Uur = false;     //Waarde2
6  boolean LichtenAan;
7
8  void setup()
9  {
10   E4SBoard.Begin();
11   Serial.begin(9600);
12
13   LichtenAan = DonkerBuiten || Na21Uur;
14   Serial.print("Lichten gaan aan = ");
15   Serial.println(LichtenAan);
16 }
17
18 void loop()
19 {
20 }
```

Opdracht :

Wijzig het programma zodat de onboard LED brandt als de lichten aangaan in je programma en uitgaat als de lichten uitgaan in je programma. Sla het programma op

NOT (!)

De boolean “NOT” (niet) operator draait een boolean om naar zijn tegengestelde waarde.
Dus een “true” wordt “false” en een “false” wordt “true”

Voorbeeld :

Als “LichtenAan” waar is (true), dan is NOT (niet) “LichtenAan” dus onwaar (false).

Maar als “LichtenAan” niet waar (false) is, dan is NOT “LichtenAan” dus waar (true).

Dus: NOT true = false, en NOT false = true.

!true = false

!false = true

Waarheidstabel

Waarde1	Resultaat van NOT Waarde1
true	false
false	true

Samengestelde (Compound) Operators.

Compound Operators, ook wel “**Samengestelde**” operators genoemd, worden vaak gebruikt om minder te hoeven typen als we een programma schrijven. **Dit wordt vaak gebruikt !**

Een **nadeel** is dat het programma **soms moeilijker leesbaar** is.

Meestal combineert een Compound Operator twee of meer operators in 1 stap.

Ik vermeld hieronder **de meest gebruikte compound operators**.

Symbool	Doel	Voorbeeld
++	Waarde verhogen	A++; //verhoog A met 1 en geef de oude waarde van A terug ++A; //verhoog A met 1 en geef de nieuwe waarde van A terug
--	Waarde verlagen	A--; // verlaag A met 1 en geef de oude waarde van A terug --A; // verlaag A met 1 en geef de nieuwe waarde van A terug
+=	Samengetrokken optelling	A += B; // hetzelfde als A = A + B;
-=	Samengetrokken aftrekking	A -= B; // hetzelfde als A = A - B;
*=	Samengetrokken vermenigvuldiging	A *= B; // hetzelfde als A = A * B;
/=	Samengetrokken deling	A /= B; // hetzelfde als A = A / B;
%=	Samengetrokken modulo	A %= B; // hetzelfde als A = A % B;

Schrijf onderstaand voorbeeld programma, compileer het en voer het uit.
Bekijk het resultaat op de seriële monitor.

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  int A = 0;
5
6  void setup()
7  {
8    E4SBoard.Begin();
9    Serial.begin(9600);
10
11   Serial.print("Setup: A = ");
12   Serial.println(A);
13 }
14
15 void loop()
16 {
17   A += 1; //hetzelfde als A = A + 1
18   delay(1000);
19
20   // print de waarden via de serial monitor
21   Serial.print ("Tellen in de loop() : A = ");
22   Serial.println(A);
23   delay(1000);
24 }
```

Als we nu regel 17 zouden veranderen naar het betere leesbare $A = A + 1$;
dan zullen we zien dat de compiler precies hetzelfde resultaat levert wat omvang betreft van het programma.

Beslissingen [Module 7]

In [Module 6] hebben we gekeken naar de vergelijking operators, waarmee we informatie (data) kunnen vergelijken. Dergelijke vergelijking geeft ons als antwoord een boolean, dus “true” (waar) of “false” (niet waar). We gaan dit gebruiken om in deze module beslissingen (decisions) te maken in ons programma.

Als simpel voorbeeld zou ons programma lichten aan of uit kunnen zetten, afhankelijk van het feit of het nu buiten licht of donker is. In ons programma moeten we dus eerst kijken of het buiten donker is.

Als dat zo is dan moeten we de code uitvoeren die de lampen gaan aanzetten.

Als het niet donker is, dan moeten we de code uitvoeren die de lampen uit zetten.

Dit zou je als volgt kunnen lezen : als ... anders ...

Dus als iets waar is, doe dan dit, als iets niet waar is, doe dan dat.

Het “als ... anders ...” vertaald zich in het Engels naar “if ... else ...”.

If ... else ...

Dit is een van de meest gebruikte manieren om beslissingen in jouw programma te maken.

Een stukje van jouw programma wordt alleen uitgevoerd als een bepaalde stelling waar is.

Dit heet een “**conditionele instructie**”.

De juiste notatie (syntax) is :

```
if ( conditie )  
  
{  
    // wat gedaan moet worden als de conditie waar (true) is  
}  
  
else  
  
{  
    // wat gedaan moet worden als de conditie niet waar (false) is  
}
```

We zien weer de zogenaamde code blokken die we eerder hebben gezien tussen de accolades.

Het “else” statement wordt aan het einde van een “if ... ” geplaatst. Het woord “else” betekent “anders”. Dus “als iets waar is, doe dan dit, anders doe dat” Als het niet nodig is dan hoeft “else” niet gebruikt te worden. Dat kan de notatie vereenvoudigen.

```
if ( conditie )  
  
{  
    // doe wat nodig is als de conditie klopt  
}
```

We openen het bestand “*sketch_bewerkingen_met_variabelen*” en passen het aan zoals het voorbeeld hieronder.

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  void setup()
5  {
6      E4SBoard.Begin();
7      // set the speed for the serial monitor:
8      Serial.begin(9600);
9
10     // define our variables
11     int ZakGeld;
12     int SpaarGeld;
13     int AlMijnGeld;
14
15     // assign the values
16     ZakGeld = 4;
17     SpaarGeld = 12;
18     AlMijnGeld = ZakGeld + SpaarGeld;
19
20     // print the values to the serial monitor
21     Serial.print("Geld op zak = ");
22     Serial.println(ZakGeld);
23
24     Serial.print("Spaargeld = ");
25     Serial.println(SpaarGeld);
26
27     Serial.print("Al mijn Geld = ");
28     Serial.println(AlMijnGeld);
29
30     if(AlMijnGeld<5)
31     {
32         Serial.println("Oh oh,... we hebben maar weinig geld!!");
33     }
34     else
35     {
36         Serial.println("Geen probleem, we hebben genoeg geld.");
37     }
38 }
39
40 void loop()
41 {
42 }
```

We gaan kijken hoeveel geld we op zak hebben en hoeveel we gespaard hebben.

Als we minder dan 5 euro hebben dan willen we een waarschuwing zien, zodat we weten dat we niet al te veel geld meer hebben.

Compileer het programma en stuur het naar de ESP32

Een deel van de code hebben we al eerder gezien toen we het hadden over variabelen.
Het beslissing-stuk vinden we vanaf regel 30 waar we gaan kijken of we nog genoeg geld hebben.
We gaan dus kijken of de variabele "AlMijnGeld" minder dan (<) 5 euro is.
Als we minder dan 5 euro hebben, dan willen we een waarschuwing zien "Oh oh, ... we hebben maar weinig geld!!". Mochten we meer dan 5 euro hebben, dan is alles OK en willen we de melding "Geen probleem, we hebben genoeg geld." zien.

Omdat de conditie "false" is ($4+12 = 16$ is groter dan 5) zal het "if" code blokje overgeslagen worden
en gaat de ESP32 meteen verder met het "else" stuk.

Geld op zak = 4
Spaargeld = 12
Al mijn Geld = 16
Geen probleem, we hebben genoeg geld.

Laten we nu regel 17 veranderen en ons spaargeld op nul zetten : `int SpaarGeld = 0;`

☐ Compileer het programma opnieuw en stuur het naar de ESP32.

De conditie in regel 30 is nu WEL WAAR, dus de "if" blok wordt uitgevoerd en het "else" blok wordt nu overgeslagen. De output geeft nu :

Geld op zak = 4
Spaargeld = 0
Al mijn Geld = 16
Oh oh, ... we hebben maar weinig geld!!

Pas het programma aan, zodanig dat we alleen een waarschuwing zien als het niet in orde is met ons geld. Als alles OK is, willen we niets zien.

Het "if ... else ..." statement heeft echter ook nog eens de mogelijkheid om meerdere condities samen te voegen.

Voorbeeld :

als **AlMijnGeld = 0** dan **paniek** anders als **AlMijnGeld < 5** dan **haal meer geld** anders **niks aan de hand**

We stapelen zeg maar meerdere "if ... else if ... else if ... else" op.
Jouw ESP32 zal mooi door alle condities lopen tot het een conditie vindt die waar is.
Hij voert dan de bijhorende code uit en verlaat dan de hele "if ... else ..." constructie !


```

1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  void setup()
5  {
6      E4SBoard.Begin();
7      // set the speed for the serial monitor:
8      Serial.begin(9600);
9
10     // define our variables
11     int ZakGeld;
12     int SpaarGeld;
13     int AlMijnGeld;
14
15     // assign the values
16     ZakGeld = 4;
17     SpaarGeld = 12;
18     AlMijnGeld = ZakGeld + SpaarGeld;
19
20     // print the values to the serial monitor
21     Serial.print("Geld op zak = ");
22     Serial.println(ZakGeld);
23
24     Serial.print("Spaargeld = ");
25     Serial.println(SpaarGeld);
26
27     Serial.print("Al mijn Geld = ");
28     Serial.println(AlMijnGeld);
29
30     if(AlMijnGeld==0)
31     {
32         Serial.println("PANIEK!!");
33     }
34     else if(AlMijnGeld<5)
35     {
36         Serial.println("Oh oh,... we hebben maar weinig geld!!");
37     }
38     else if(AlMijnGeld>10)
39     {
40         Serial.println("Woohoo - we zijn rijk!!");
41     }
42     else
43     {
44         Serial.println("Geen probleem, we hebben genoeg geld.");
45     }
46 }
47
48 void loop()
49 {
50     // leave empty for now
51 }

```

Laten we nu de regels 16 en 17 aanpassen.

ZakGeld = 0;
SpaarGeld = 0;

Dus AlMijnGeld zal dus nul gaan worden in ons programma.

Als we nu naar onze “if ...” statements gaan kijken zien we iets interessants wat een fout (**bug**) in ons programma kan veroorzaken.

AlMijnGeld = 0

Dus de eerste conditie SLAAGT, en het bericht “PANIEK!!” verschijnt. Echter, ... conditie 2 (AlMijnGeld kleiner dan 5) is ook WAAR, maar zal nooit uitgevoerd worden.

Als een conditie gevonden wordt die WAAR is (TRUE), dan wordt de rest van de “if ...”

gezien als “klaar” en zullen de andere conditie niet eens getest worden.

Als je dit niet zou weten, dan zou je ook de melding “Oh oh, ... we hebben maar weinig geld!!” verwachten.

Dit komt omdat we condities gemaakt hebben die elkaar overlappen. Daarmee bedoelen we dat 2 of meer condities WAAR kunnen zijn onder bepaalde omstandigheden, en dat kan soms niet gewenst zijn.

Wat moeten we doen om beide meldingen toch zichtbaar te maken ?

We zouden kunnen proberen om de “if ... ” instructies uit elkaar te halen en afzonderlijk op te zetten.

```
if(AlMijnGeld==0)
{
    serial.println("PANIEK!!");
}

if(AlMijnGeld<5)
{
    serial.println("Oh oh,... we hebben maar weinig geld!!");
}

if(AlMijnGeld>10)
{
    serial.println("Woohoo - we zijn rijk!!");
}
```

Dit werkt, maar ... omdat we in onze oorspronkelijke code een “else” hebben voor de overige scenario’s, lopen we tegen een probleem aan. Waar zouden we “else” dan moeten zetten ?

In ons oorspronkelijk programma zeiden we :

```
If AlMijnGeld == 0 Serial.println “paniek” else if AlMijnGeld < 5 Serial.println “Oh Oh”
else if AlMijnGeld > 10 Serial.println “Woohoo”, else Serial.println “Geen probleem”
```

Door het uit elkaar trekken van de “if ... ”, werkte alles behalve wat er na het laatste “else” statement gebeurt.

```
If AlMijnGeld == 0 Serial.println “Paniek”
if AlMijnGeld < 5 Serial.println “Oh Oh”
if AlMijnGeld > 10 Serial.println “Woohoo”
```

We kunnen bij geen enkele van die condities een “else” toevoegen, anders zou deze uitgevoerd kunnen worden op ongewenste momenten.

```
If AlMijnGeld == 0 Serial.println “Paniek” else Serial.println “Geen probleem”
if AlMijnGeld < 5 Serial.println “Oh Oh” else Serial.println “Geen probleem”
if AlMijnGeld > 10 Serial.println “Woohoo” else Serial.println “Geen probleem”
```

Controleer maar eens bij alle drie de condities :

Stel dat AlMijnGeld 8 zou zijn. Wat zouden we dan te zien krijgen ? Klopt dat ?

Stel dat AlMijnGeld nul zou zijn. Wat zouden we dan te zien krijgen ? Klopt dat ?

Stel dat AlMijnGeld 4 zou zijn. Wat zouden we dan te zien krijgen ? Klopt dat ?

Hoe lossen we dit probleem op ? Laten we de impact van iedere conditie eens bekijken :

Conditie 1 zorgt voor "0" (nul).

Conditie 2 zorgt voor "0, 1, 2, 3, 4" (kleiner dan 5 d.w.z. dat 5 niet meetelt)

Conditie 3 zorgt voor "11, 12, 13, ... oneindig" (groter dan 10 dus 10 telt hier niet mee)

Zie je nu welke getallen er tussenuit vallen ? "5, 6, 7, 8, 9, 10"

Dus alles groter of gelijk aan ($\geq$) 5 **EN** kleiner of gelijk aan ($\leq$) 10

Hier kunnen de "Boolean Operators" een oplossing bieden

Zie tabel rechts

Symbool	Doel
&&	AND (en)
	OR (of)
!	NOT (niet)

Om de getallen op te vangen die buiten onze oorspronkelijke condities vielen, zouden we onze bijkomende condities als volgt moeten schrijven :

De eerste conditie is dat het getal groter of gelijk aan 5 moet zijn, dus : `AlMijnGeld >= 5`

De tweede conditie is dat het getal kleiner of gelijk aan 10 moet zijn, dus : `AlMijnGeld <= 10`

We zouden een "if ... "conditie in een "if ... " conditie kunnen steken.

Voorbeeld :

```
if(AlMijnGeld>=5)
{
    if(AlMijnGeld<=10)
    {
        // doe dit
    }
}
```

Dit zou werken, maar dergelijke code wordt snel erg rommelig ... Laten we eens naar de tabel met de "Boolean Operators" kijken. Om de getallen op te vangen die er tussenuit vallen moeten we een conditie hebben voor de getallen ≥ 5 en (and) ≤ 10 . In code kunnen we dit als volgt schrijven : `(AlMijnGeld >= 5) && (AlMijnGeld <= 10)`

Pas je programma nu aan zodat het werkt voor alle situaties.

We moeten ook de melding "Geen probleem, we hebben genoeg geld." krijgen als aan de bijkomende conditie voldaan is.

Switch ... case ...

Het “switch ... case ...” statement, is een soort super “if ... then ...” variant die niet vaak gebruikt wordt maar wel erg veel werk kan besparen. Het vervangt een groep “if ... then ...” statements in een bepaald aantal situaties.

Het “if” statement staat toe dat we condities kunnen gebruiken met behulp van vergelijkingen en booleans.

Het “switch” statement werkt alleen maar met “equal to” (“gelijk aan”) situaties.

Hoe werkt het ?

```
switch (variabele) {  
    case waarde1:  
        // doe dit als de variabele gelijk is aan waarde1  
        break;  
    case waarde2:  
        // doe dit als de variabele gelijk is aan waarde2  
        break;  
  
    ... // etc.  
  
    default:  
        // optioneel: doe dit als geen van de stellingen waar is  
        break;  
}
```

Het “switch” statement, neemt de huidige waarde van de “variabele” en vergelijkt het met de waarde die bij iedere “case” statement staat, om te bepalen of het de daaropvolgende code moet uitvoeren.

De optionele waarde “default” (Engels voor “standaard”) geeft aan welke code wordt uitgevoerd als geen van de stellingen waar zijn.

Je bent dus niet verplicht om “default” te gebruiken in een switch statement.

Laten we nu het volgende programma eens bekijken en wat verwachten we ?

De variabele “led” hebben we de waarde “2” toegekend.

Het switch statement gebruikt de variabele “led”.

“led” heeft dus de waarde “2” , daardoor wordt er gesprongen naar “case 2”.

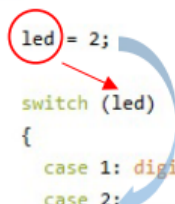
Bij “case 2” wordt nu de bijhorende code uitgevoerd.

Er zal “TEST” op de seriële monitor verschijnen en de LED gekoppeld aan GPIO17 zal oplichten.

```

1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  byte ledPin_1 = 16;
5  byte ledPin_2 = 17;
6  byte ledPin_3 = 18;
7  byte ledPin_4 = 19;
8  byte led;
9
10 void setup()
11 {
12     E4SBoard.Begin();
13
14     Serial.begin(9600);
15     pinMode(ledPin_1, OUTPUT);
16     pinMode(ledPin_2, OUTPUT);
17     pinMode(ledPin_3, OUTPUT);
18     pinMode(ledPin_4, OUTPUT);
19
20     digitalWrite(ledPin_1, LOW);
21     digitalWrite(ledPin_2, LOW);
22     digitalWrite(ledPin_3, LOW);
23     digitalWrite(ledPin_4, LOW);
24 }
25
26 void loop()
27 {
28
29     led = 2;
30
31     switch (led)
32     {
33         case 1: digitalWrite(ledPin_1, HIGH);
34         case 2:
35             Serial.println("TEST");
36             digitalWrite(ledPin_2, HIGH);
37         case 3: digitalWrite(ledPin_3, HIGH);
38         case 4: digitalWrite(ledPin_4, HIGH);
39     }
40 }

```



Schrijf dit programma, compileer en voer het programma uit.

Wat zien we ? Dat is toch niet hetgeen we verwacht hebben !

Inderdaad, de reden dat dit fout gaat is omdat het “switch” statement zal zoeken naar een “case” die true is en dan de opvolgende code zal uitvoeren. Omdat led = 2, worden alle statements vanaf “case 2:” uitgevoerd.

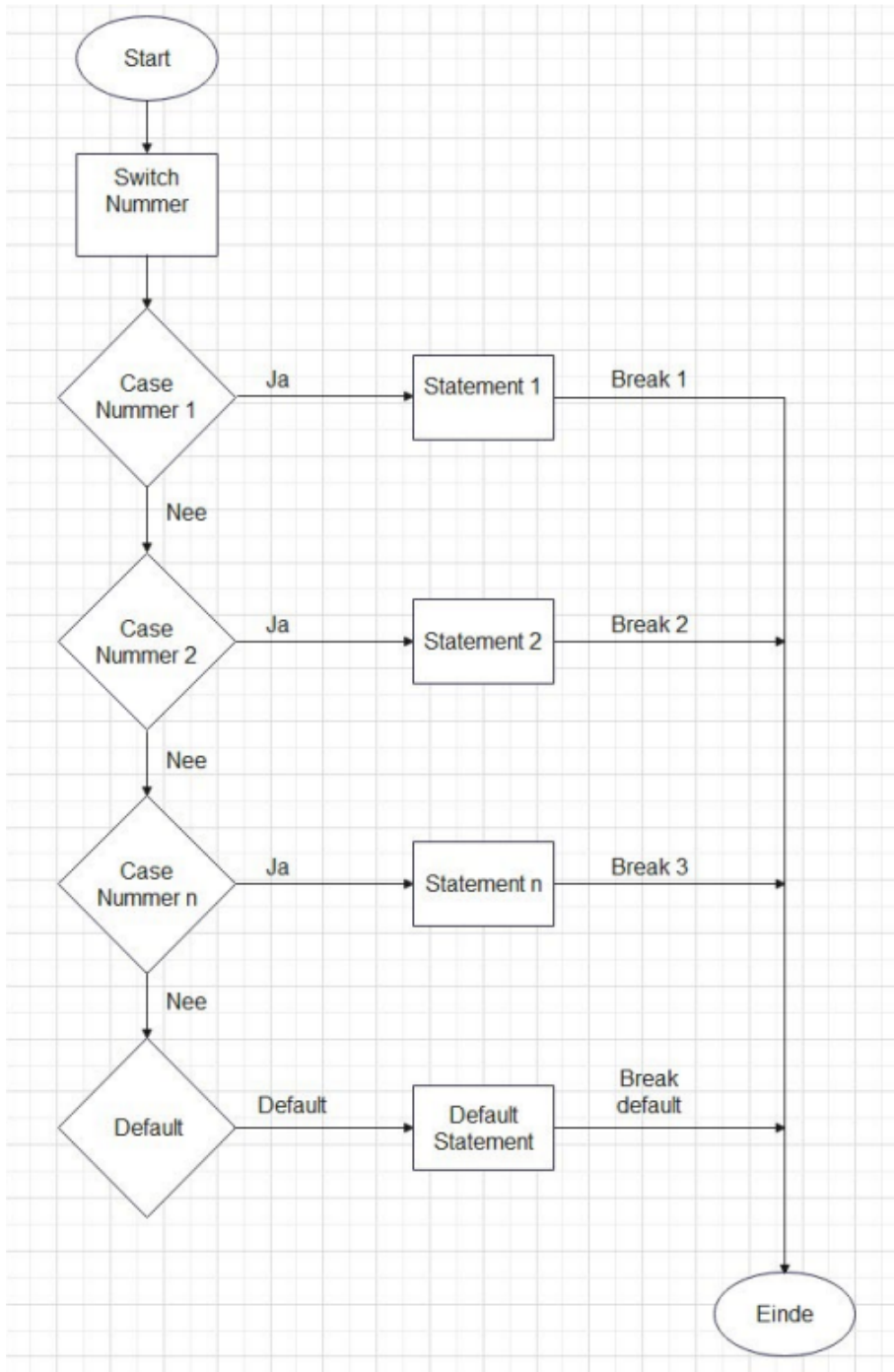
Dit kan zeer handig zijn in bepaalde situaties, maar in de meeste gevallen willen we dat de “switch” statement stopt nadat het hier in dit geval “case 2:” heeft gevonden. Om dat te bewerkstelligen, hebben we het “break” statement nodig aan het einde van iedere “case”.

Om **meer inzicht** te verwerven hoe ons programma moet werken kunnen we een **“flowchart”** of **stroomdiagram** tekenen.

FLOWCHART

Een flowchart is een schematische voorstelling van een aaneenschakeling van stappen waarbij instructies (statements) worden uitgevoerd die bijdragen tot het maken van een programma. In de module **“verdieping 7”** maken we kennis met de **symbolen en hun betekenis** die we in een flowchart kunnen gebruiken.

Hiernaast zie je een flowchart van ons “switch case” programma.




```

1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  byte ledPin_1 = 16;
5  byte ledPin_2 = 17;
6  byte ledPin_3 = 18;
7  byte ledPin_4 = 19;
8  byte led;
9
10 void setup()
11 {
12     E4SBoard.Begin();
13
14     Serial.begin(9600);
15     pinMode(ledPin_1, OUTPUT);
16     pinMode(ledPin_2, OUTPUT);
17     pinMode(ledPin_3, OUTPUT);
18     pinMode(ledPin_4, OUTPUT);
19
20     digitalWrite(ledPin_1, LOW);
21     digitalWrite(ledPin_2, LOW);
22     digitalWrite(ledPin_3, LOW);
23     digitalWrite(ledPin_4, LOW);
24 }
25
26 void loop()
27 {
28
29     led = 2;
30
31     switch (led)
32     {
33     case 1: digitalWrite(ledPin_1, HIGH);
34             break;
35     case 2:
36         Serial.println("TEST");
37         digitalWrite(ledPin_2, HIGH);
38         break;
39     case 3: digitalWrite(ledPin_3, HIGH);
40             break;
41     case 4: digitalWrite(ledPin_4, HIGH);
42             break;
43     }
44 }

```

Verbeter nu het programma zoals hiernaast,

compileer het en sla het op onder de naam :“sketch_switch_case”

Lussen [Module 8]

FOR - loop

Schrijf een programma om de LED's verbonden met GPIO16; GPIO17; GPIO18 en GPIO19 aan te schakelen. Test je programma uit. We moeten dus voor iedere LED dezelfde instructie herhalen ... Stel dat we duizend LED's zouden moeten aanschakelen ... Dat zou een hoop werk zijn !! Dit kan ook veel korter door gebruik te maken van een FOR-lus. De **FOR-lus is handig als er voor een aantal zaken hetzelfde moet gebeuren terwijl daarbij slechts één variabele verandert**

Bekijken we het volgende programma :

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  byte Led;
5
6  void setup()
7  {
8      E4SBoard.Begin();
9
10     for (Led = 16; Led < 20; Led++)
11     {
12         pinMode(Led, OUTPUT);
13     }
14     for (Led = 16; Led < 20; Led++)
15     {
16         digitalWrite(Led, LOW);
17     }
18 }
19
20 void loop()
21 {
22     for (Led = 16; Led < 20; Led++)
23     {
24         digitalWrite(Led, HIGH);
25     }
26 }
```

Dit ziet er ingewikkelder uit dan het is. Eerst wordt de waarde van de variabele Led op 16 gezet. Dan volgt een test of de waarde van de variabele Led kleiner is dan 20. De eerste keer is dat natuurlijk zo want de waarde van de variabele Led is dan 16. Vervolgens wordt de code tussen de accolades uitgevoerd. Led wordt als OUTPUT gedefinieerd. Vervolgens wordt de waarde van de variabele Led met 1 verhoogd (Led++) en wordt de lus herhaald.

Dit gaat zo door totdat Led de waarde 19 heeft. (Led < 20) De waarden van 16 tot en met 19 worden dus als OUTPUT gedefinieerd in de deze eerste FOR-loop. Op dezelfde manier gaan we de LED's met GPIO nummer 16 tot en met GPIO nummer 19 uit zetten in de setup en vervolgens aan zetten in de loop.

LET OP ! Sluit de FOR-regel niet af met een puntkomma !

ACTIE

- ☐ Schrijf een programma om de LED's verbonden met GPIO16; GPIO17; GPIO18 en GPIO19 aan te schakelen met behulp van enkele FOR-lussen. Gebruik in de FOR-lussen de instructie Led-- in plaats van Led++
- ☐ Test je programma uit.
- ☐ Verander je programma zodanig dat de LED's aan en uit knipperen met een tussentijd van 1 seconde.
- ☐ Test je programma uit.
- ☐ Schrijf een programma om de LED's verbonden met GPIO16; GPIO17; GPIO18 en GPIO19 één voor één aan te schakelen (looplicht van links naar rechts) met behulp van enkele FOR-lussen.
- ☐ Test je programma uit.
- ☐ "Knight Rider" is een film en televisieserie uit de jaren 1980, waarin alles draait om KITT een zwarte Pontiac. Op YouTube kun je hem zien:
<https://www.youtube.com/watch?v=iQwlrEdka6Q> Schrijf een programma die ervoor zorgt, dat er één LED op de ESP32 device heen en weer zal lopen zoals op de auto in de video.

WHILE – loop

Een while-loop controleert eerst of een vooropgestelde conditie waar is. Als dat het geval is, wordt het codeblok uitgevoerd, en start de loop opnieuw. Dit gaat zo verder tot de vooropgestelde conditie niet meer waar is en de loop verlaten wordt. Je zou dit kunnen zien als een eindeloos herhalende “if”. Het blijft zich herhalen, zolang de vooropgestelde conditie een “true” oplevert. Dit in tegenstelling tot het “if” statement die maar één keer doorlopen wordt. We zouden ons programma die we voorzien hadden met de FOR-loop als volgt kunnen herschrijven met een while-loop.

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  byte Led = 16;
5
6  void setup()
7  {
8      E4SBoard.Begin();
9
10     while (Led < 20)
11     {
12         pinMode(Led, OUTPUT);
13         Led++;
14     }
15     Led = 16;
16     while (Led < 20)
17     {
18         digitalWrite(Led, LOW);
19         Led++;
20     }
21
22     void loop()
23     {
24         Led = 16;
25         while (Led < 20)
26         {
27             digitalWrite(Led, HIGH);
28             Led++;
29         }
30     }
```

We starten in ons voorbeeld met het declareren van de variabele Led met een waarde van 16.

In dit programma is de vooropgestelde conditie Led < 20. Als deze conditie waar is (en deze conditie is waar, want we hebben Led de waarde 16 gegeven) dan wordt de code tussen de accolades uitgevoerd en de waarde van Led wordt dus met 1 verhoogd.

Het codeblok wordt steeds opnieuw uitgevoerd zolang de vooropgestelde conditie Led < 20 waar is. De waarden van 16 tot en met 19 worden dus als OUTPUT gedefinieerd in de deze eerste while-loop.

Op dezelfde manier gaan we de LEDs met GPIO nummer 16 tot en met GPIO nummer 19 uit zetten in de setup en vervolgens aanzetten in de loop.

“DO ... WHILE ...” LOOP

Een “do ... while ...” voert de instructie in het codeblok eerst uit om daarna pas te kijken of de conditie waar is.

Indien de conditie waar is, wordt de loop herhaald. Indien de conditie niet waar is, wordt de loop verlaten.

Bekijk onderstaand voorbeeld

```
1  #include <E4SBoard.h>
2  E4SClass E4SBoard;
3
4  byte Led = 16;
5
6  void setup()
7  {
8      E4SBoard.Begin();
9
10     do
11     {
12         pinMode(Led, OUTPUT);
13         Led ++;
14     }
15     while (Led < 20);
16
17     Led = 16;
18     do
19     {
20         digitalWrite(Led, LOW);
21         Led ++;
22     }
23     while (Led < 20);
24 }
25
26 void loop()
27 {
28     Led = 16;
29     do
30     {
31         digitalWrite(Led, HIGH);
32         Led++;
33     }
34     while (Led < 20);
35 }
```

LET OP : In de DO ...WHILE... loop volgt na de while conditie een puntkomma.

ACTIE

- ☐ Schrijf een programma om de LEDS verbonden met GPIO16; GPIO17; GPIO18 en GPIO19 aan te schakelen **met behulp van enkele WHILE-lussen**. Gebruik in je programma de instructie “Led = Led - 1;” in plaats van “Led = Led + 1;”
- ☐ Test je programma uit.
- ☐ Schrijf een programma om de LEDS verbonden met GPIO16; GPIO17; GPIO18 en GPIO19 aan te schakelen **met behulp van enkele DO ... WHILE-lussen**. Gebruik in je programma de instructie “Led = Led - 1;” in plaats van “Led = Led + 1;”
- ☐ Test je programma uit.

Digitale input en output [Module 9]

Bij de meeste toepassingen van de ESP32 draait het om twee dingen: je stopt er iets in (input) en je verwacht dat er iets uitkomt (output). Je drukt op een knop (input) en er gaat een lamp branden (output). Een temperatuur komt boven een bepaald niveau (input) en een relaiscontact (output) zorgt ervoor dat de verwarming uitgaat.

Een lichtsensor wordt onderbroken (input) en de draairichting van een motor keert om (output).

Met drukknoppen of sensoren op de ingangen en actuatoren (vb. led, relais, contactor ...) op de uitgangen kan je een controller zoals de ESP32 van alles laten doen.

Onze e-4S ESP32 device heeft 10 uitgangen, 4 ingangen die we rechtstreeks kunnen aansturen en 16 ingangen die we gaan aansturen via een multiplexer (elektronische 16-standen schakelaar).

Drukknop vs schakelaar

Dagelijks druk je tientallen malen op een drukknop of “push button”. De knop van de deurbel, de zap-knop op de afstandsbediening van de TV, de lettertoetsen op je toetsenbord enz... **Een drukknop heeft een terugverende knop en twee schakelstanden.**



Symbol :



Knop	Stand
Indrukken	De drukknop maakt contact, hij is ‘gesloten’. (aansluitingen zijn doorverbonden, er kan stroom vloeien)
Loslaten	Het contact is verbroken, de drukknop staat ‘open’. (aansluitingen zijn niet doorverbonden, er kan geen stroom vloeien)

Symbol :

Schakelaars daarentegen blijven in een gekozen positie staan: druk je eenmaal, dan maakt de schakelaar contact, ook als je de knop loslaat. Druk je nogmaals, dan wordt het contact verbroken enz... Het symbool voor een schakelaar kun je hiernaast zien.

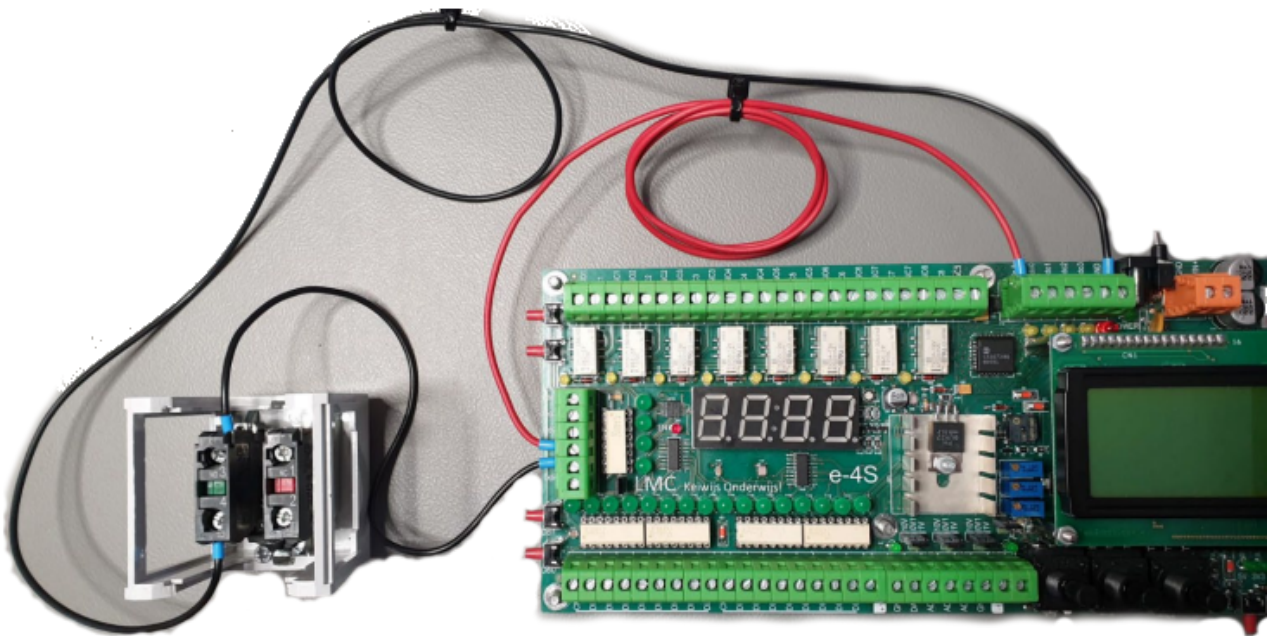


LED in- en uitschakelen.

In dit praktisch voorbeeld bedien je met een drukknop de “inboard” LED van de ESP 32 device. De LED brandt zolang je de knop ingedrukt houdt. In onderstaand voorbeeld gaan we een ingang gebruiken die we rechtstreeks kunnen aansturen. We zullen die directe ingang met een externe drukknop aansturen.

TIJD VOOR ACTIE

We gebruiken het “Normal Open” (NO) contact van de drukknop. Verbind aansluiting Vo van de ESP32 device met CM+ van de ESP32 device. **Verbind GND van de ESP32 device met klem 3 van het NO contact.** Verbind klem 4 van het NO contact met DDin1. Deze input is via een elektronische interface verbonden met GPIO32.



Schrijf het programma die op volgende bladzijde vermeld staat in de IDE en sla het op onder de naam “`sketch_directe_input_drukknop`”

```

1  /*Gebruik van directe input.
2   Onboard LED op GPIO2 gaat aan als op de drukknop wordt gedrukt*/
3
4  #include <E4SBoard.h>
5  E4SClass E4SBoard;
6
7  const byte buttonPin = 32;           //GPIO32 voor drukknop
8  const byte ledPin = 2;               //GPIO2 voor onboard LED
9  bool buttonState = 0;               //Status van de drukknop
10
11 void setup()
12 {
13   E4SBoard.Begin();
14
15   Serial.begin(9600);                //Stel de seriële monitor in
16   pinMode(ledPin, OUTPUT);           //Initialiseer ledPin als output
17   pinMode(buttonPin, INPUT);         //Initialiseer buttonPin als input
18 }
19
20 void loop()
21 {
22   //Lees de status van de drukknop (LOW/HIGH)
23   buttonState = digitalRead(buttonPin);
24
25   //Toon de status van de drukknop in de seriële monitor
26   delay(1000);
27
28   Serial.println(buttonState);
29
30   //Controleer of de drukknop wordt ingedrukt,
31   //dus check of de status van de drukknop hoog (HIGH)is :
32
33   if (buttonState == HIGH)
34   {
35     digitalWrite(ledPin, HIGH);      //Zo ja, zet dan de onboard LED aan
36   }
37   else //Zo nee, (anders) de drukknop is niet ingedrukt dus LOW)
38   {
39     digitalWrite(ledPin, LOW);       //Zet de onboard LED uit
40   }
41 }

```

We overlopen eens de verschillende functies en waarden (constanten en variabelen) die we gebruiken in ons programma :

const

In regel 7 en 8 worden twee constanten gedefinieerd. De eigenschap van een constante is dat hij niet verandert gedurende de uitvoering van de sketch. Het is een vaste waarde die alleen gelezen kan worden. Zo heeft de constante 'buttonPin' de waarde 32, en de constante 'ledPin' de waarde 2. Beide constanten zijn van het type byte, dat wil zeggen een geheel getal zonder komma.

```
const byte buttonPin = 15;  
const byte ledPin = 2;
```

bool

In regel 9 wordt een variabele gedefinieerd : 'buttonState'
Deze krijgt de waarde 0 toegewezen. Deze variabele is van het type bool, dat wil zeggen dat deze variabele maar twee waarden kan aannemen "0" of "1".
`bool buttonState = 0;`

pinMode

In regel 16 wordt aangegeven dat ledPin een uitgang (OUTPUT) is. Omdat eerder aan ledPin de waarde 2 is toegekend, betekent dit dus dat GPIO 2 als uitgang fungeert. In regel 17 wordt buttonPin aangeduid als ingang (INPUT). Omdat eerder aan buttonPin de waarde 32 is toegekend, betekent dit dus dat GPIO 32 als ingang fungeert.

void loop()

De lus (loop) is begonnen. Bij elke uitvoering van de lus wordt in regel 23 de huidige waarde van 'buttonPin' gelezen. Deze waarde wordt als variabele 'buttonState' in het geheugen opgeslagen.

```
buttonState = digitalRead(buttonPin);
```

if ... else

In regel 33 t.e.m. 40 van de lus wordt met een if...else-functie getest en besloten wat er moet gebeuren. De functie bestaat uit drie delen: vergelijking, {actie1}, {actie2}. Met het dubbele isgelyktekens "==" wordt een vergelijking gemaakt: in dit geval wordt de waarde van buttonState vergeleken met de waarde HIGH. Is de vergelijking waar (voorwaarde vervuld), dan wordt actie1 uitgevoerd, dus alles tussen de twee accolades.
Is de vergelijking niet waar (voorwaarde niet vervuld) , dan wordt actie2 uitgevoerd, dus alles tussen het tweede accolade paar.

```
if (buttonState == HIGH) {  
  digitalWrite(ledPin, HIGH);    //actie 1  
}  
else {  
  digitalWrite(ledPin, LOW);     //actie 2  
}
```

In regel 33 t.e.m. 40 staat dus : als (if) buttonState == HIGH, zet dan de onboard LED aan, en anders (else) zet de onboard LED uit.

OPGELET :

In een programmeertaal is er een groot verschil tussen “=” en “==”

Bekijk de volgende voorbeelden :

linkerLid = rechterLid

Hier zeggen we dat we de inhoud van de variabele “rechterLid” toekennen aan de variabele “linkerLid”

Voorbeeld :

A = A+1

Hier kunnen we zeggen dat A nu A+1 wordt. Dus “=” is hier niet zoals in de wiskunde “is gelijk aan”

Als we “==” gebruiken dan vertrekken we vanuit een vergelijking, vraagstelling, voorwaarde.

buttonState==HIGH *Hier vragen we ons af als “buttonState” gelijk is aan “HIGH”*

Behalve == zijn ook andere vergelijkingen mogelijk.

In volgende tabel zie je hoe die genoteerd worden en wat hun betekenis is.

Vergelijking	Betekenis
x == y	x is gelijk aan y
x != y	x is niet gelijk aan y
x < y	x is kleiner dan y
x > y	x is groter dan y
x <= y	x is gelijk aan of kleiner dan y
x >= y	x is gelijk aan of groter dan y

Stel dat er in je programma zou staan :

```
if (1) {  
  digitalWrite(ledPin, HIGH);    //actie 1  
}  
else {  
  digitalWrite(ledPin, LOW);     //actie 2  
}
```

Wat zal het programma doen ?

Als er tussen haakjes “1” staat, wordt actie 1 uitgevoerd.

Als er tussen de haakjes “0” zou staan wordt “else” uitgevoerd, dus actie 0 zou uitgevoerd worden.

Pas je programma aan en test het uit. *We stellen vast dat het programma geen voorwaarde check meer doet. Het programma controleert dus niet als buttonState gelijk is aan HIGH. Het programma voert gewoon direct actie 1 uit. Met de “else” dus actie 2 houdt het programma geen rekening meer. Dit is eigenlijk onzin want dan zou je evengoed kunnen de if ... else ... voorwaarde weglaten. Maar in plaats van “1” of “0” tussen die haakjes zou je ook een voorwaarde tussen de haakjes kunnen plaatsen. vb. A < B Dus als A < B waar is, dan komt dit overeen met een “1” tussen haakjes en als A < B onwaar is, dan komt dit overeen met een “0” tussen haakjes.*

Gestructureerd programmeren

Om ervoor te zorgen dat onze programma's overzichtelijk blijven, zullen we moeten gestructureerd programmeren ! We zullen dit illustreren aan de hand van een voorbeeldoefening :

Opgave :

Als een deur (Ddi0) en een raam (Ddi1) gesloten zijn brandt de verwarming (Ddo3). De verwarming gaat uit als de deur of het raam geopend worden.

Hoe gaan we te werk ?

We maken voor iedere input (deur, raam) en output (verwarming) een variabele aan en die variabelen gaan we koppelen aan een GPIO – nummer

Voorbeeld :

```
byte Deur = 33;           //deurcontact
byte Raam = 32;           //raamcontact
byte Verwarming = 19;     //verwarming
```

We maken voor iedere input (deur, raam) en output (verwarming) een variabele aan die de status of toestand weergeeft.

Deze status heeft in ons voorbeeld twee mogelijke toestanden.

Voor het raam en de deur is dit “open” of “dicht”.

Voor de verwarming is dit “aan” of “uit”

We zullen dus voor het type van deze variabele een “**bool**” gebruiken.

Voorbeeld :

```
bool DeurState;           //status van de deur
bool RaamState;           //status van het raam
bool VerwarmingState;     //status van de verwarming
```

In de setup() gaan we de variabelen die gekoppeld zijn aan een GPIO-nummer definiëren als input of output :

```
pinMode (Deur, INPUT);    //deurcontact is een input
pinMode (Raam, INPUT);    //raamcontact is een input
pinMode (Verwarming, OUTPUT); //verwarming is een output
```

We gaan ook onze output initialiseren :

```
digitalWrite (Verwarming, LOW); //zet verwarming uit
```


In de loop() gaan we onze inputs lezen :

We lezen de toestand van het deurcontact en wijzen dit toe aan de variabele DeurState.
We lezen de toestand van het raamcontact en wijzen dit toe aan de variabele RaamState.

```
DeurState = digitalRead (Deur);           //toestand deurcontact  
RaamState = digitalRead (Raam);           //toestand raamcontact
```

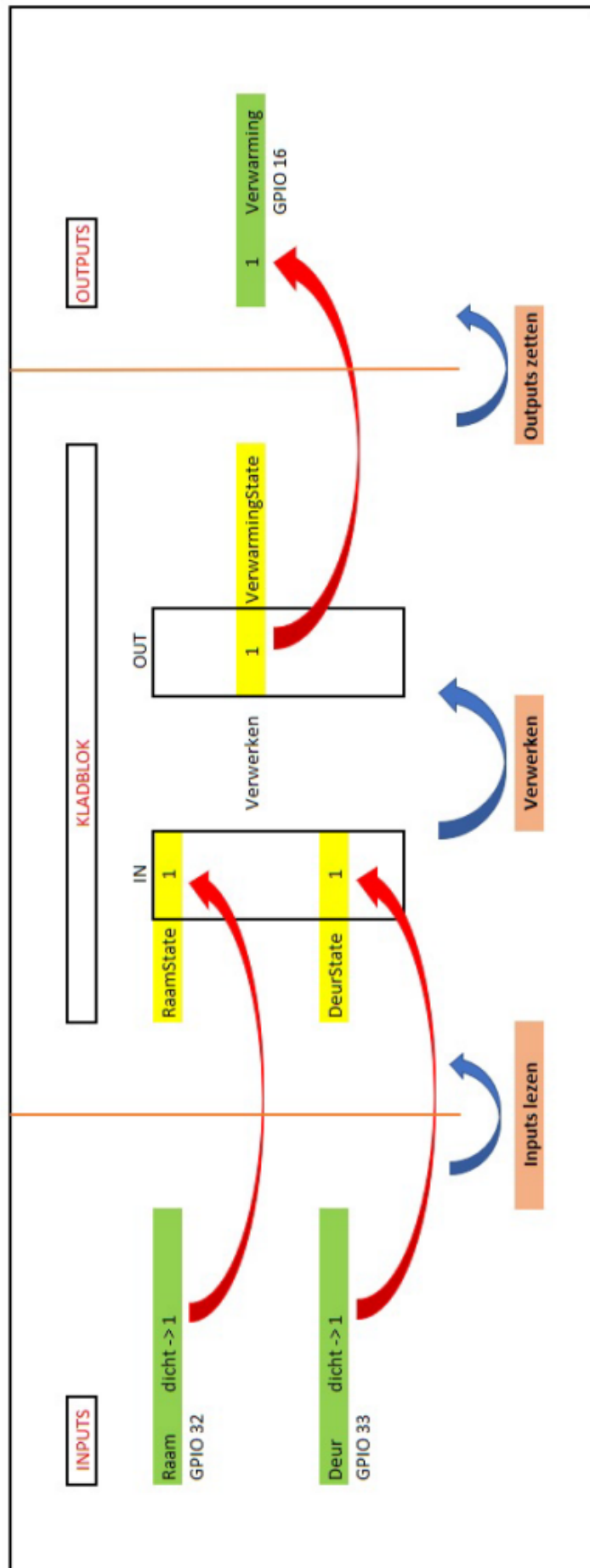
Vervolgens gaan we de status van de inputs verwerken
en bepalen we dus hoe de output hierop zou moeten reageren.

In dit stadium is de output nog niet geschakeld. We definiëren enkel hoe de toestand van de output (VerwarmingState) er zou moeten uitzien.
We zijn als het ware onze input informatie aan het verwerken in een kladblok.

```
if ((DeurState == 1) && (RaamState ==1)) //is de deur dicht EN is het raam dicht ?  
{  
    VerwarmingState = 1;                //zo ja, zet de toestand van de verwarming op "aan"  
}  
else  
{  
    VerwarmingState = 0;                //zo nee, zet de toestand van de verwarming op "uit"  
}
```

- Vervolgens gaan we de output zetten :

```
//de verwarming wordt geschakeld volgens de toestand van de variabele "VerwarmingState"  
  
digitalWrite (Verwarming, VerwarmingState);
```



```

1 // *****
2 // ***** bibliotheken *****
3 // *****
4 #include <E4SBoard.h>
5 E4SClass E4SBoard;
6
7 // *****
8 // ***** variabelen *****
9 // *****
10 byte Deur = 33;          // deurcontact
11 byte Raam = 32;          // raamcontact
12 byte Verwarming = 19;    // verwarming
13 bool DeurState;
14 bool RaamState;
15 bool VerwarmingState;
16
17 void setup() // voert deze routine 1 keer uit
18 {
19     E4SBoard.Begin();    // initialiseer ESP32-board
20     // *****
21     // ***** input/output definitie *****
22     //*****
23     pinMode(Deur, INPUT); // deurcontact is een input
24     pinMode(Raam, INPUT); // raamcontact is een input
25     pinMode(Verwarming, OUTPUT); // verwarming is een output
26
27     // *****
28     // ***** input/output initialisatie *****
29     //*****
30     digitalWrite(Verwarming, LOW); // zet verwarming uit
31 }
32
33 void loop() // voert deze routine steeds opnieuw uit
34 {
35     // *****
36     // ***** inputs lezen *****
37     // *****
38     DeurState = digitalRead(Deur); // de toestand van het deurcontact
39     RaamState = digitalRead(Raam); // de toestand van het raamcontact
40
41     // *****
42     // ***** verwerken *****
43     // *****
44     if ((DeurState == 1) && (RaamState == 1)) // is deur dicht EN is raam dicht?
45     {
46         VerwarmingState = 1; // ja? zet verwarmingstoestand aan
47     }
48     else
49     {
50         VerwarmingState = 0; // nee? zet verwarmingstoestand uit
51     }
52     // *****
53     // ***** outputs zetten *****
54     // *****
55     digitalWrite(Verwarming, VerwarmingState); // de verwarming schakelen volgens toestand
56 }

```

ACTIE

Programmeer onderstaande opgaven :

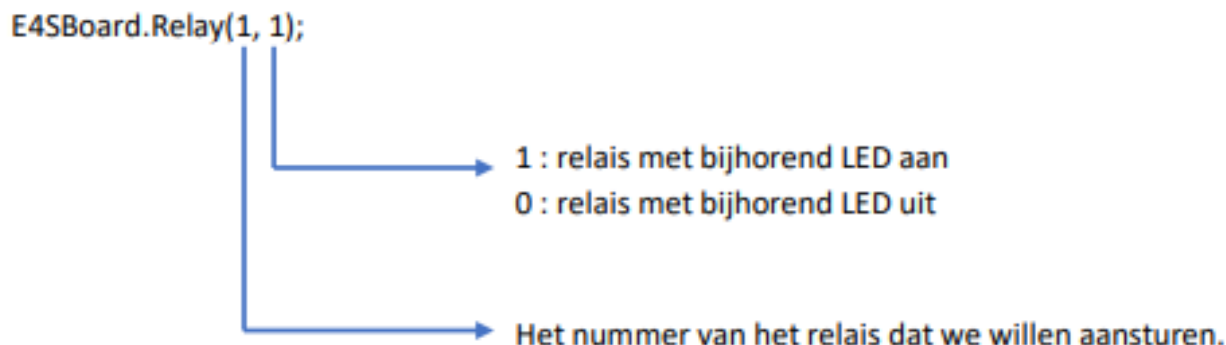
- ☐ Led “Ddo1” en “Ddo2” branden bij het sluiten van “Ddi2” en “Ddi3”.
- ☐ Het lampje van de frigo (Ddo0) brandt als de deur (Ddi0) van de frigo open staat. Als de frigo deur dicht is moet het lampje uit zijn.
- ☐ Als Ddi1 gesloten is brandt Ddo0. Als Ddi2 gesloten is brandt Ddo1. Als geen van beiden gesloten zijn brandt Ddo3.
- ☐ Als de poorten (Ddi0 en Ddi1) van een ferryboot niet volledig gesloten zijn, moet de kapitein daarvan verwittigd (Ddo1) worden in de stuurcabine. Bij het laden en lossen staan de poorten open en moet de bemanning het waarschuwingssysteem (Ddo1) kunnen uitschakelen.
- ☐ Ontwerp een schakeling waarbij het mogelijk is om met Ddi1 led Ddo1 aan te steken en met Ddi2 led Ddo1 te doven en led Ddo2 te laten branden. Vervolgens met Ddi3 led Ddo2 te doven en led Ddo3 aan te steken.

**KEEP
CALM
BECAUSE
WE CAN
DO THIS!**

Shift Register uitgangen

Tot nu toe hebben we enkel de 4 uitgangen gebruikt die we direct kunnen aansturen. Daar we gebruik maken van de bibliotheek "E4SBoard.h" kunnen we nu ook de overige 8 relais uitgangen gebruiken via het Shift Register.

We gebruiken de volgende instructie :



Programmavoorbeeld :

We willen het derde relais 5 maal laten aan en uitgaan met een tussentijd van 1 s.

```
#include <E4SBoard.h>

E4SClass E4SBoard;

void setup()
{
  for (int i = 0; i < 5; i++) // herhaal 5 maal
  {
    E4SBoard.Relay(3, 1);    // relais 3 geactiveerd
    delay(1000);
    E4SBoard.Relay(3, 0);    // relais 3 uitgeschakeld
    delay(1000);
  }
}

void loop()
{
}
```

ACTIE

Programmeer onderstaande opgaven

Schrijf je programma op een gestructureerde manier zoals aangeleerd in dit hoofdstuk.

- ☐ Als ik op drukknop Ddi1 druk wordt relais 1 geactiveerd. Als ik op drukknop Ddi2 druk wordt relais 2 geactiveerd. Met drukknop Ddi0 worden de relais die geactiveerd zijn uitgeschakeld.
- ☐ Met drukknop Ddi1 wordt relais 1 geactiveerd. Voor ik relais 2 kan activeren met drukknop Ddi2 moet ik eerst relais 1 uitschakelen met drukknop Ddi0. Als ik relais 1 terug wil activeren moet ik eerst relais 2 uitschakelen met drukknop Ddi0.
- ☐ Met drukknop Ddi1 wordt relais 1 geactiveerd. Met drukknop Ddi2 wordt relais 2 geactiveerd maar dan moet tevens relais 1 uitgeschakeld worden. Ik kan nu terug relais 1 terug activeren met drukknop Ddi1 maar dan moet tevens relais 2 uitgeschakeld worden. Relais 1 en relais 2 mogen nooit op hetzelfde moment geactiveerd zijn.

**KEEP
CALM
BECAUSE
WE CAN
DO THIS!**

Multiplexer ingangen

Door gebruik te maken van de bibliotheek "E4SBoard.h" kunnen we naast de 4 directe ingangen ook de 16 multiplexer ingangen gebruiken. Deze multiplexer ingangen moeten met externe drukknoppen of sensoren aangestuurd worden.

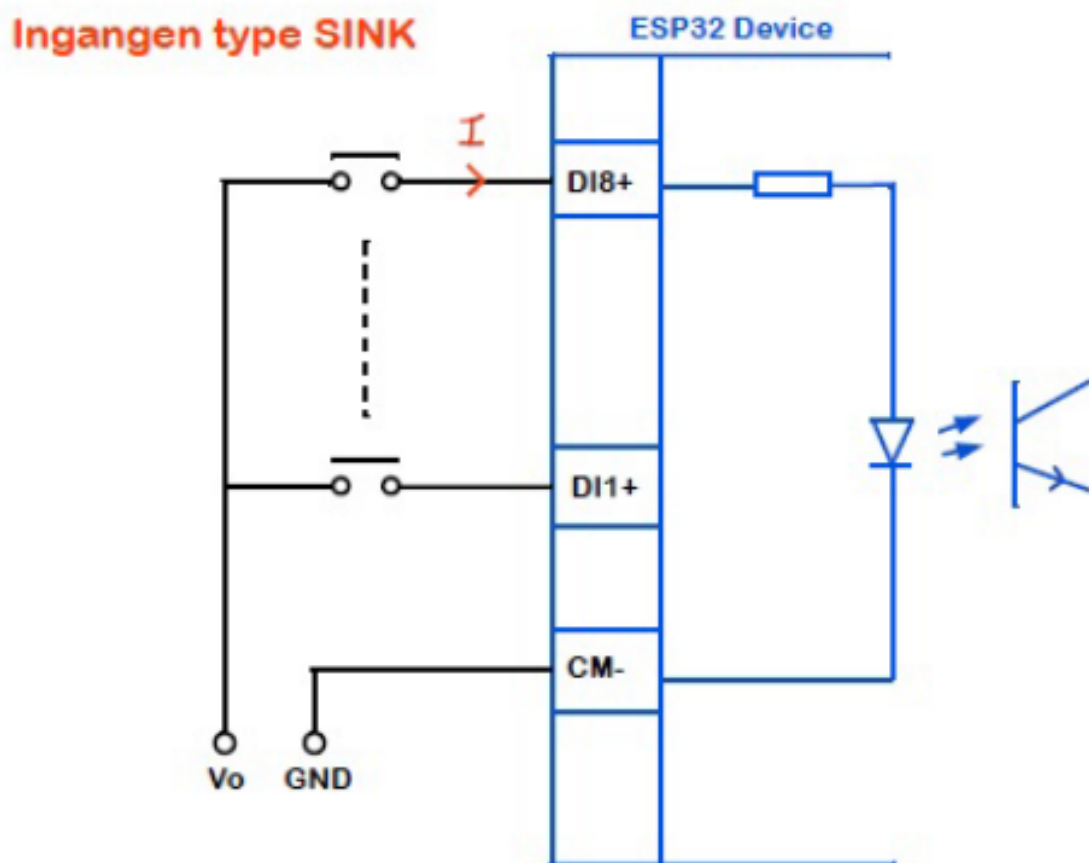
We onderscheiden twee groepen ingangen

Ingangen van het type SINK

De min-klem van de bron (GND) wordt verbonden met de gemeenschappelijke klem (CM-) van de multiplexeringang. De positieve kant van de bron (Vo) wordt dan via de verschillende externe drukknoppen of sensoren naar de respectievelijke ingangsklem (DI1+ DI8+) gebracht. Indien er op een externe drukknop gedrukt wordt, dan vloeit er een stroom in de ingangsklem, vandaar de naam SINK (opname van stroom).

Onze e-4S ESP32 device neemt via de drukknoppen de stroom op die uit de Vo komt.

Dit type van ingangen is geschikt voor het aansluiten van **PNP-sensoren**. Later hierover meer.



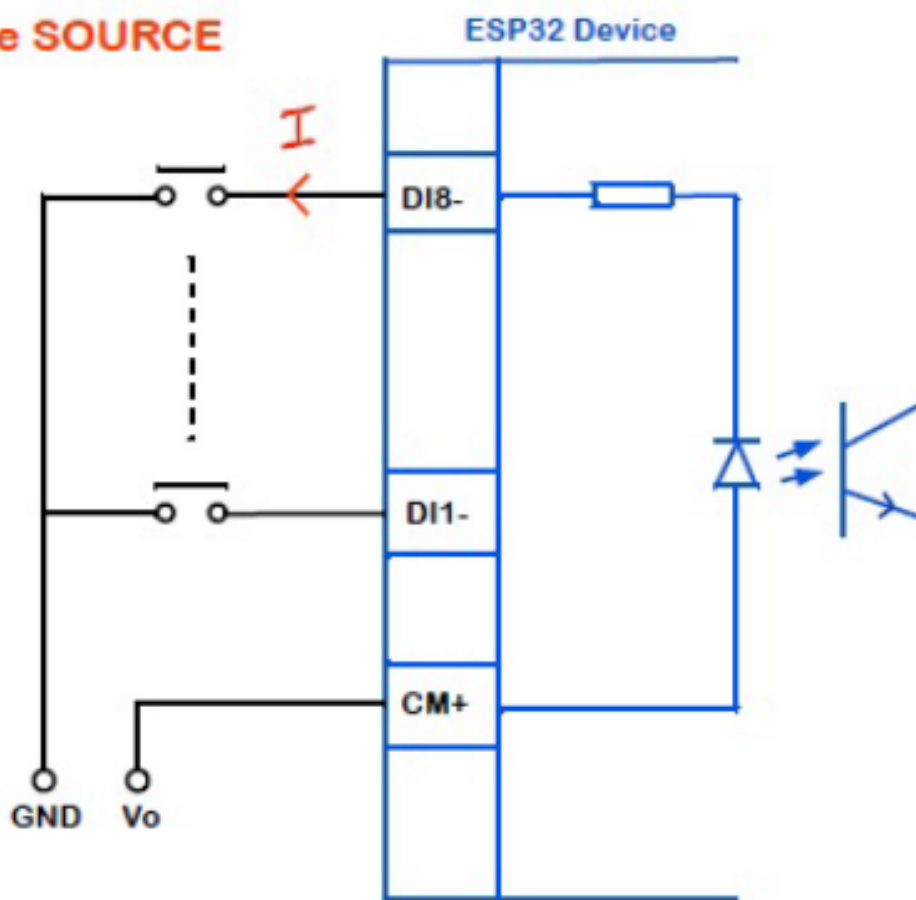
Ingangen van het type SOURCE

De plus-klem van de bron (V_o) wordt verbonden met de gemeenschappelijke klem ($CM+$) van de multiplexer ingang. De negatieve kant van de bron (GND) wordt dan via de verschillende externe drukknoppen of sensoren naar de respectievelijke ingangsklem ($DI1-$ $DI8-$) gebracht. Indien er op een externe drukknop gedruwd wordt, dan vloeit er stroom uit de ingangsklem, vandaar de naam SOURCE (afgifte van stroom).

Onze e-4S ESP32 device geeft via de drukknoppen stroom af naar de GND .

Dit type van ingangen is geschikt voor het aansluiten van **NPN-sensoren**. Later hierover meer.

Ingangen type SOURCE



*We verbinden een drukknop aan multiplexer ingang 5 van het type **SINK***

Hoe kunnen we nu de status van een multiplexer ingang lezen ?

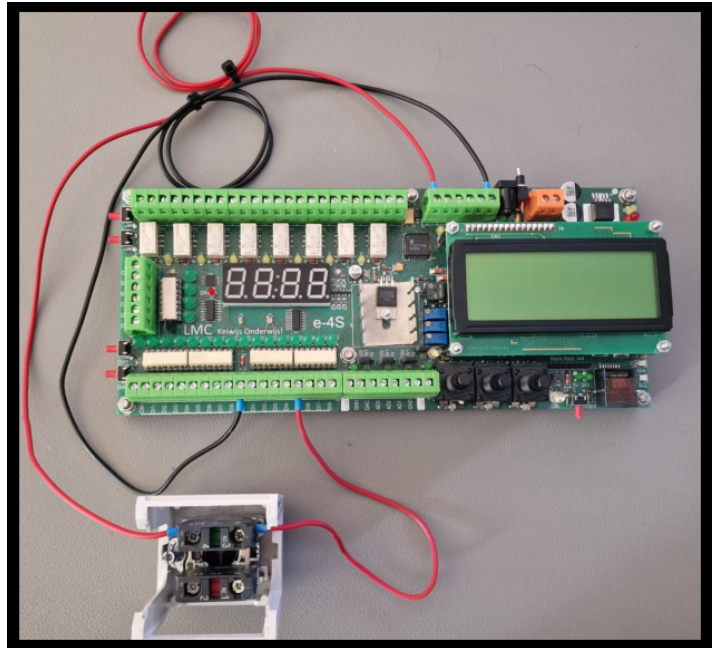
We willen bijvoorbeeld de status lezen van multiplexer ingang van het type SINK DI5

We gebruiken de volgende instructie :

`E4SBoard.DIn(4);`

Je denkt nu waarschijnlijk dat ik een typfoutje heb gemaakt door het getal “4” te schrijven. Neen dus !

Er zijn 8 “source” ingangen (DI1- tot en met DI8-) en 8 “sink” ingangen (DI1+ tot en met DI8+). We tellen van 0 tot en met 7. Dus DIn(4) in ons programma staat voor sink ingang DI5+



We kunnen de status van ingang DI5+ gelijk in een variabele steken.

`Ingang5State = E4SBoard.DIn(4);`

*Stel dat we de status wilden lezen van de multiplexer ingang van het type source DI7
Dan zouden we de volgende instructie gebruiken :*

`E4SBoard.Dip(6);`

Programmavoorbeeld

Als ik op een drukknop duw die verbonden is met de multiplexer sink ingang 5, dan moet shift register uitgang 8 geactiveerd worden.

```
#include <E4SBoard.h>
E4SClass E4SBoard;
//*****
//***** variabelen*****
//*****

bool Relay8State;
bool DIn5State;

void setup()
{
    //*****
    //*****input/output defenitie*****
    //*****

    DIn5State = 0;
}

void loop()
{
    //*****
    //***** inputs lezen*****
    //*****

    DIn5State = E4SBoard.DIn(4);

    //*****
    //*****verwerken*****
    //*****

    if (DIn5State == 1)
    {
        Relay8State = 1;
    }

    //*****
    //*****outputs Zetten*****
    //*****

    E4SBoard.Relay(8, Relay8State);
}
```

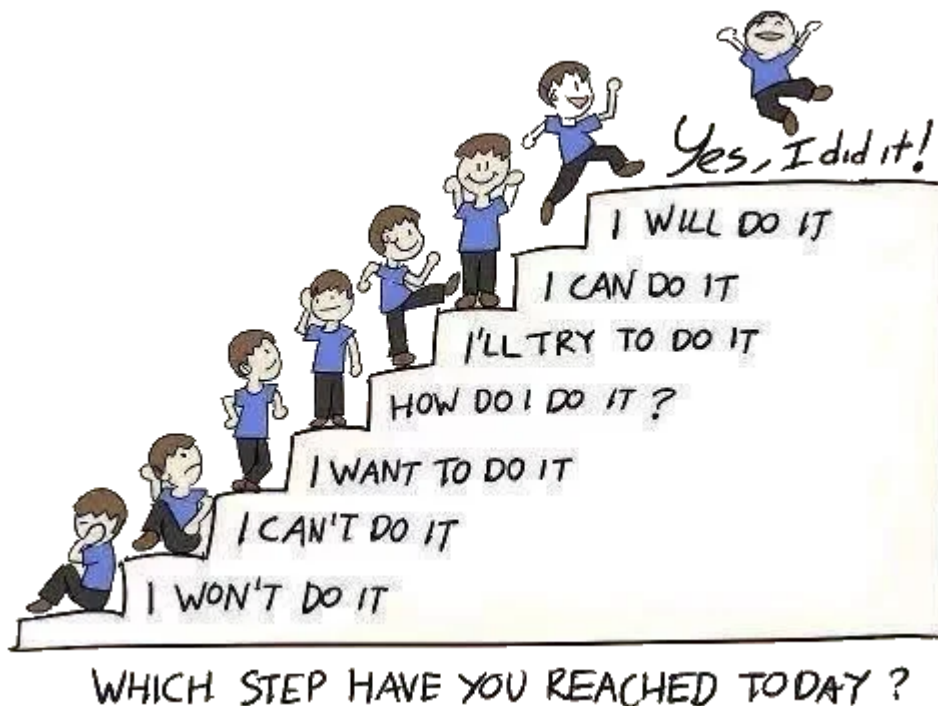
ACTIE

Programmeer onderstaande opgave :

Schrijf je programma op een gestructureerde manier zoals aangeleerd in dit hoofdstuk.

- ☐ Met drukknop Ddi1 wordt relais 1 en relais 2 geactiveerd. Met drukknop Ddi2 wordt relais 1 en relais 2 gedeactiveerd. Met externe drukknop 3 willen we de sink ingang DI4+ aansturen, als gevolg hiervan moet relais 3 geactiveerd worden. Met externe drukknop 4 willen we de sink ingang DI5+ aansturen, als gevolg hiervan moet relais 3 gedeactiveerd worden.

- ☐ Met drukknop Ddi1 wordt relais 1 gedeactiveerd en relais 2 geactiveerd. Met drukknop Ddi2 wordt relais 1 geactiveerd en relais 2 gedeactiveerd. Met drukknop Ddi3 wordt relais 1 en relais 2 gedeactiveerd. Met externe drukknop "Drukknop" willen we de sink ingang DI5+ aansturen, als gevolg hiervan moet relais 1 geactiveerd worden en relais 2 gedeactiveerd worden. Met externe drukknop "Sensor" willen we de sink ingang DI6+ aansturen, als gevolg hiervan moet relais 2 geactiveerd worden en relais 1 gedeactiveerd worden.



Proficiat u bent doorheen Module 9

Hopelijk heb je heel wat bijgeleerd en kun je nu beginnen aan het echte werk, laat deze cursus een leidraad zijn om uw verdere projecten uit te bouwen.

Machten [Verdieping 4]

Machten met gehele exponenten

Inleiding

Voor we de module “verdieping 5 en verdieping 6” bekijken is het handig om te weten wat we bedoelen met “machten”:

Machten zijn een veel voorkomend gegeven in de wiskunde en wij gebruiken dit ook vaak bij technische toepassingen.

Het lijkt in eerste instantie allemaal eenvoudig, maar er gebeuren nog veel fouten in het rekenen met machten. Daarom dat we eerst het begrip macht en zijn onderdelen bespreken

Regels voor het rekenen met machten toepassen bij het rekenen met getallen en met letters !

Begripsvorming :

Vooraleer we leren hoe we met machten kunnen rekenen, gaan we eerste een "macht" eens ontleden :

a^b (We zeggen: “a tot de macht b”)

“a” is het grondtal en “b” de exponent .

a^b in zijn geheel noemen we een macht !

Nu stellen we ons twee vragen :

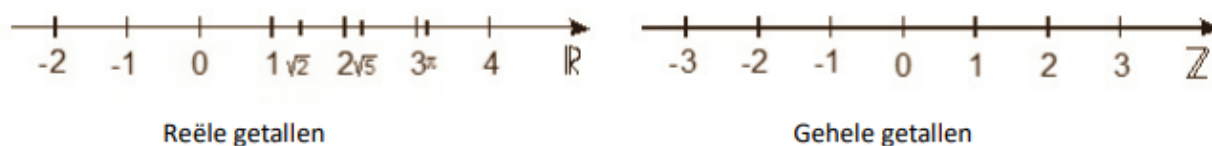
Kan elk getal dienen als grondtal van een macht ?

Kan elk getal exponent zijn van een macht ?

Om deze vragen te beantwoorden moet je een begrip hebben van de opbouw van de getallenverzamelingen.

We veronderstellen eerst dat “a” een reëel getal en “b” een geheel getal is.

Voorbeeld van reële en gehele getallen.



Voorbeelden van machten :

$$2^3 = 8 \quad (\text{We zeggen: 2 tot de 3^{de} macht of kortweg 2 tot de derde})$$

$$1^2 = 1$$

$$-2^4 = 16$$

$$0^3 = 0$$

$$3/4^1 = 3/4$$

Maar hoe kunnen we nu praktisch een macht berekenen ?

$$2^3 = 2 * 2 * 2 = 8$$

$$1^2 = 1 * 1 = 1$$

$$-2^4 = (-2) * (-2) * (-2) * (-2) = 16$$

$$0^3 = 0 * 0 * 0 = 0$$

$$a^b = a * a * a * \dots * a \quad (\text{dit doen we b-keer})$$

We maken één afspraak : $a^0 = 1$ (a is verschillend van nul)

Getalstelsels [Verdieping 5]

Ons algemeen gebruikt en vertrouwde rekenstelsel is slechts één van de vele mogelijkheden om getallen voor te stellen.

In het **decimale rekenstelsel** werken we met cijfers (symbolen) gaande van 0 tot 9.

De waarde van het cijfer hangt af van de rang waar het staat in een bepaald getal.

Het getal waarmee het dan moet vermenigvuldigd worden, noemen we het gewicht.

Voorbeeld :

$$\begin{aligned} 237 &= 2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0 \\ &= (2 * 100) + (3 * 10) + (7 * 1) \\ &= 200 + 30 + 7 \\ &= 237 \end{aligned}$$

In het decimale stelsel zijn de gewichten de opeenvolgende machten van 10.

10 is het GRONDGETAL.

Om aan te duiden dat we in het decimaal stelsel werken, kunnen we de getallen als volgt schrijven :

237_{10} -> zo wordt dit aangegeven in het vak wiskunde
`d237` -> zo wordt dit aangegeven in een programmeeromgeving

Het binaire rekenstelsel.

In het binaire rekenstelsel is het GRONDGETAL = 2 en zijn de gewichten de opeenvolgende machten van 2. Daarbij werkt men slechts met twee symbolen : "0" en "1" dat we "bit" noemen.

Voorbeeld :

$$\begin{aligned} 101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= (1*4) + (0*2) + (1*1) \\ &= 4 + 0 + 1 \\ &= 5_{10} \end{aligned}$$

In een programmeeromgeving kunnen we zeggen dat `0b101` gelijk is aan `d5`

De bit met de kleinste waarde wordt uiterst rechts geplaatst.

Men noemt hem de "Minst beduidende bit" (Least significant bit "LSB").

De uiterst linkse bit is dan de "Meest beduidende bit" (Most significant bit "MSB")

Gezien we in het binair rekenstelsel maar twee symbolen hebben, is het uitstekend geschikt om digitaal te tellen. Een digitaal systeem heeft immers maar twee toestanden "0" en "1".

Tellen met binaire getallen :

Decimaal getal	Binair getal			
	2^3	2^2	2^1	2^0
	MSB			LSB
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Voorbeeld uit de tabel :

$$13_{10} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$13_{10} = 1101_2$$

In een programmeeromgeving kunnen we zeggen dat **d13** gelijk is aan **0b1101**

Het hexadecimaal stelsel

Het GRONDGETAL is in dit stelsel 16 en de gewichten dus de opeenvolgende machten van 16. Het stelsel gebruikt 16 symbolen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E en F.

Voorbeeld :

$$\begin{aligned} A6E_{16} &= A \times 16^2 + 6 \times 16^1 + E \times 16^0 \\ &= (10 \times 256) + (6 \times 16) + (14 \times 1) \\ &= 2670_{10} \end{aligned}$$

In een programmeeromgeving kunnen we zeggen dat **0xA6E** gelijk is aan **d2670**

Dit getalstelsel is op zichzelf niet bruikbaar in de techniek, maar wordt veel gebruikt om binaire getallen verkort weer te geven.

Conversie van het binair naar het hexadecimaal stelsel :

Vb. $10110011100101_2 = (?)_{16}$

We delen het binaire getal in groepjes van 4 van rechts naar links.

Als het laatste groepje geen vier digits meer bevat, **dan plaatsen we nullen bij** tot we een groepje van vier hebben.

0010 1100 1110 0101

2	12	14	5	(<- we vervangen de binaire groepjes door decimale getallen
2	C	E	5	(<- we vervangen de decimale getallen groter dan 9 door hun hexadecimaal symbool)

$$10110011100101_2 = 2CE5_{16}$$

0b10110011100101 is gelijk aan **0x**2CE5

We kunnen ook een programma de conversies laten maken :

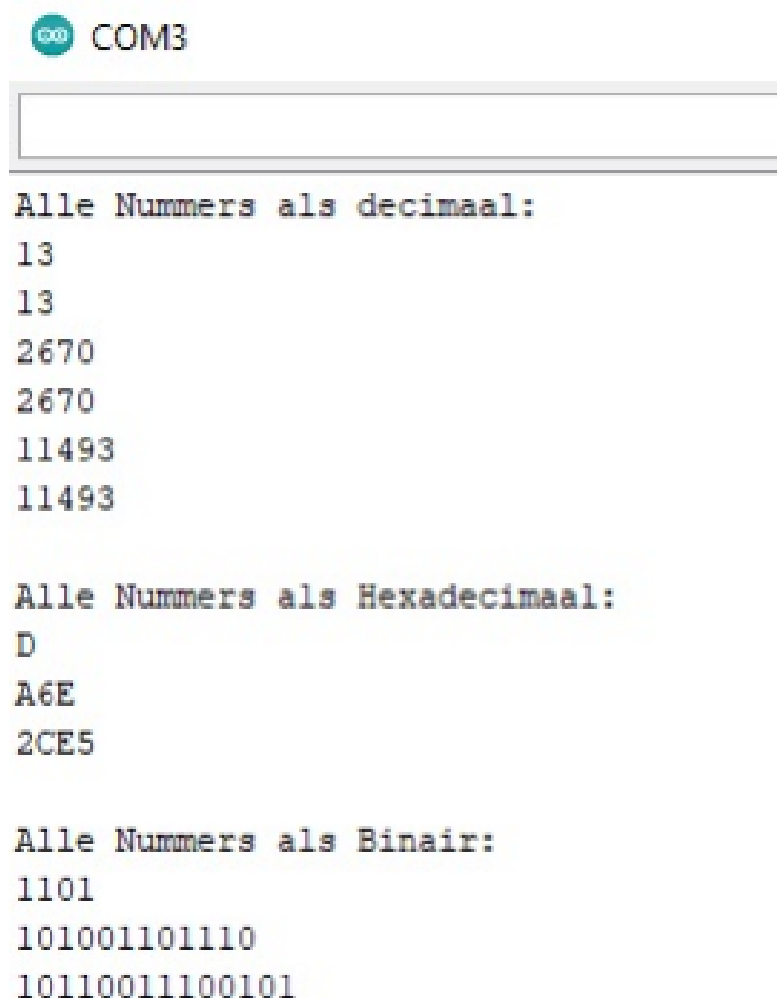
Schrijf het onderstaand programma en bekijk op de seriële monitor het resultaat.
(Sla het programma op onder de naam "sketch_conversie_getallen")

The image shows the Arduino IDE interface. At the top, there's a title bar with the sketch name 'sketch_conversie_getallen | Arduino 1.8.13'. Below it is a menu bar with 'Bestand', 'Bewerken', 'Schets', 'Hulpmiddelen', and 'Help'. A toolbar with icons for check, run, serial monitor, upload, and download is visible. The main area shows the sketch code for 'sketch_conversie_getallen'.

```
1 void setup() {
2   Serial.begin(9600);
3
4   int decNummer = 13;
5   int hexNummer = 0xA6E;
6   int binNummer = 0b10110011100101;
7
8   Serial.println("Alle Nummers als decimaal:");
9   Serial.println(decNummer);
10  Serial.println(decNummer, DEC);
11  Serial.println(hexNummer);
12  Serial.println(hexNummer, DEC);
13  Serial.println(binNummer);
14  Serial.println(binNummer, DEC);
15  Serial.println();
16  Serial.println("Alle Nummers als Hexadecimaal:");
17  Serial.println(decNummer, HEX);
18  Serial.println(hexNummer, HEX);
19  Serial.println(binNummer, HEX);
20  Serial.println();
21  Serial.println("Alle Nummers als Binair:");
22  Serial.println(decNummer, BIN);
23  Serial.println(hexNummer, BIN);
24  Serial.println(binNummer, BIN);
25 }
26
27 void loop() {
28   // leave empty for now
29 }
30
```

We gebruiken het statement "**Serial.Println();**" om een blanco lijn tussen te laten op de seriële monitor.

We zien het volgende resultaat op de seriële monitor :



```
COM3

Alle Nummers als decimaal:
13
13
2670
2670
11493
11493

Alle Nummers als Hexadecimaal:
D
A6E
2CE5

Alle Nummers als Binair:
1101
101001101110
10110011100101
```

Besluit :

Het is niet nodig om een 'd' voor een decimaal getal te vermelden.

Als we de output niet definiëren :

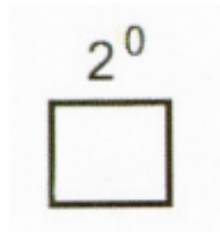
vb. `Serial.println(0xA6E);`

vb. `Serial.println(0b10110011100101);`

Dan krijgen we op de seriële output de decimale waarde te zien.

Bit en Byte [Verdieping 6]

De kleinste eenheid van informatie in een digitaal systeem noemt men de **bit** (Binary digIT).

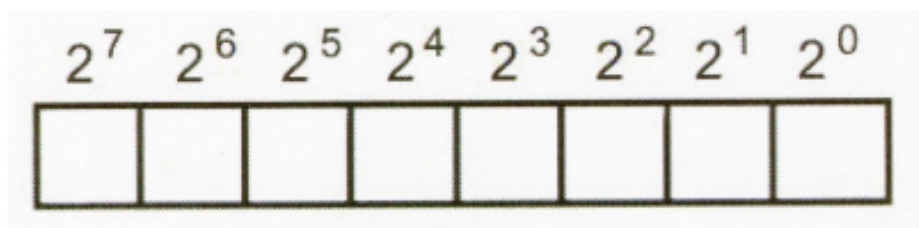


Een bit kan slechts twee waarden aannemen: "1" of "0".

In een binair systeem spreken we bijvoorbeeld van "aan" of "uit" of van "waar" of "vals"

De controller organiseert de te verwerken data in groepen,

Een groep van 8 bits noemt men een byte.



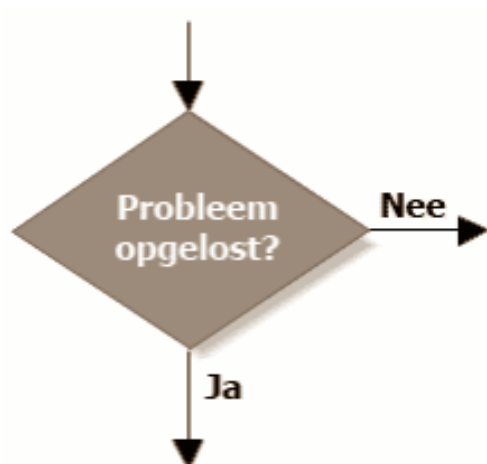
Flowchart [Verdieping 7]

BETEKENIS VAN FLOWCHART SYMBOLEN



Taak of activiteit

Een rechthoek staat voor een taak of activiteit die moet worden uitgevoerd.





Bevestiging e-mailen

Input/output

Een parallellogram staat voor het inkomen of uitgaan van informatie. Bijvoorbeeld het ontvangen of sturen van een e-mail.

Rapport

Document

Dit symbool staat voor het aanmaken van een document. Bijvoorbeeld een formulier of rapport.

Database

Database Op dit moment in het proces worden gegevens opgeslagen in een database.

Wachten op sein

Wachten Het proces wordt gepauzeerd.

Pagina 3

Naar een volgende pagina

EN POORT (and gate)

- Een EN poort is een schakeling met één uitgang en meerdere ingangen zodanig dat de uitgang 1 is, en slechts dan 1 is, als alle ingangen 1 zijn.
- De uitgang wordt 0 zodra één van de ingangen 0 wordt.

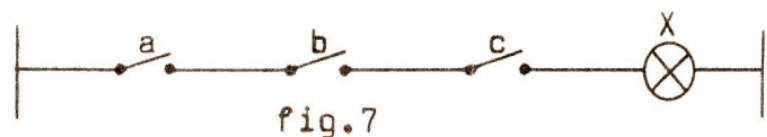
EN POORT (and gate)



Waarheidstabel

IN				UIT			
a	b	c	X		a	b	c X
0	0	0	0	of	L	L	L L
1	0	0	0		H	L	L L
0	1	0	0		L	H	L L
1	1	0	0		H	H	L L
0	0	1	0		L	L	H L
1	0	1	0		H	L	H L
0	1	1	0		L	H	H L
1	1	1	1		H	H	H H

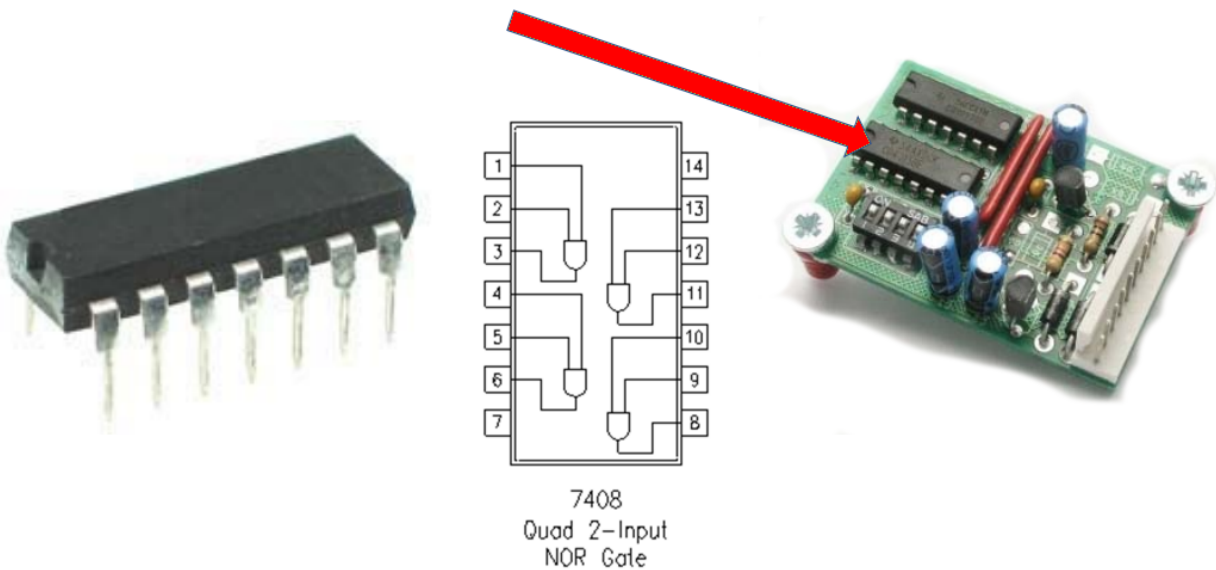
Elektrisch schema



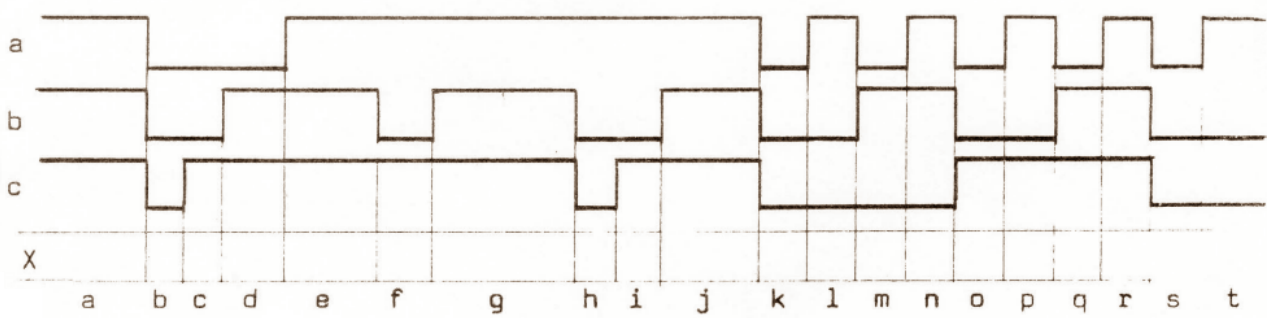
Symbol



EN poorten in een IC op een printplaat.



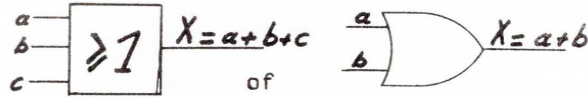
Beredeneer hoe het uitgangssignaal verloopt



OF POORT (or gate)

- Een OF poort is een schakeling met één uitgang en meerdere ingangen zodanig dat de uitgang 1 is, zodra één of meerdere ingangen 1 zijn.
- De uitgang wordt 0, en slechts dan 0, als alle ingangen 0 zijn.

Symbol



Waarheidstabel

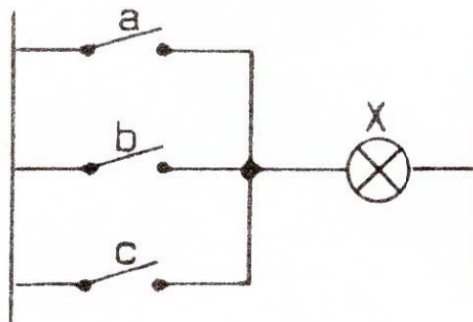
OF POORT
(or gate)

a	b	c	X
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

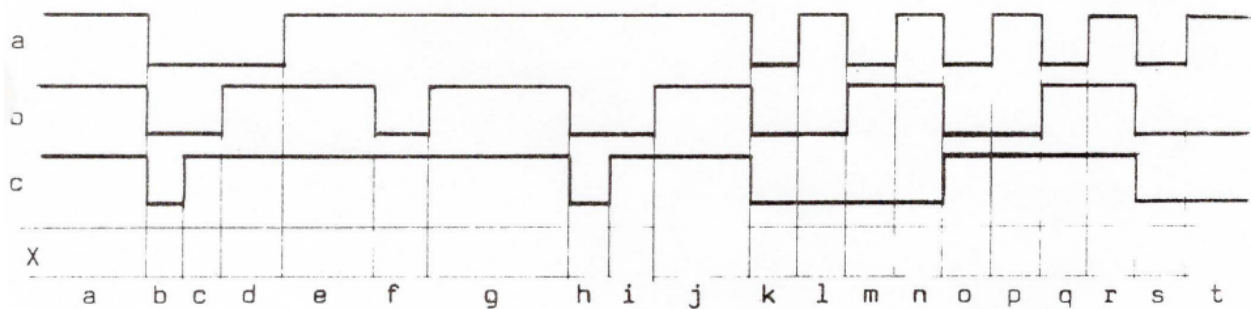
of

a	b	c	X
L	L	L	L
H	L	L	H
L	H	L	H
H	H	L	H
L	L	H	H
H	L	H	H
L	H	H	H
H	H	H	H

Elektrisch
schema



Beredeneer hoe het uitgangssignaal verloopt



NIET POORT (not gate)

- Een NIET poort is een schakeling met één ingang en één uitgang, waarvan de uitgang altijd de inverse toestand geeft van de ingang.
- De uitgang is 1 als de ingang 0 is, en is 0 als de ingang 1 is.

Waarheidstabel

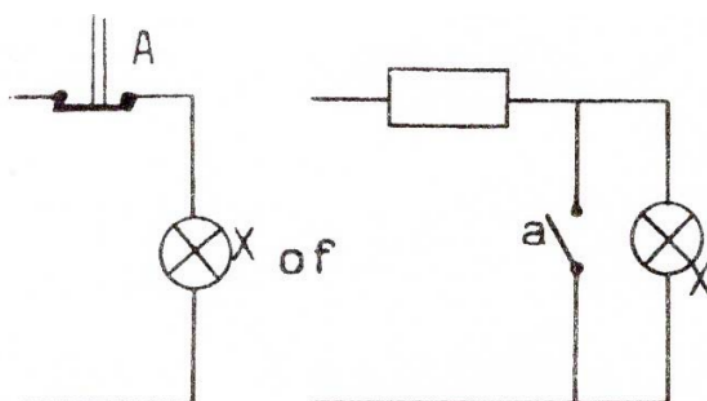
NIET POORT
(NOT gate,
invertor)

a	X
0	1
1	0

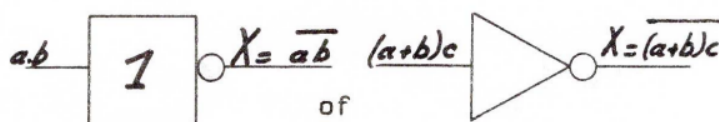
 of

a	X
L	H
H	L

Elektrisch
schema



Symbool



NEN POORT (nand gate)

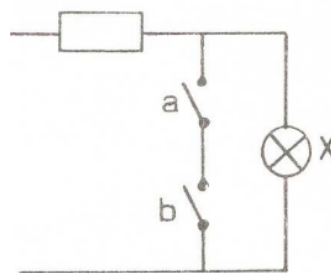
- Is een EN poort gevolgd door een NIET poort.
- De uitgang wordt 1 zodra één of meerdere ingangen 0 zijn. De uitgang is dan 0 enkel als alle ingangen 1 zijn.

Waarheidstabel

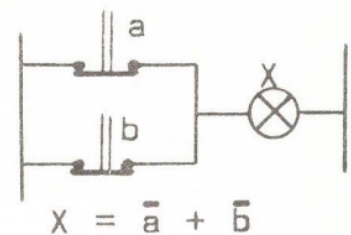
NEN POORT (NAND gate)

a	b	c	X		a	b	c	X
0	0	0	1	of	L	L	L	H
1	0	0	1		H	L	L	H
0	1	0	1		L	H	L	H
1	1	0	1		H	H	L	H
0	0	1	1		L	L	H	H
1	0	1	1		H	L	H	H
0	1	1	1		L	H	H	H
1	1	1	0		H	H	H	L

Elektrisch schema

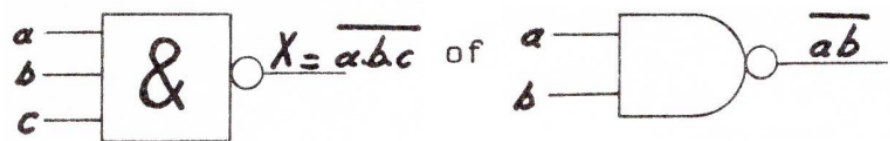


$$X = \overline{ab}$$

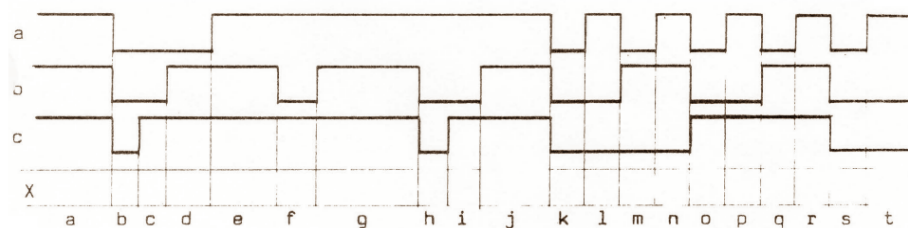


$$X = \overline{a} + \overline{b}$$

Symbool



Bereken hoe het uitgangssignaal verloopt



NOF POORT (nor gate)

- Is een OF poort gevolgd door een NIET poort.
- De uitgang wordt 0 zodra één of meerdere ingangen 1 zijn. De uitgang is dan 1 enkel als alle ingangen 0 zijn.

Waarheidstabel

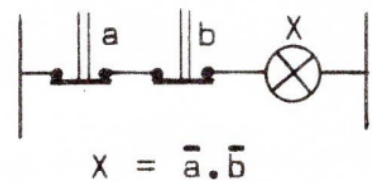
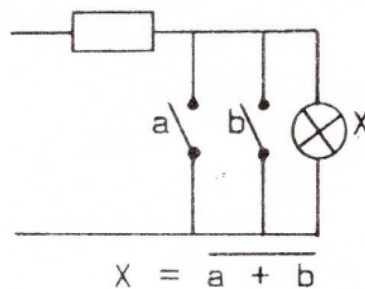
NOF POORT (NOR gate)

a	b	c	X
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0

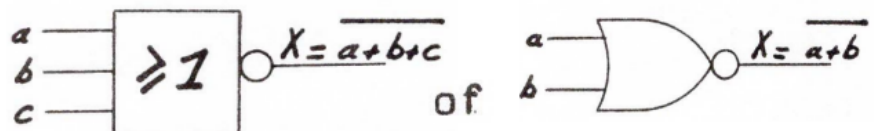
of

a	b	c	X
L	L	L	H
H	L	L	L
L	H	L	L
H	H	L	L
L	L	H	L
H	L	H	L
L	H	H	L
H	H	H	L

Elektrisch schema



Symbool



Berekeneer hoe het uitgangssignaal verloopt



EXOF POORT (exor gate)

- Is een poort waarvan de uitgang 1 is als één en slechts één ingang 1 is.
- De uitgang is dan 0 als beide ingangen dezelfde logische toestand hebben.

Waarheidstabel

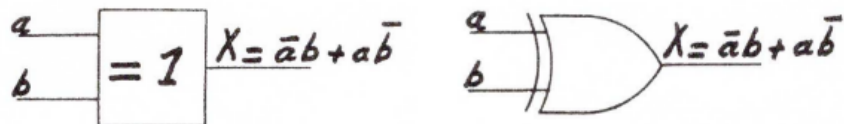
a	b	X
0	0	0
1	0	1
0	1	1
1	1	0

EXCLUSIEVE OF
(EXOR)

Elektrisch
schema



Symbool



VERGELIJKER (comparator)

- Is een poort waarvan de uitgang 1 is, en slechts één is, als alle ingangen dezelfde logische toestand hebben.

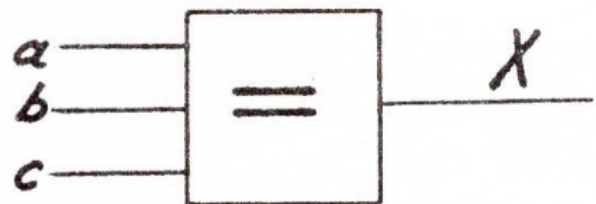
Waarheidstabel



VERGELIJKER (COMPARATOR)

a	b	c	X
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Symbool



Extra uitbreiding

Wat is dat allemaal? GPIO, SPI, I2C, en UART communicatie

Protocol	# Pins Required	Max Speed
GPIO	1	1kHz
SPI	3+ (MOSI, MISO, SCK + 1 GPIO pin)	25MBit/s
I2C	2 (SCL and SDA)	100kHz or 400kHz configurable by module port
UART	2 (TX and RX)	8Mbit/s

Drieletterwoorden hebben normaal gesproken niet veel toelichting nodig. Afkortingen van 3 (of 4) letters vaak wel. Daarom in dit bericht aandacht voor 4 afkortingen die te maken hebben met de communicatie tussen apparaten zoals een Raspberry Pi, een Arduino, de ESP8266, de BBC micro:bit met randapparatuur of sensoren.

GPIO / General Purpose Input Output

Het is zo ongeveer het eerste waar een designer / maker naar kijkt bij een apparaat: zitten er “pinnen” aan? 😊

En dan natuurlijk net zo belangrijk: zijn die pinnen als input of output poorten te gebruiken. De Raspberry Pi heeft ze, een Arduino heeft ze, de ESP8266 idem (al hebben sommige modellen er maar heel weinig), ook de BBC micro:bit heeft ze al worden ze daar niet standaard als pinnen ontsloten, door gebruik te maken van een breakout-box die je zelf kunt maken of kunt kopen, komen ze ook daar beschikbaar.

En het kan zijn dat je heel lang vooruit kunt zonder meer nodig te hebben dan dat. Wil je bv een LED aan of uit-schakelen, de waarde van een eenvoudige thermometer lezen, een relais schakelen, dan heb je aan 1 GPIO poort per device voldoende en zul je de andere protocollen niet eens gebruiken.

In andere gevallen wil je echter complexere data versturen en/of meer devices kunnen aansturen, en dan heb je verschillende keuzes. Voor alle drie de hieronder genoemde opties geldt dat zowel de Raspberry Pi (eerst aanzetten), een Arduino, de BBC micro:bit en de ESP8266 er ondersteuning voor hebben.

SPI / Serial Peripheral Interface

De eerste optie voor is SPI of Serial Peripheral Interface. Zoals uit de tabel al blijkt is dit het “snelste” van alle hier beschreven protocollen. Dus als er veel data over de lijn moet, bijvoorbeeld voor het aansturen van een ingewikkeld grafisch display of als er juist veel data ontvangen moet kunnen worden, dan is SPI daar geschikt voor.

Het vergt wel meer GPIO poorten: 3 + 1 voor elk aangesloten device (elke "slave").

De aansluitingen voor SPI zijn (hier is bv de Arduino de "master" en een LCD-scherm de slave):

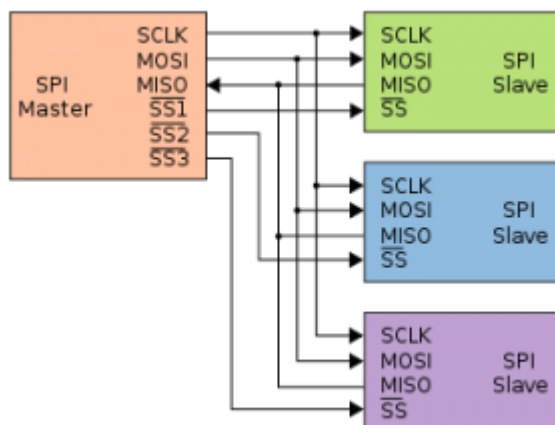
SCK = dit is het clock signaal dat er voor zorgt dat de master en de slave op dezelfde snelheid (de bitrate) met elkaar communiceren. Anders begrijpen ze elkaar niet.

MOSI = Master Out Slave In en is de verbinding waarmee de host met de slave communiceert

MISO = Master In Slave Out en is de verbinding waarmee de slave communiceert met de host

SS of CS = Slave Select (SS) of Chip Select (CS) hiermee wordt aangegeven met welke slave de master wil communiceren.

De selectie voor een slave kan op twee manieren: de master kan één GPIO-pin per slave gebruiken of er kan een ketting van slaves opgebouwd worden.



Meer info over het gebruik van SPI voor verschillende platformen:

Raspberry Pi

SPI staat standaard uit, je met SPI eerst

enablen WiringPi library voor SPI

SPI is beschikbaar in o.a. Python, PHP en

Ruby Voorbeeld Raspberry Pi <-> Arduino

via SPI. Let op! Arduino gebruikt 5V en

Raspberry Pi werkt met 3,3V!

Arduino

SPI op de Arduino

Voorbeeld uitlezen luchtdruksensor via SPI

De LED Matrix die ik getest heb maakt

gebruik van chained SPI, d.w.z. dat ik maar

één SS poort nodig heb, ook als er

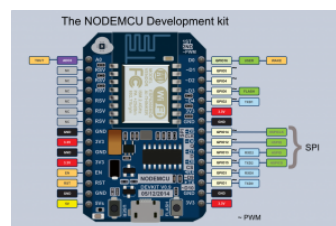
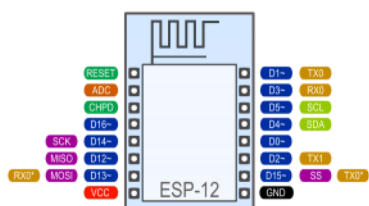
meerdere matrix modules gebruikt worden.

BBC micro:bit

microPython en SPI

ESP8266

Pin-out voor SPI bij de verschillende modellen is soms even zoeken, het handigste is sowieso om een GPIO-afbeelding te zoeken voor jouw specifieke ESP8266 (via Google heb je die zo gevonden) zodat je de juiste poorten (SCK, MOSI, MISO, SS) kunt opzoeken:

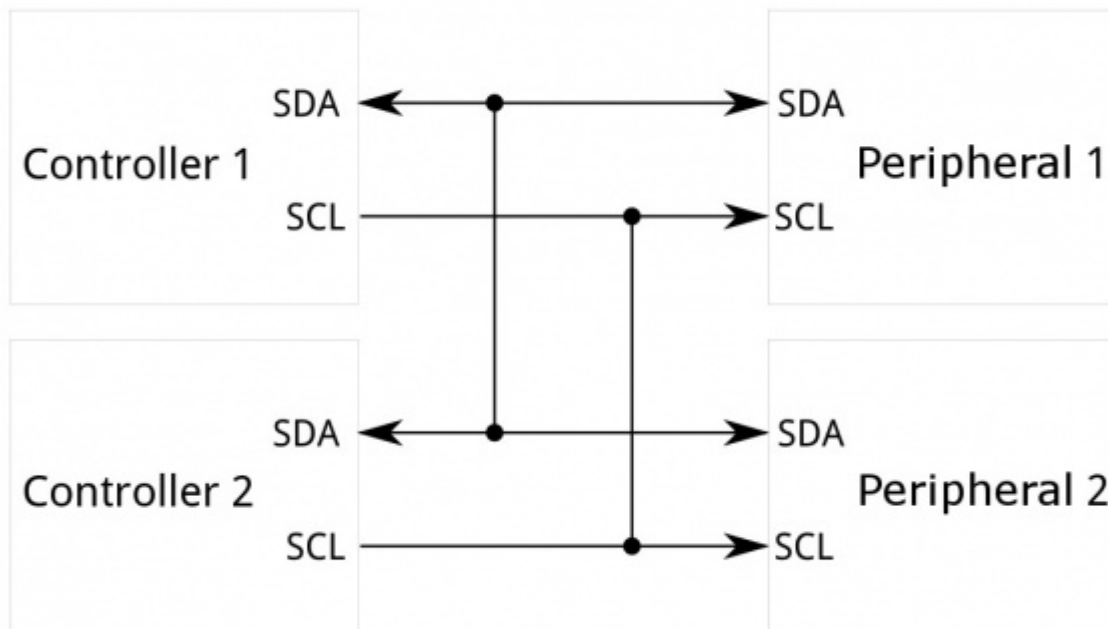


I2C / Inter-Integrated Circuit

De tweede optie is I2C of I²C, daarom vaak uitgesproken als “I squared C”, “I two C” of “I-I-C”. Dit protocol is niet zo snel als SPI, maar je hebt slechts 2 kabels nodig, ook als je meerdere devices wilt koppelen:

- **SCL** = dit is het clock signaal dat er voor zorgt dat de master en de slave op dezelfde snelheid (de bitrate) met elkaar communiceren. Anders begrijpen ze elkaar niet. Net als SCK bij SPI dus.
- **SDA** = dit is de datalijn voor de communicatie tussen de master(s) en de slave(s). Nu dus niet aparte lijnen zoals bij SPI, maar 1 datalijn. De master(s) zorgen voor de coördinatie zodat maar 1 device tegelijkertijd aan het “praten” is.

Zoals je ziet staat hierboven “master(s)”. Anders dan bij SPI namelijk kan I2C overweg met meer dan één master



Je hebt dus minder kabels nodig bij I2C dan bij SPI, maar levert daarvoor wel snelheid in. Bij I2C heeft elk device een eigen identificatienummer zodat het device op de “bus” geadresseerd kan worden. Deze identificatie ligt vaak vast en als twee devices hetzelfde nummer hebben kun je ze niet samen op een I2C bus gebruiken.

Een voorbeeld van twee I2C devices in een ketting geschakeld aan een Arduino kun je terugvinden op: http://www.goodliffe.org.uk/arduino/i2c_devices.php

UART / Universal Asynchronous Receiver / Transmitter

Last but not least: UART (Universal Asynchronous Receiver / Transmitter), een dure naam voor wat de ouderen onder ons gewoon kennen als een seriële poort. Je weet wel, dat ding waar je je computermuis op aansluit voordat er USB-muizen waren of zelfs PS/2 muizen. Overigens, voor wie het niet weet: USB = Universal Serial Bus = Universele Seriële Bus. USB is een stuk complexer (hogere snelheid, meerdere apparaten) en ga ik hier verder niet op in, UART wel omdat je ook die kunt gebruiken bij de apparaten waar we het eerder over hadden.

Voor de volledigheid ook nu de aansluitingen:

TX = wordt gebruikt door de master om data naar de slave te sturen

RX = wordt gebruikt door de slave om data naar de master te sturen

Als je een master en een slave op elkaar aansluit die UART ondersteunen, dan zullen beide apparaten een label TX en RX hebben. Let op! Je moet de TX van de master verbinden aan de RX van de slave en de TX van de slave gaat aan de RX van de master (stop en lees die zin nog even een keer!)

Om een master en slave via UART met elkaar te laten communiceren moeten ze een aantal dingen van elkaar weten:

- *Hoeveel data bits worden er per keer verstuurd (5 tot 8)?*
- *Hoe snel worden de data verstuurd (baud rate)?*
- *Wordt er een “parity bit” (controle bit) aan het einde verstuurd en is dat een 1 of een 0?*
- *Hoeveel stopbits worden er aan het einde van een transmissie verstuurd?*

Dit zijn zaken die je dus meestal in de code vooraf zult aangeven. Het voordeel van UART is dat het brede ondersteuning kent. Het is echter wel langzamer dan SPI en I2C en je kunt geen meerdere apparaten op dezelfde bus aansluiten.

Voorbeelden:

Arduino met SD-kaart als datalogger via UART

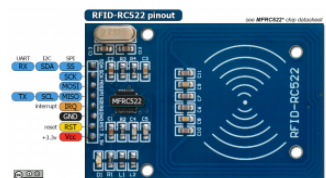
<https://docs.arduino.cc/tutorials/ethernet-shield-rev2/Datalogger>

Arduino met RTC (real time clock) via UART

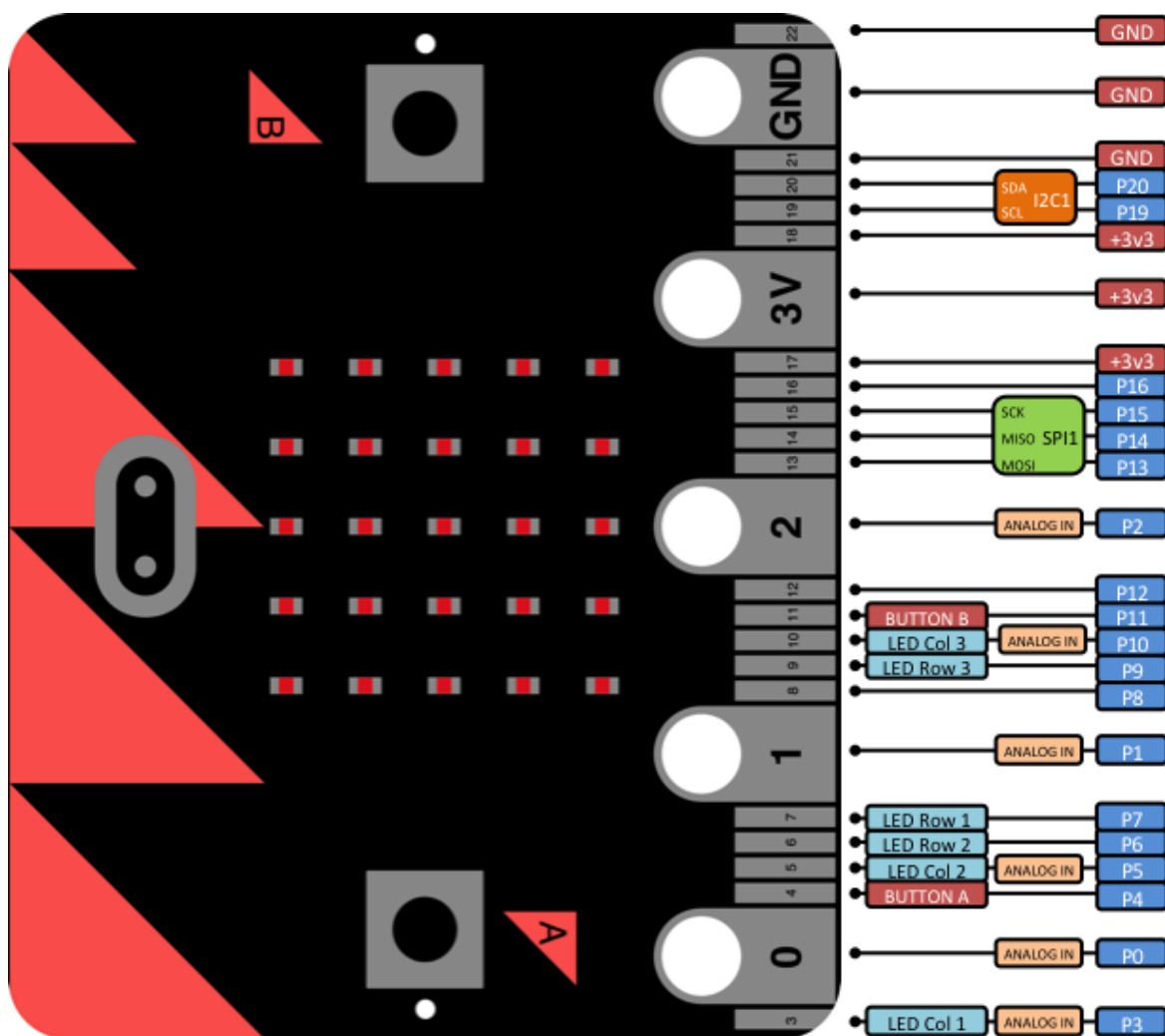
<https://www.instructables.com/Setting-the-DS1307-Real-Time-Clock-using-the-Seria>

Let op!

Soms kan het zijn dat een device voor meerdere protocollen ondersteuning heeft. Dan kan het nog afhankelijk zijn van de driver. De RFID-RC522 RFID-scanner bijvoorbeeld, zie ook de afbeelding hiernaast, zowel ondersteuning voor SPI, I2C als UART, maar de Arduino library heeft bv enkel alleen ondersteuning voor SPI.



Soms zijn de pinnen voor een bepaald protocol niet altijd even eenvoudig te herkennen. Ik noemde al de ESP8266, ook bij de BBC micro:bit zitten ze goed verstopt:

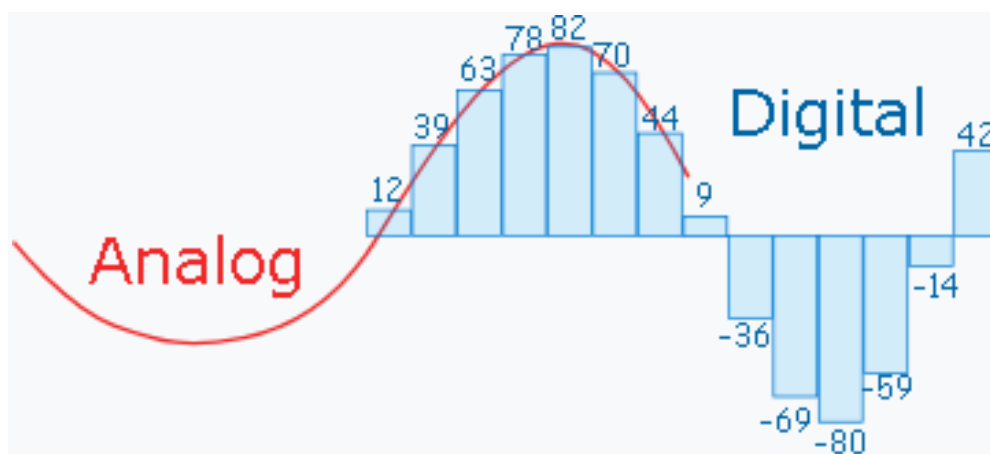


Maar als het goed is zou je nu voldoende moeten weten om een keuze te maken tussen GPIO, SPI, I2C en UART én moeten kunnen herkennen aan de labels bij de pins of in de documentatie welke opties een device heeft.

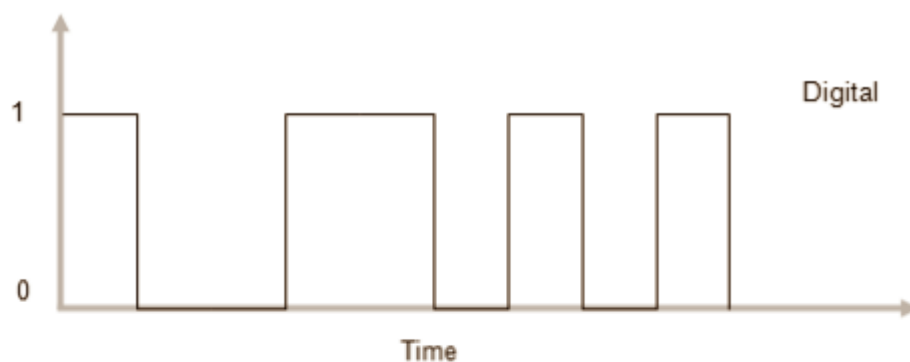
Wie zich verder wenst te verdiepen in arduino raad ik volgende boeken aan:

- [Arduino Development Cookbook](#)
- [Arduino Electronics Blueprints](#)
- [Arduino Projects for Amateur Radio](#)
- [Arduino Cookbook](#)

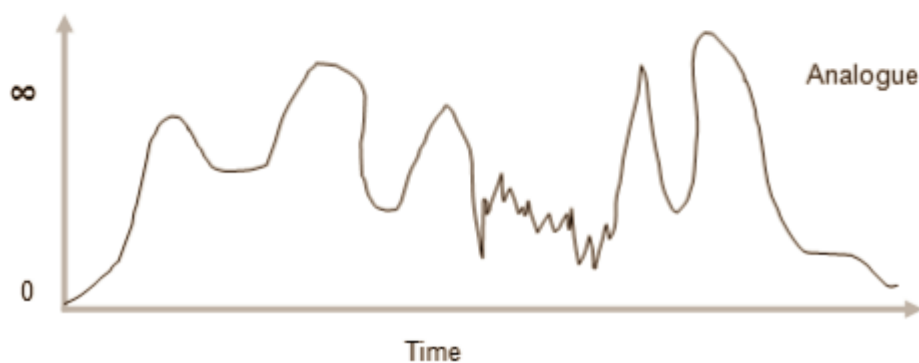
Ditjes en datjes Analooq vs Digitaal



Digitaal en analoog

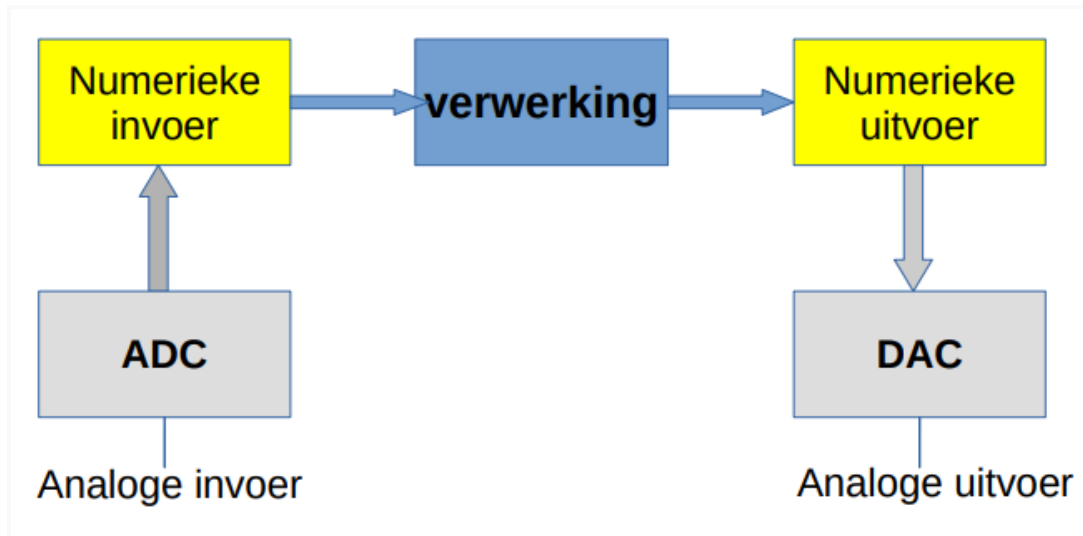


Een digitaal signaal is discontinu in tijd en de amplitude heeft alleen discrete waarden



Een analoog signaal is continu in tijd en amplitude

Digitale systemen



Elk digitaal systeem werkt met getallen!

Getalsystemen

Decimaal getal systeem

grondtal = 10 er zijn 10 cijfers:

0,1,2,3,4,5,6,7,8,9

positie cijfer bepaalt de getalwaarde

$2019 = 9 \times 1 + 1 \times 10 + 0 \times 100 + 2 \times 1000$

$2019 = 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 9 \times 10^0$

dit geldt ook voor getallen kleiner dan 1:

$1,75 = 1 \times 10^0 + 7 \times 10^{-1} + 5 \times 10^{-2}$

negatieve getallen worden voorafgegaan door '-' teken

BINAIR getal systeem

grondtal = 2 er zijn 2 cijfers: 0,1

positie cijfer bepaalt de getalwaarde

$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$1011 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 11$ decimaal

dit geldt ook voor getallen kleiner dan 1:

$11,01 = 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

$11,01 = 2 + 1 + 0 + 0,25 = 3,25$ decimaal

negatieve getallen worden (meestal) weergegeven

met het 'two-complement' voorbeeld: - 17

- 1011 -----> 01011 -----> 10100 -----> 10101

nul voorzetten inverteren 1 bijtellen

Machten van 2

n bits:

Aantal combinaties
= 2^n

Grootste dec. Getal
= $2^n - 1$

aantal keer snijden	aantal lagen
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
17	131 072
18	262 144
19	524 288
20	1 048 576
21	2 097 152
22	4 194 304
23	8 388 608
24	16 777 216
25	33 554 432
26	67 108 864
27	134 217 728
28	268 435 456
29	536 870 912
30	1 073 741 824
31	2 147 483 648
32	4 294 967 296

aantal keer snijden	aantal lagen
33	8 589 934 592
34	17 179 869 184
35	34 359 738 368
36	68 719 476 736
37	137 438 953 472
38	274 877 906 944
39	549 755 813 888
40	1 099 511 627 776
41	2 199 023 255 552
42	4 398 046 511 104
43	8 796 093 022 208
44	17 592 186 044 416
45	35 184 372 088 832
46	70 368 744 177 664
47	140 737 488 355 328
48	281 474 976 710 656
49	562 949 953 421 312
50	1 125 899 906 842 624
51	2 251 799 813 685 248
52	4 503 599 627 370 496
53	9 007 199 254 740 992
54	18 014 398 509 481 984
55	36 028 797 018 963 968
56	72 057 594 037 927 936
57	144 115 188 075 855 872
58	288 230 376 151 711 744
59	576 460 752 303 423 488
60	1 152 921 504 606 846 976
61	2 305 843 009 213 693 952
62	4 611 686 018 427 387 904
63	9 223 372 036 854 775 808
64	18 446 744 073 709 551 616

HEXADECIMAAL getal systeem (h of \$ of 0x)

grondtal = 16 er zijn 16 cijfers: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

positie cijfer bepaalt de getalwaarde

$$3FA6 = 3 \times 16^3 + 15 \times 16^2 + 10 \times 16^1 + 6 \times 16^0$$

$$3FA6 = 3 \times 4096 + 15 \times 256 + 10 \times 16 + 6 \times 1 = 16294 \text{ decimaal}$$

dit geldt ook voor getallen kleiner dan 1:

$$2F,3A = 2 \times 16^{-1} + 15 \times 16^{-2} + 3 \times 16^{-3} + 10 \times 16^{-4}$$

$$2F,3A = 32 + 15 + 3 \times 0,0265 + 10 \times 0,0039625 = 47,119125 \text{ decimaal}$$

negatieve getallen worden van een '-' teken voorzien

DEC – HEX – BIN

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

Rekenen met binaire getallen

Rekenregels

$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 0$ en 1 links bijtellen
 $0 - 1 = 1$ en links 1 aftrekken

Voorbeelden

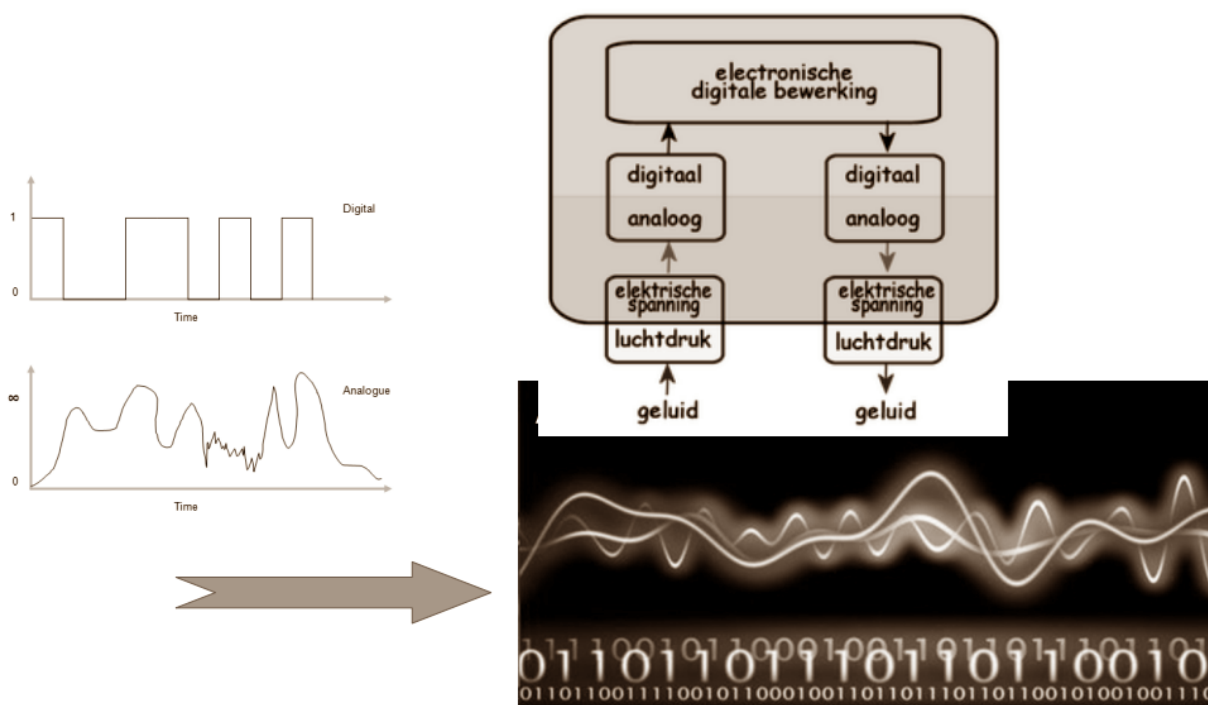
```
  1011
+ 1100
-----
 10111
```

```
  10111
- 1101
-----
 01010
```

```
  10100
X   101
-----
  10100
 00000
 10100
-----
1100100
```

Binair rekenen gebeurt op dezelfde manier als met decimale getallen

Verband binaire getallen en digitale signaalverwerking



Binaire codesystemen

BCD-code

- voor het coderen van arabische cijfers met een 4-bit code
- er zijn daarom alleen 10 'toegelaten' codes!
- Is een gewogen code (8-4-2-1)
- wordt gebruikt o.a. voor displays

	MSB	BCD ₈₄₂₁		LSB
Decimal	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Voorbeeld:

De **cijfers** 183 in BCD-code:

0001.1000.0011

Opmerking: het **getal** (=aantal) 183

183 = B101010111

GRAY-code

- voor het coderen van 16 verschillende toestanden met 4 bits
- Is een niet-gewogen code-systeem
- de eerste 10 codes zijn BCD-codes
- voor elke volgende code wijzigt slechts 1 bit in de code
- wordt gebruikt in bijvoorbeeld rotary encoders

Decimal Number	4 bit Binary Number <u>ABCD</u>	4 bit Gray Code <u>G₁G₂G₃G₄</u>
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

ASCII-code

- binaire code voor letters, cijfers en controletekens
- ASCII = American Standard Code for Information Interchange
- bedoeld om alfanumerieke data tussen apparaten uit te wisselen
- oorspronkelijk 7 bit brede code (maximaal 128 tekens)
- veelal 8 bit breed (maximaal 256 tekens)
- er bestaan tegenwoordig diverse uitbreidingen: extended ascii, IBM (PC-) ascii, unicode...

(Oorspronkelijke) ASCII-codetabel

0-31
controletekens

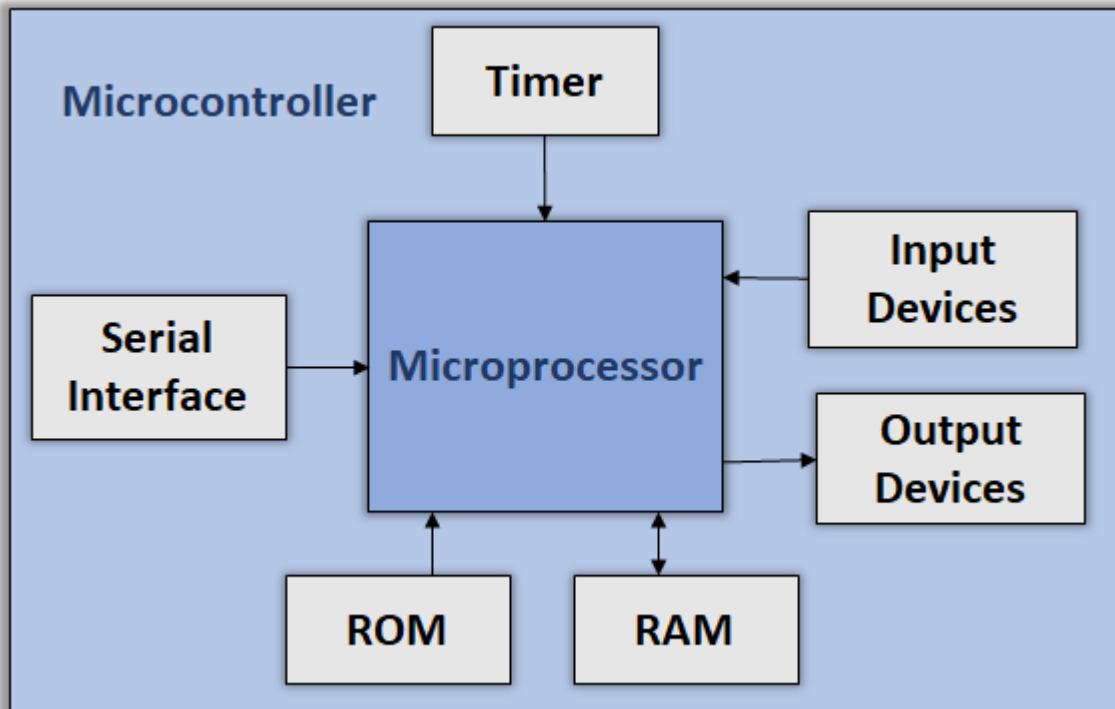
48-57
Cijfers

65-90
Hoofdletters

97-122
kleine letters

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F		127	7F	□

Ditjes en datjes over Microprocessor, Micro Computer, Microcontroller



Microprocessor:

- Is de CPU (CPE) van een computersysteem op één chip
- Uitgevoerd in verschillende bit breedtes: 4, 8, 16, 32... bit
- In verschillende families (Z80, 8085, 8086, pentium,) met eigen instructieset (RISC of CISC)

Microcomputer:

- Computer gebouwd rond een microprocessor
- Met discrete geheugenchips en discrete ondersteunende chips (systeemklok, parallelle en/of seriële I/O-schakelingen, interrupt controller, enz...)
- De samenstellende onderdelen zijn verbonden met een adres-, data- en controlebus.
- Inzetbaar in een groot aantal, niet vooraf gedefinieerde taken.

Microcontroller:

- Microcomputer met de samenstellende delen (CPU, geheugen, poorten, timers, ADC, enz. op één chip
- Hoofdzakelijk gebruikt voor een specifieke applicatie in meet- en regel toepassingen
- Doorgaans met beperkt geheugen
- Busstructuur extern niet toegankelijk
- Communicatie loopt via programmeerbare poorten
- Meestal intern Harvard-architectuur
- Twee grote families: PIC en AVR (vroeger: 8051 en co)

Ditjes en datjes over Programmeertalen

Er bestaan wel honderden programmeertalen:

- **1ste generatie:** *machinetaal*: in binaire opcode programmeren
- **2-de generatie:** *assembler*: in mnemonics direct in machinetaal omzetbaar
- **3-de generatie:** *procedurele talen*
zoals cobol, basic, pascal, C, C++
- **4-de generatie:** *talen met een hoger abstractieniveau*
zoals Progress, SQL

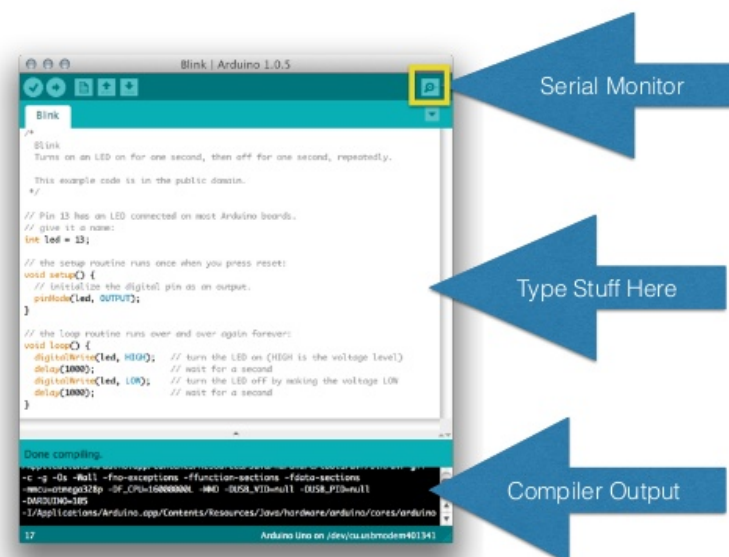
```

010010000000010100011011000000100110100011
110001011011010001000111111111111011010000100
01010011011000010101101101101101101010010001
11011000000101010010001000001100010011111
01001100010101001011011011101101110111111
00010101001include<stdio.h>01010000011001
10001001000000000100010001111
10001001int main()0001111111
1010010010001000100
11001001return f("Hello World");0001100
0001001000010001000100100100100100100100100
00010100110110110110110110000001010001111
1000100100001010100100110110100101111
11001100110010101101100010101001001000
110011000001011111010100111011011001100
1000000111111010010010101010101101111

```

Een IDE (Integrated development environment)

Arduino IDE



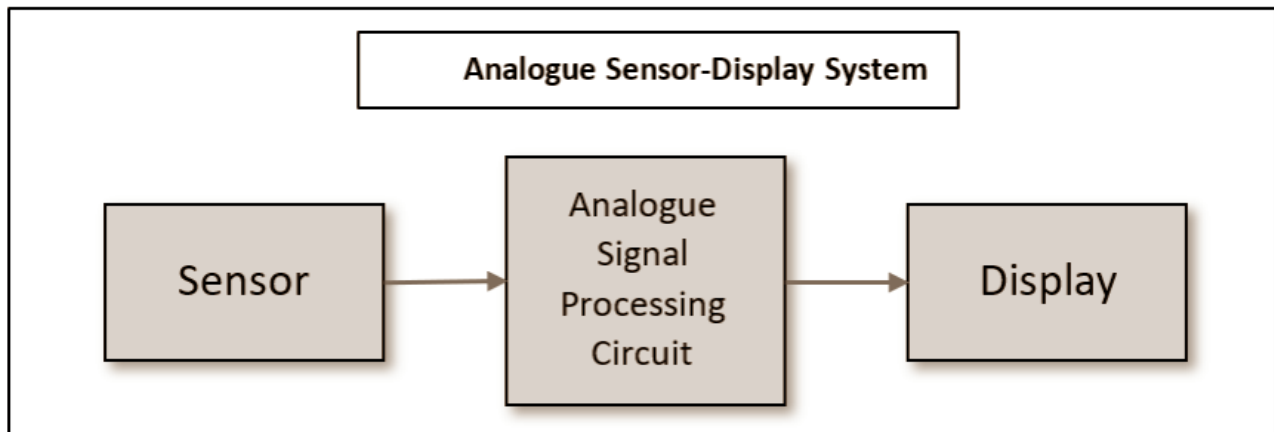
Een IDE bestaat minstens uit:

- een editor
- een compiler/linker
- documentatie

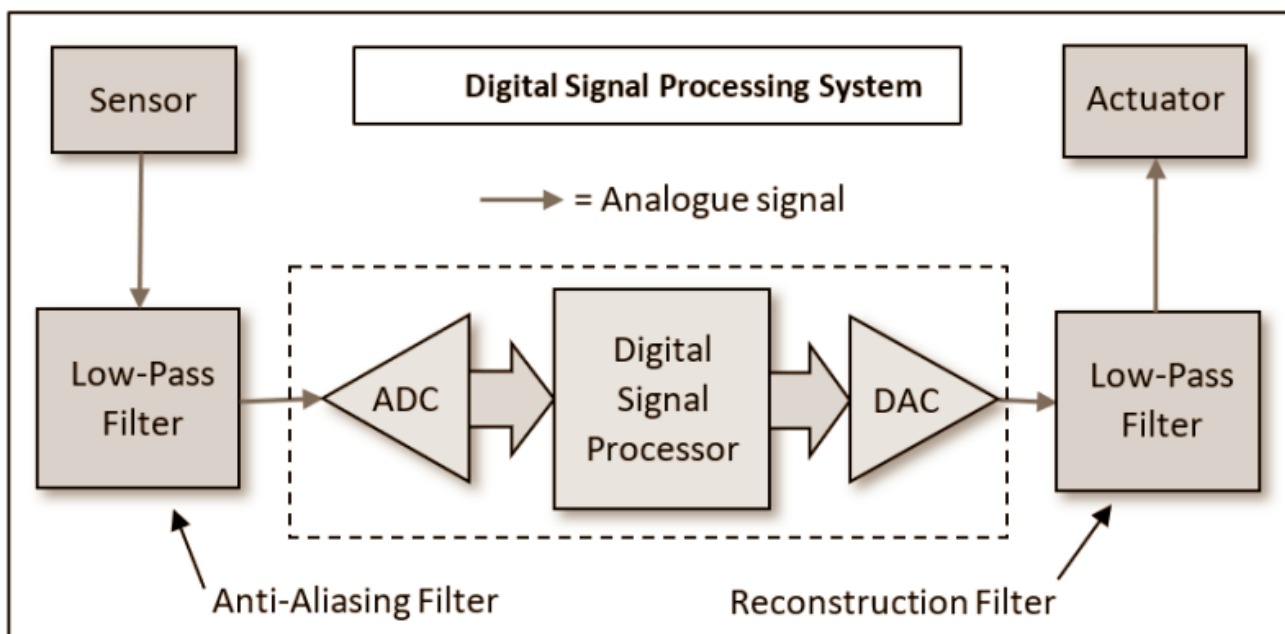
eventueel uitgebreid met:

- *een debugger*
- *grafische interface*
- *hulpmiddelen zoals versiebeheer*

Analoog wordt digitaal (en omgekeerd)



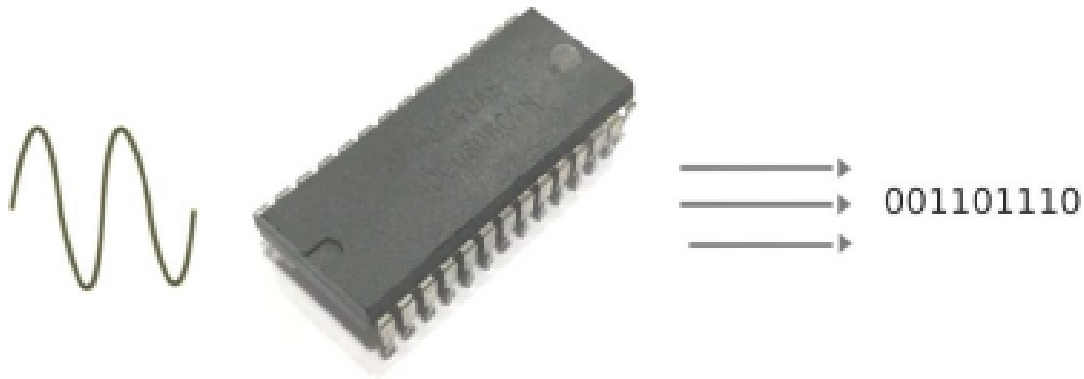
- In de 'echte' wereld veranderen alle fysische grootheden op continue wijze
- Een sensor zet de waarde van een fysische grootheid om in een overeenkomstige elektrische spanning (= analoog signaal)
- Na (eventuele) bewerking wordt dit weergegeven op een analoog 'display'



In een digitaal systeem wordt het signaal voor verwerking omgezet in binaire getallen via bemonstering.

Na bewerking kunnen de binaire getallen terug omgezet worden in een analoog signaal om een actuator aan te sturen.

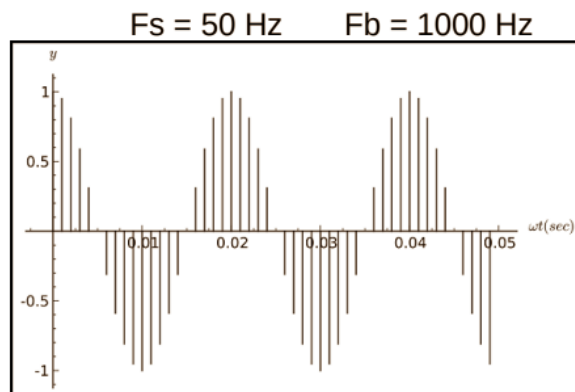
Analog to Digital Converters



Bemonstering:

- bemonsterfrequentie
- resolutie: bitbreedte

Stelling van Nyquist:

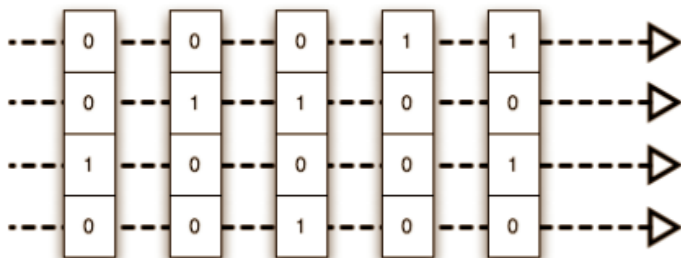


De bemonsterfrequentie moet minstens 2 x zo hoog zijn als de hoogste frequentie in het signaal

Daarom: laagdoorlaatfilter bij complexe signalen om 'aliassen' te vermijden

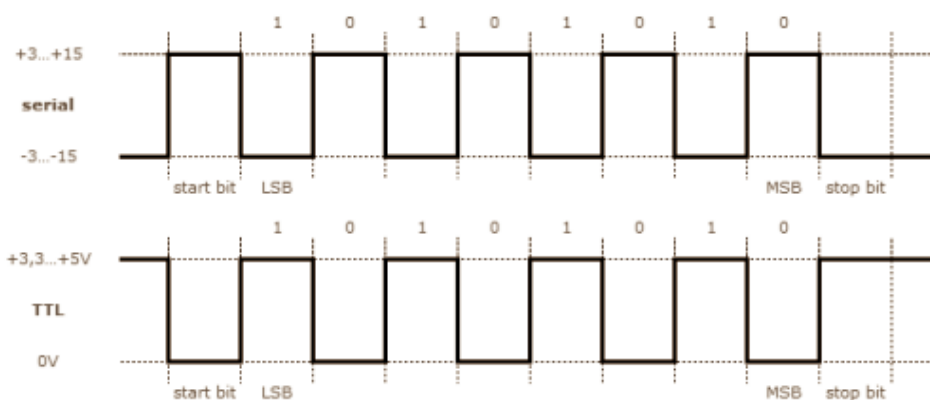
Communicatie tussen digitale apparaten

PARALLELE



Groepen van bits worden gelijktijdig overgedragen (meerdere lijnen)

SERIEEL



Groepen van bits worden bit per bit over 1 lijn overgedragen

Communicatiesystemen en protocollen

Parallele communicatie

- *simplex, half- en full duplex*
- *handshaking: softwarematig: XON en XOFF(CTRL-P/Q)*
- *hardware mating: ACK, BUSY, STROBE*

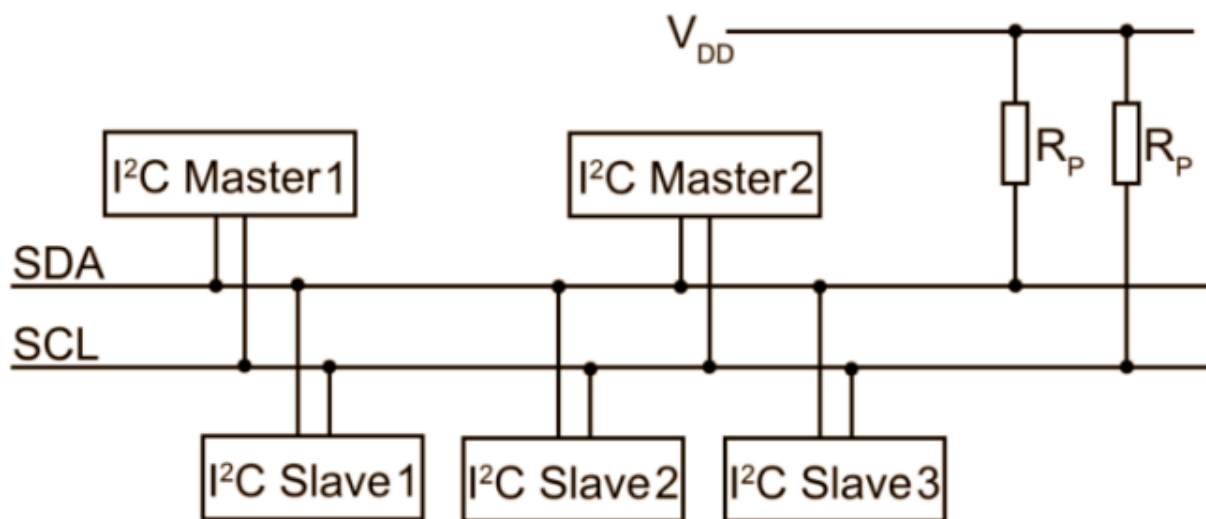
Seriële communicatie (asynchroon)

- *simplex, half en full duplex; RxD en TxD*
- *baud rate (bit breedte/teken x aantal tekens/seconde)*
- *start- en stopbits*
- *spanningsniveau: RS-232 en TTL*
- *RTS en CTS – DTR en DSR*

Synchrone seriële communicatie: I2C

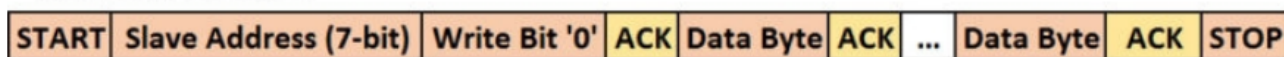
BUS:

- *2 lijnen: SDA (data) en SCL (clock)*
- *gebruikt een 7 bit adres*
- *maximaal 128 adressen waarvan er 12 gereserveerd zijn*
- *multi-master, multi-slave*
- *pull-up-weerstanden zijn vereist!*

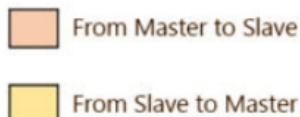
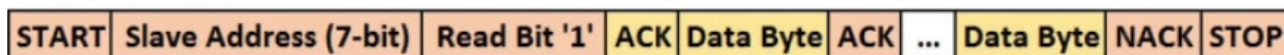


PROTOCOL

Write data to slave



Read data from slave



ACK = Acknowledge (SDA Low)
NACK = Not Acknowledge (SDA High)

START en STOP

- Terwijl SCL hoog is zijn start en stop gedefinieerd als stijgende/dalende flank van SDA

ADRES en R/W

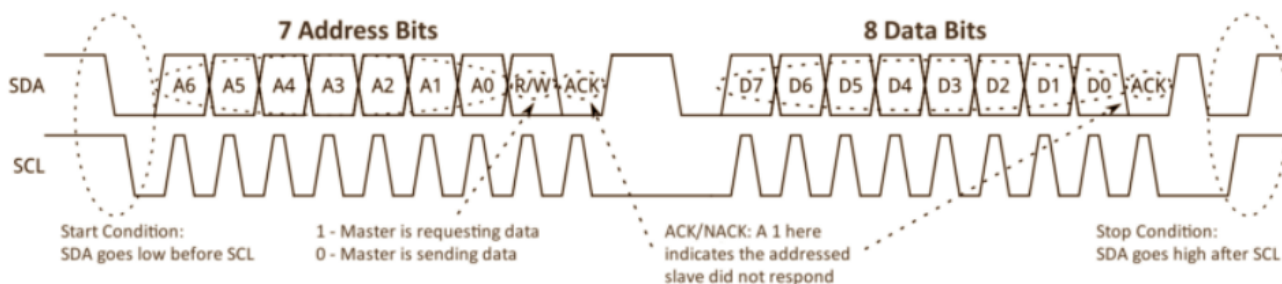
- Na START volgt het adres gevolgd door R/W-bit op de SDA-lijn

DATA Zenden (R/W=0):

- Na ACK ('0') door slave op SDA-lijn zendt master een databyte slave antwoordt na elke databyte met ACK ('0')

lezen (R/W=1)

- Na ACK op uitgezonden adres leest master een databyte en stuurt een ACK. Als het de laatste byte is stuurt de master een NACK ('1') op de SDA-lijn



Voorbeeldprogramma gebruik LCD met e-4S board

```
#include <E4SBoard.h>
E4SClass E4SBoard;
#include <LiquidCrystal_I2C.h>
//*****
//***** variabelen *****
//*****

bool DrukknopState;

int lcdColumns = 20;           //initialiseer aantal kolommen
int lcdRows = 4;               //initialiseer aantal rijen
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows); //adres=0x27

void setup()
{
  //*****
  //*****input/output defenitie*****
  //*****

  lcd.begin();                 //altijd oproepen voor dat je andere LCD statements gebruikt.
  lcd.backlight();             //verlichting LCD aan.
}

void loop()
{
  //*****
  //***** inputs lezen *****
  //*****

  DrukknopState = E4SBoard.DIn(4);

  //*****
  //*****verwerken *****
  //*****

  if (DrukknopState == 1)
  {
    lcd.clear();               //maak LCD scherm leeg
    lcd.setCursor(0, 0); // (col,row) //zet de cursor op kolom 0 en rij 0
    lcd.print("EINDELIJK");    //schrijf EINDELIJK
    lcd.setCursor(0, 1);       //zet de cursor op kolom 0 en rij 1
    lcd.print("TEXT");         //schrijf TEXT
    lcd.setCursor(0, 2);       //zet de cursor op kolom 0 en rij 2
    lcd.print("OP HET DISPLAY"); //schrijf OP HET DISPLAY
  }

  //*****
  //***** outputs Zetten *****
  //*****

}
```

Notities

Notities

Notities

Notities

Notities
