

**Geïntegreerde Proef:**

**Home Control**

--- “Eenvoudig en snel huishoudelijke apparaten aansturen met een gebruiksvriendelijke interface zonder hierbij internet-privacy te verstoren.”

TSO Elektriciteit elektronica

2018 – 2019

**Technisch directeur:** E. Van Gucht

**Mentor school:** E. Arckens

**Leerling:** Stijn Rogiest

**Voorwoord**

Als eindejaarsleerling in de richting Elekriciteit-Electronica kreeg ik de opdracht om een geïntrigeerde proef af te leggen. Dit houdt in dat we een volledig jaar aan een project moeten werken om dan aan het einde van het jaar dit voor te stellen aan een jury.

Na veel brainstormen heb ik gekozen om een domotica systeem te maken, dit houdt in: het eenvoudig aansturen van verschillende elektronische apparaten via een gebruiksvriendelijke interface.

Deze opdracht heb ik uitgevoerd met een klasgenoot genaamd *Dylan Dedapper*. Om een goede werkomgeving te creëren hebben we gezorgd voor een goede taakverdeling, dit was zeker niet altijd even makkelijk.

Tijdens het project zijn we verschillende obstakels en uitdagingen tegengekomen. Ik heb tijdens dit project vele dingen bijgeleerd door mijzelf uit te dagen en hiervoor een oplossing te zoeken.

Hierbij wil ik alvast mijn begeleidende leerkrachten meneer Coppejans en meneer Arckens bedanken voor hun hulp en steun gedurende het schooljaar.

Inhoud

[2 Inleiding 4](#_Toc7017824)

[3 Omschrijving van de opdracht 4](#_Toc7017825)

[4 Werking van het apparaat 5](#_Toc7017826)

[4.1 Algemene werking 5](#_Toc7017827)

[4.2 Uitgebreide uitleg software 5](#_Toc7017828)

[4.2.1 Bij de ‘master’ 6](#_Toc7017829)

[4.2.2 Bij de ‘slave’ 7](#_Toc7017830)

[4.2.3 Protocol berekeningen 8](#_Toc7017831)

[4.3 Uitgebreide uitleg hardware 11](#_Toc7017832)

[4.3.1 Microcontrollers 11](#_Toc7017833)

[4.3.2 Communicatie 12](#_Toc7017834)

[5 Elektrische schema’s 13](#_Toc7017835)

[5.1.1 Prototypes 13](#_Toc7017836)

[5.1.2 Stroommeting 15](#_Toc7017837)

[5.1.3 Componentenlijst 16](#_Toc7017838)

[5.1.4 Professioneel PCB 16](#_Toc7017839)

[6 Design en afwerking 16](#_Toc7017840)

[6.1 Smartphone app 16](#_Toc7017841)

[6.2 Behuizing 16](#_Toc7017842)

[7 Bronnenlijst 16](#_Toc7017843)

[8 Logboek 17](#_Toc7017844)

[9 Besluit en zelfreflectie 17](#_Toc7017845)

[10 Bijlagen 17](#_Toc7017846)

# Inleiding

Deze bundel is de schriftelijke weergave van mijn geïntegreerde proef waaraan ik heel dit schooljaar aan heb gewerkt. Zoals de titel aangeeft, is het hoofddoel van dit eindwerk ‘Home Control’.

In dit document zullen verschillende dingen aan bod komen:

* Eerst zul je een duidelijke omschrijving van de opdracht te horen krijgen: *wat is een ‘Home Control’-systeem? Wat was belangrijk bij het kiezen van de opdracht?*
* Dan een uitgebreide uitleg over de werking van software en hardware: *welke programmatuur was belangrijk? Hoe worden datapakketten verstuurd en opgesteld? Welke elektronische componenten zijn gebruikt + wat doen ze? Ontwerp van de schema’s*.
* Vervolgens een logboek, een schriftelijk overzicht over de ‘werkuren’ binnen deze opdracht.
* Ten slotte een besluit en zelfreflectie: *Wat vind ik van het eindresultaat? Wat heb ik allemaal bijgeleerd? Wat had ik liever beter gedaan?*

Bij het maken van dit eindwerk heb ik enorm veel bronnen van het internet gebruikt, deze zullen zich bevinden waar ik ze raadpleegde. Er is ook een bronnenlijst bijgevoegd aan het einde van dit document.

# Omschrijving van de opdracht

We kregen de gelegenheid om zelf een GIP-opdracht uit te vinden en voor te stellen aan de leerkrachten. Uiteindelijk kwam dit tot een Dominica-systeem, een systeem dat het leven in huis makkelijker zal maken door gebruik te maken van elektronica. Simpelweg: draadloos lichten bedienen vanaf je smartphone, laptop of tablet.

Privacy was ook enorm belangrijk om even over na te denken, de home-control systemen die zich nu al op de markt bevinden hebben dit internet-privacy probleem. Al deze systemen verbinden met een database om gebruikersactiviteiten te loggen, en dat is ‘not done’. Gebruikers van dit soort systemen willen niet dat hun hele leven zich ergens op een computer bevindt zonder dat ze er iets van weten! Daarom wordt er in ons systeem geen verbinding gemaakt met het internet, dus er kan ook geen gelogde data online komen, dit zal al veel mensen geruststellen.

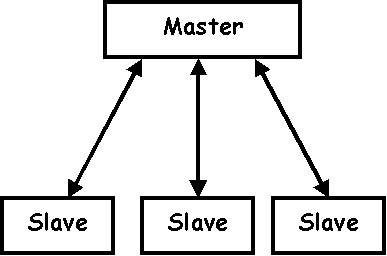
In het kort: een draadloos home-control systeem waarbij je eenvoudig en snel huishoudelijke apparaten kan aansturen via een gebruiksvriendelijke smartphone app zonder jezelf vragen te stellen over privacy.

# Werking van het apparaat

In dit deel leg ik uit hoe de hardware en de software van dit grote project samenvloeit. Het protocol wordt hier zeer uitgebreid uitgelegd, omdat dit ook het hart van het project is, de draadloze communicatie tussen de verschillende modules in het huis.

## Algemene werking

Ons systeem bestaat uit 2 verschillende modules, een slave en een master. Bij een typisch gebruik van dit systeem is een maar 1 master en zijn er meerdere slaves. De master ontvangt commando’s van de smartphone app die hij dan zal doorverwijzen naar zijn slaves. Een simpele schematische voorstelling ziet er zo uit:



Elke slave heeft een uniek adres: hiermee kan elke slave apart bestuurd worden. De master verstuurd een datapakket naar alle slaves met hierin het adres van de ontvanger, als de slave een datapakket ontvangt en het bevat zijn adres, interpreteert de slave de data, anders wordt het pakket genegeerd. Met het ontvangen datapakket kan hiermee een licht (of iets anders) aangestuurd worden. Wanneer het interpreteren succesvol was stuurt de slave een bericht terug naar de master dat alles ‘oké’ is.

In het volgende onderdeel wordt uitgelegd hoe de communicatie tussen master en slave(de pijlen) op software niveau word gerealiseerd. Hierna ook op hardware niveau.

## Uitgebreide uitleg software

De communicatie tussen de master en de slaves wordt gerealiseerd door een protocol: HCP oftewel ‘Home Control Protocol’, dit is een enorm onderdeel van dit project. Het hardware aspect van de communicatie wordt later nog uitgebreid uitgelegd.

Om uiteindelijk een stabiel protocol ontwikkeld te hebben, heeft het protocol maar liefst 4 versies opgevolgd. Na de eerste 3 versies, was ik klaar om de 4de versie zo foutloos mogelijk te ontwikkelen, ik had al heel wat ervaring opgelopen met de draadloze communicatie en de programmeertaal C++, ik wist welke problemen er ontweken moesten worden. Ik ga het in dit deel dus alleen hebben over hoe de 4de versie van het protocol werkt (hoewel ze allemaal volgens hetzelfde principe werken).

Een lijst van problemen die ontweken moesten worden in HCP versie 4 + de oplossing hiervoor:

* Er mag op geen enkel moment 2 modules op hetzelfde moment data versturen: dit vergroot de kans van datacorruptie enorm. *Oplossing: een geavanceerd timings-systeem dat ervoor zorgt dat elke slave zijn moment heeft om te sturen, deze momenten zijn niet overlappend.*
* Draadloze communicatie kan zeer snel verstoren, er moet dus een systeem zijn dat zorgt dat er geen corrupte data wordt verwerkt. *Oplossing: een ‘Cyclic redundancy check’ (of checksum;* [*https://en.wikipedia.org/wiki/Cyclic\_redundancy\_check*](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)*) bijvoegen (een code die een datapakket zijn inhoud beschrijft in 2 bits, als er 1 bit wordt verstoord in het pakket, veranderd de code meteen), hiermee kan er gecheckt worden of data is gewijzigd tussen de zender en ontvanger.*
* Adressering en koppelen van slaves: in vorige versies werden slaves een adres toegewezen bij het uploaden van het softwareprogramma. Dit beperkt het aantal slaves op planeet aarde omdat elk een uniek adres moet hebben + het datapakket mag niet te groot worden (het adres wordt meegestuurd). *Oplossing: een pair modus uitvogelen, gebruik maken van een fabriekscode (unieke code voor elke slave op de aarde) en een manier om deze codes op te slaan samen met een adres in de master module. Elke fabriekscode correspondeert met een lokaal adres.*

Een lijst van doelen die ik wil implementeren in HCP versie 4:

* Een overzicht kunnen verkrijgen van de master waarop je kan zien welke slaves er offline (ontkoppeld) zijn en welke er online zijn.
* Meerdere commando’s per pakket versturen. *Praktisch voorbeeld: de 3 rood, groen en blauw waardes die nodig zijn om een ledstrip aan te sturen in 1 pakket versturen i.p.v. aparte pakketten. Dit zal de snelheid enorm verhogen.*
* Een variabele lengte van datapakketten: elk datapakket mag geen onnodige bytes bevatten, dit vertraagt het verzenden en ontvangen van data. Elk datapakket heeft een variabele lengte van 4 bytes tot 20 bytes.

Zowel de master en de slave maken gebruik van de HCP-library die ik ontwikkeld heb in C++, simpelweg: dit is een verzameling van gedeelde code. Elke aanpassing die ik maak aan de library zorgt voor een aanpassing bij de slave en bij de master.

De library zorgt voor verschillende functies die zowel de master als de slave gebruiken: verzenden en ontvangen van datapakketten, inhoud van datapakketten vertalen ...

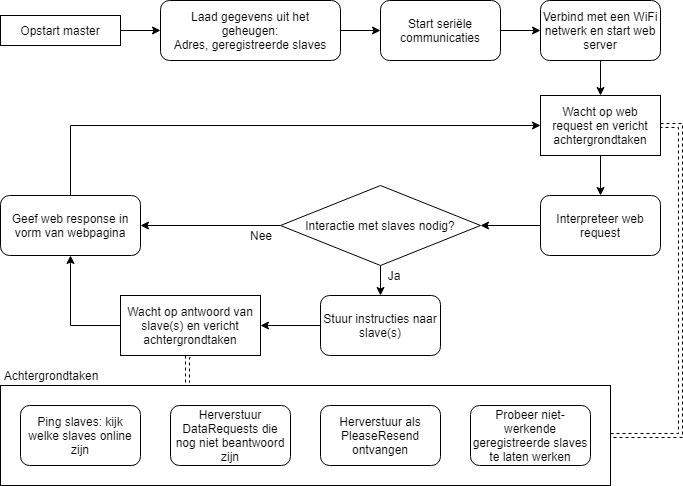
Hoe dit in zijn werk gaat, leg ik verder uit in het deel berekeningen bij het protocol.

Natuurlijk verschilt het programma van de master met het programma van de slave. Hieronder leg ik uit hoe het werkt bij elk.

### Bij de ‘master’

De master heeft het meest complexe programma van allemaal. Alleen de master kan verbinden met het internet en commando’s ontvangen van de smartphone app. De master start een seriële connectie met zijn HC12 (module die antenne aanspreekt, zie componentenlijst), die dan berichten kan doorgeven naar andere antennes. De berichten zijn zogenaamde datapakketten die kunnen omgezet worden tot een lijst van bytes (zie je nog bij protocol berekeningen). De master gebruikt deze pakketten om commando’s te sturen naar zijn geregistreerde slaves.

Het werkingsprincipe van de master is hier voorgesteld in deze flowchart:



Ik kan jammer genoeg niet het programma lijn voor lijn uitleggen, dit zal even duren want het master programma bevat zowat 1000 lijnen code (zonder library). Al mijn code staat op GitHub (zie bronnenlijst).

### Bij de ‘slave’

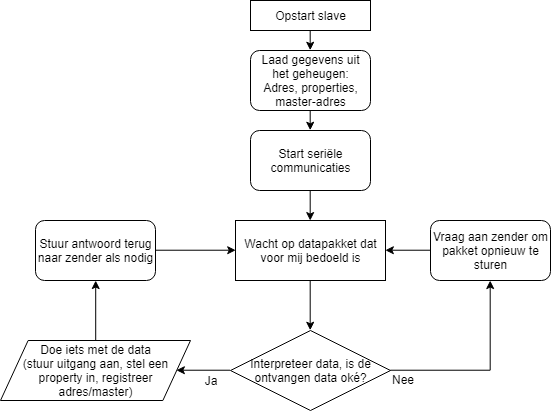
Bij de slave wordt er niet veel meer dan gebruik gemaakt dan de gedeelde library zelf, omdat deze ook al veel achtergrondberekeningen uitvoert. Elke slave implementeert zijn eigen praktische uitwerking van de commando’s dat hij ontvangt, hier een paar voorbeelden van commando’s:

* Registreer (bind): de slave bindt zich aan het meegestuurde adres als de meegestuurde fabriekscode matched met de fabriekscode van de slave.
* Registratie ongedaan maken (unbind): de slave ontbinden van een adres, het adres wordt op 0 ingesteld en verwijderd uit het geheugen van de master.
* Ping (ping): een commando waar de slave altijd ‘oke’ (code 0xFF) op antwoord, dit is om te checken of een slave online is.
* Instellen (prop): Dit stelt een ‘property’ of ‘properties’ in bij de slave. Zie ‘property-principe’.

Property-principe

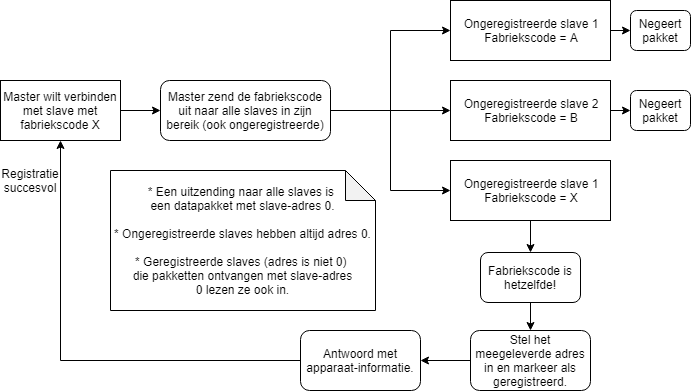
Elke slave heeft een lijst met 256 bytes (of ‘properties’), de master kan een instructie sturen om deze bytes te manipuleren. Hiermee kan dan elke individuele slave deze ‘properties’ gebruiken op een praktische manier. Voorbeeld: byte 1 is de rood-waarde van de ledstrip die een slave aanstuurt, byte 2 de groen-waarde en byte 3 de blauw-waarde.

Net zoals de master heb ik een versimpelde visuele voorstelling van de werking in een flowchart:



### Protocol berekeningen

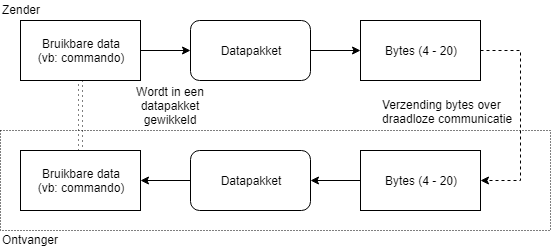
Bij het protocol komen veel wiskundige berekeningen kijken, om iets te versturen moet er een lijst van bytes worden doorgestuurd naar de HC12-module (zie componentenlijst), deze module stuurt een antenne aan en verstuurd die bytes naar alle andere HC12 modules en hun antennes. Elke byte die de zender verstuurd, ontvangen alle ontvangers. Er moet dus een systeem uitgevogeld worden om elke slave te onderscheiden. Hiervoor heb ik de ‘fabriekscode’ en het ‘adres’ ingevoerd: de fabriekscode is een ‘global unique id’ (<https://nl.wikipedia.org/wiki/Globally_unique_identifier>), deze code is voor elke slave op de aardbol uniek, deze code is 7 bytes groot en heeft dus ongeveer 2557 combinaties (= ~7 \* 1016 combinaties, de kans op een duplicaat is dus enorm klein). Het ‘adres’ heeft een waarde van minimum 1 en maximum 63. Elke slave heeft een standaard adres 0, alle data naar het adres 0 wordt door elke slave gelezen. Om een slave te registreren, stuurt de master een datapakket met ontvanger 0 (wordt dus door elke slave opgenomen) en met een specifieke fabriekscode en adres, enkel de slave die dezelfde fabriekscode heeft als de ontvangen data, geeft een antwoord terug en stelt zijn adres in naar het ontvangen adres, hier een visuele voorstelling van dit proces:



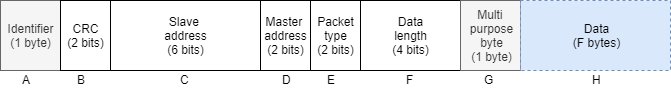
Nadat alle slaves geregistreerd zijn kan de master ze allemaal apart bereiken, elke slave ontvangt alleen een datapakket waarin hun adres zit bijgevoegd, anders wordt het pakket genegeerd.

Samenstelling datapakket

Datapakketten is een opsomming van bruikbare waardes die kunnen omgezet worden naar bytes en omgekeerd. Deze gebruiken we om dus bruikbare getallen om te zetten naar bytes en te versturen, hier een simpel overzicht:

****

Dit is het basisprincipe van de datatransmissie, nu nemen we een kijkje hoe het datapakket wordt opgebouwd, hier een visuele voorstelling van alle velden in het pakket:



1. Identifier: een byte die het begin van het datapakket aanduid, hiermee kan de ontvanger weten vanaf waar hij moet lezen en dus de ingelezen data kan begrijpen. Dit deel is 1 byte lang (256 combinaties) en heeft in ons geval altijd de waarde 0x69 (105).
2. CRC: ‘Cyclic redundancy check’ (of checksum), dit is een getal die het datapakket zijn inhoud beschrijft in 2 bits (4 combinaties), als er 1 bit wordt verstoord in het pakket, veranderd de code meteen, hiermee kan er gecheckt worden of data is gewijzigd tussen de zender en ontvanger. Als dit zo is, lees dan geen data en vraag aan de zender om het pakket nogmaals te versturen (zie ‘pakkettypes’). Dit is om te voorkomen dat er corrupte en verkeerde data wordt geïnterpreteerd. (<https://en.wikipedia.org/wiki/Cyclic_redundancy_check>)
3. Slave address: Een 6-bit getal (64 combinaties) dat het adres van de slave die aan deze communicatie deelneemt bevat. Er is dus een maximum van 63 slaves per master (slaves met adres 0 tellen niet, deze hebben nog geen adres).
4. Master address: Een 2-bit getal (4 combinaties) dat het adres van de master die aan de communicatie deelneemt bevat (er is dus een maximum van 4 masters in hetzelfde antennebereik). *Opmerking: een slave kan dus onmogelijk naar een slave sturen en een master onmogelijk naar een master omdat het pakket altijd 1 slave-adres en 1 master-adres bevat.*
5. Pakket type: Een 2-bit getal (4 combinaties) dat de soort van het pakket beschrijft, meer hierover in het onderdeel ‘pakket-types’.
6. Data lengte: Een 4-bit getal (16 combinaties) dat de lengte van de komende data aangeeft. Hiermee kan de lengte van elk pakket verschillen: minimum 4 bytes (velden A-G vormen samen 4 bytes) en maximum 20 bytes (4 bytes + F bytes als F = 16).
7. Multi purpose byte:Deze byte (256 combinaties) is wat het zegt, het heeft meerdere mogelijke gebruikssituaties, je kunt het zien als een extra data-byte.
8. Data: Hier bevindt zich de bruikbare data, alles wat moet verwerkt worden aan de ontvanger. De lengte van dit veld wordt bepaald door veld F, dit veld kan ook weggelaten worden.

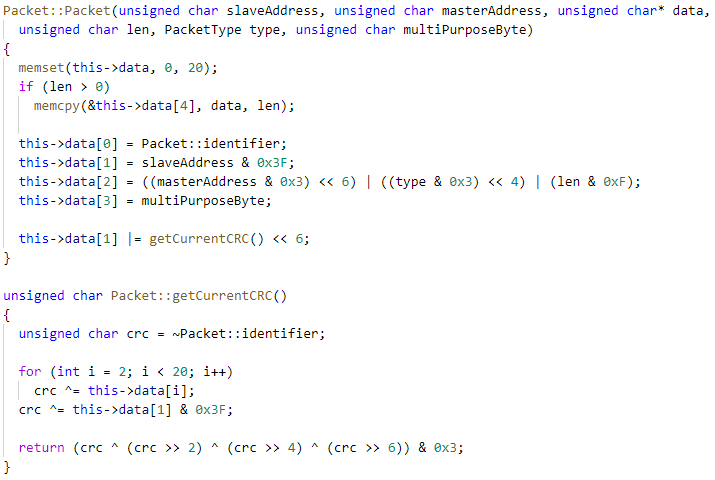
Pakket-types

Dit onderdeel beschrijft het werk van veld E in het datapakket. Er zijn 4 pakket types, elk heeft zijn eigen functie:

* DataRequest: Dit geeft aan dat het pakket een ‘request’ is, het pakket vraagt naar data of naar acties en verwacht altijd een antwoord hierop, om zeker te weten dat er werkt verricht was. Elk slave commando wordt op deze manier verstuurd, om zeker te weten of het commando wel degelijk is uitgevoerd. Wanneer een pakket van dit type wordt verstuurd wordt de ‘multi-purpose-byte’ ingesteld op een code, die later wordt teruggestuurd samen met het antwoord, hiermee weet de master welk verstuurd ‘request’ bij welk antwoord past.
* Push: Dit geeft aan dat het pakket ‘gepushed’ word, het pakket wordt verstuurd en verwacht geen antwoord. Een ontvanger kan er mee doen wat hij wilt.
* Answer: Dit geeft aan dat het pakket een antwoord op een ‘request’ bevat, dit is dus altijd het gevolg van een ‘DataRequest’ pakket. Een antwoord wordt als ‘oké’ beschouwd als de eerste byte van de data gelijk is aan 0xFF (255), elke andere waarde (of geen) markeert de ‘request’ als mislukt.
* PleaseResend: Dit geeft aan dat de ontvanger van dit pakket zijn vorige pakket opnieuw moet sturen, dit kan gebeuren als de checksum (zie veld B) veranderd was.

Praktische uitvoering berekeningen

Deze code neemt verschillende parameters en slaat ze op in een lijst van bytes (data) op deze manier wordt er een datapakket opgesteld. De onderste functie berekent een checksum.



## Uitgebreide uitleg hardware

### Microcontrollers

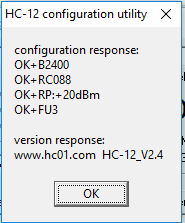
Bij de ontwikkeling van ons systeem zijn er van meerdere microcontrollers gebruik gemaakt, dit is het brein van elke module, je kan het zien als een kleine computer: het stuurt digitale uitgangen aan of leest ze in, sommige kunnen verbinden met het Wi-Fi netwerken, PWM signalen vrijgeven, seriële communicatie ... Er zijn natuurlijk heel wat soorten, hier een overzicht van de controllers die wij gebruikt hebben bij zowel de master en de slave:

|  |  |  |
| --- | --- | --- |
|  | Slave | Master |
| Naam | Arduino Pro Mini (5V type) | ESP8266 module |
| Foto | Afbeeldingsresultaat voor arduino pro mini | Afbeeldingsresultaat voor esp8266 |
| Microprocessor | ATmega168 (@ 16 MHz) | ESP8266 (32-bit low power @ 80MHz) |
| Werkvoltage | 5V | 3.3V |
| Digitale pinnen | 16 (waarvan 6 PWM) | 9 (waarvan 8 PWM) |
| Analoge ingangen | 8 | 1 |
| Flash geheugen grootte | 16 kilobytes | 512 kilobytes |
| EEPROM geheugen grootte | 512 bytes | 4 kilobytes |
| Max uitgang stroom | ~40 mA | ~12 mA |
| Wi-Fi verbinding? | Nee | Ja |
| Nominaal stroomverbruik | ~10 mA | ~100 mA |
| Waarom deze module? | De Arduino Pro Mini is klein, verbruikt niet veel stroom en heeft veel uitgangen, dit is ideaal om een slave mee samen te stellen. | De ESP8266 module heeft een krachtige microprocessor die kan verbinden met het internet, dit was noodzakelijk in ons geval. Het heeft ook een enorm flash geheugen en SRAM geheugen (t.o.v. Arduino). Perfect om complexe programma’s te laten werken. |



### Communicatie

De draadloze communicatie tussen de master en slaves wordt gerealiseerd met een HC12 module (rechts een foto), het is een kleine radiomodule die radiogolven uitstraalt op een frequentieband van 433.4 MHz tot 473.0 MHz. Het genereerd de nodige signalen om een antenne aan te sturen, zodat we niet ons eigen veel te ingewikkeld elektronisch circuit moeten ontwikkelen.

Om een stabiele communicatie te voorzien, moeten we eerst wat met de instellingen van de HC12 prutsen om verschillende eigenschappen in te stellen zoals: kanaal, uitstraalvermogen, seriële datasnelheid, modus … Wij hebben de volgende eigenschappen voor ALLE modules gebruikt:

* 2400 baud transmissiesnelheid (relatief traag maar groter bereik)
* Zend- en ontvang-kanaal 88
* +20 dBm zendvermogen (hoogste)
* Modus FU3: zorgt voor groot bereik en relatief laag stroomgebruik. Beperkt transmissiesnelheid tot 2400 baud; hoge snelheden zijn voor ons niet noodzakelijk.

Om de eigenschappen te versturen naar de HC12, gebruikte we een computerprogramma genaamd ‘HC-12 configuration utility’ samen met een UART (<https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter>; simpelweg: een usb naar HC12 zodat we deze aan de computer konden aansluiten).

Document over HC12 eigenschappen: <https://www.elecrow.com/download/HC-12.pdf>

De HC12 is compatibel met verschillende antennes, een spiraalantenne (foto rechts) of een andere antenne met coaxiale aansluiting. We kregen van de leerkracht grotere coaxiale antennes (foto links), deze zouden voor een groter omnidirectionaal (<https://en.wikipedia.org/wiki/Omnidirectional_antenna>; een antenne die in alle richtingen golven uitstuurt, de antenne heeft geen zendrichting) bereik kunnen zorgen. Later zijn we tot de conclusie gekomen dat deze bijna geen verschil geven tegenover de spiraalantennes, na een paar uren zoekwerk is de oorzaak hiervoor is nog altijd een mysterie.

# Elektrische schema’s

Nu dat er over alle software is nagedacht, gaan we over naar het hardware deel.

### Prototypes

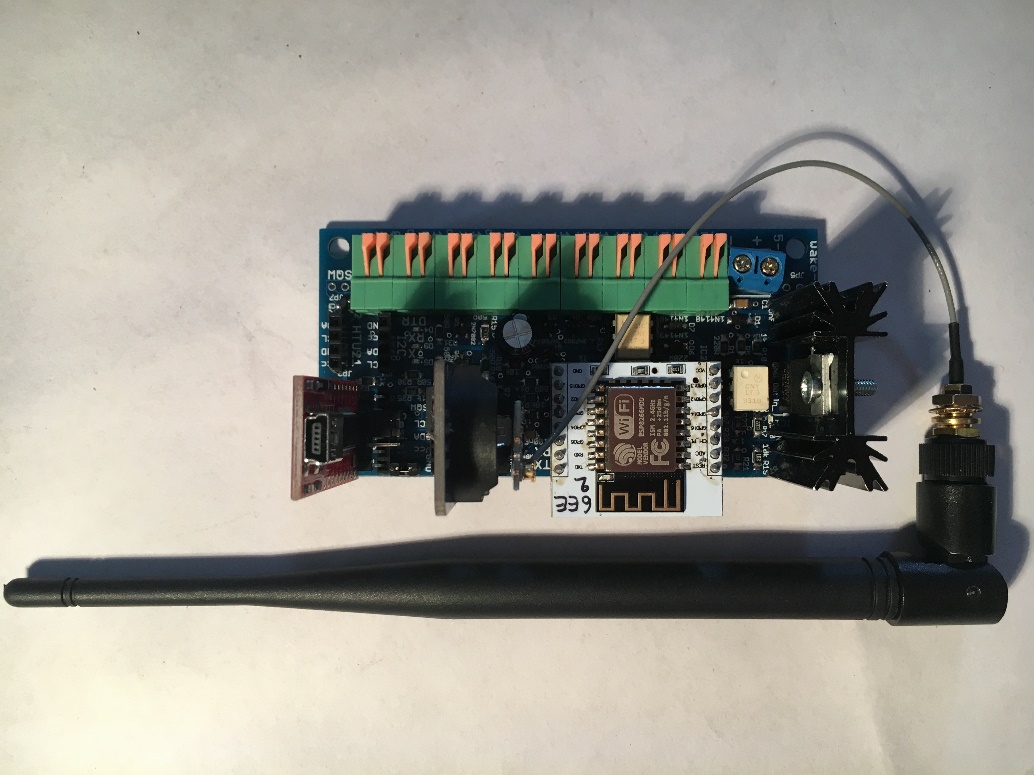
Het moet ergens beginnen, we kregen een paar HC12 modules en antennes en we konden starten met experimenteren. De eerste succesvolle test was al aanwezig in het eerste uur na het ontvangen van de componenten: Een knop dat een Arduino Nano aanstuurde om een signaal te versturen naar een andere Arduino Nano die dan een ledje aanstuurde. De problemen kwamen wanneer een groter bereik werd ingevoerd, soms moest er 2 keer op de knop gedrukt worden voordat er een signaal werd opgevangen. Dit was de motivatie om een vraag-antwoord (‘requests’) systeem in te voeren.

Sommige prototypes zijn gemaakt op een ‘breadboard’en sommige op ‘perfboard’.

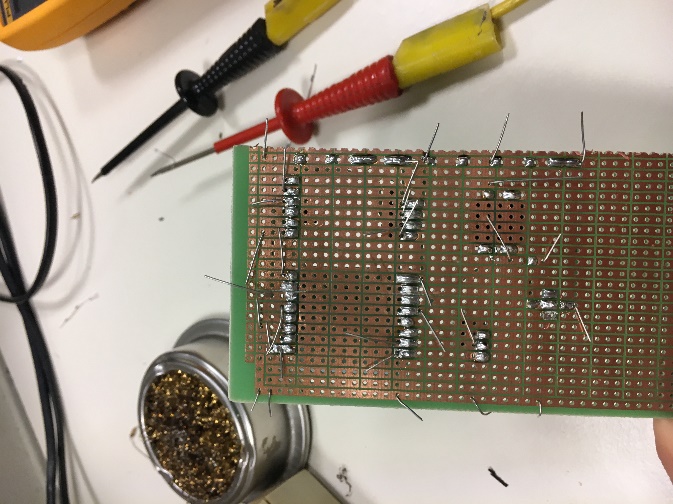
Foto’s van de prototypes

We testen de master software eerst via het bordje die de leerkracht ons gegeven had (dankuwel Erik Arckens), het was makkelijk om naar te uploaden, maar we zijn later nog overgegaan naar andere bordjes omdat de uitgangen niet altijd even ideaal waren voor ons gebruik en omdat dit maar een tijdelijke oplossing was.

Bordje leerkracht:

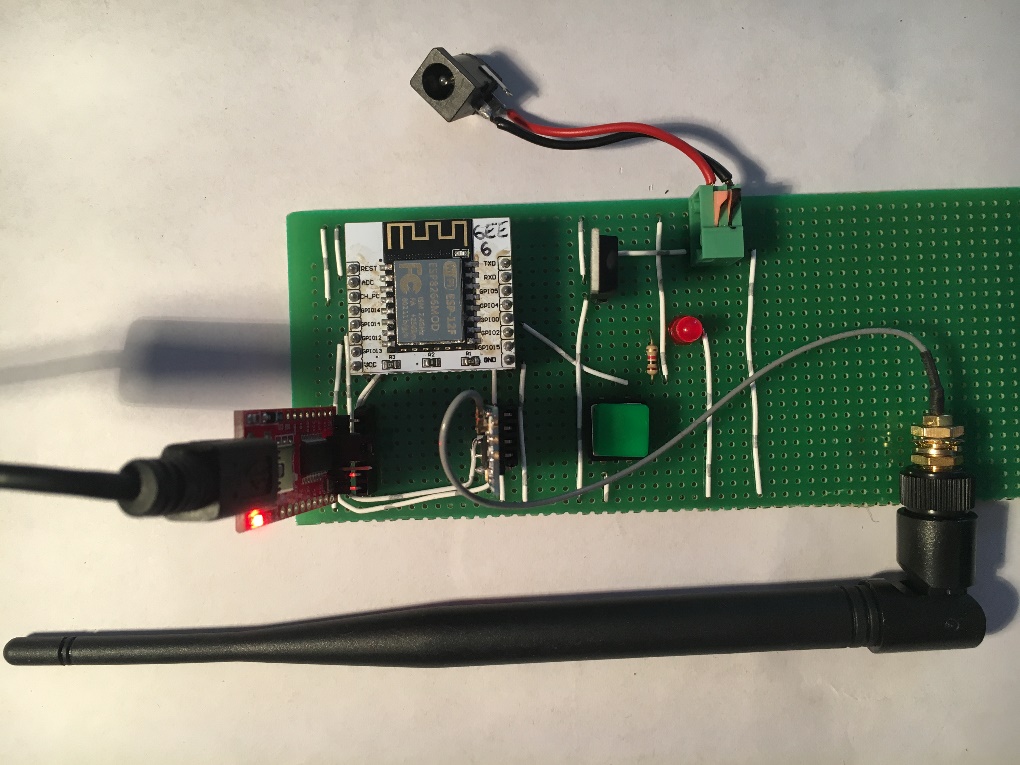


Hier werd er een testbordje gemaakt voor de master module:

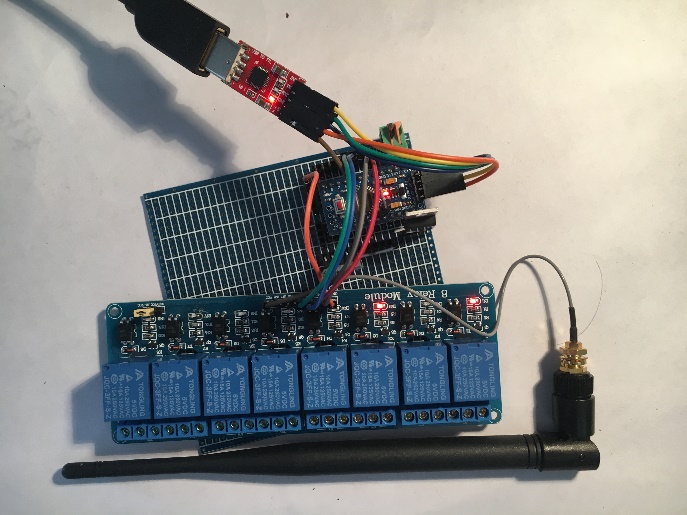


Het is een simpel bordje met een aan-uit-ledje, een programmeer knop (om de controller in programmeermodus op te starten), header voor een HC12 module, een header voor de FTDI en een header voor de ‘voltage regulator’. (Zie componentenlijst) We wilden dit maken op een ‘perfboard’ in plaat van op een ‘breadboard’ om storing met de antenne en slechte contacten te vermijden.

Hier is het afgewerkte master prototype op ’purfboard’:



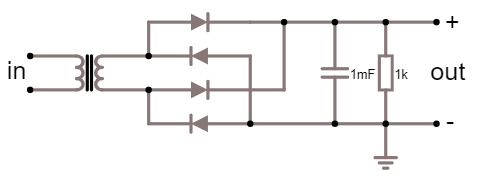
Dit zijn de afgewerkte slave prototypes:

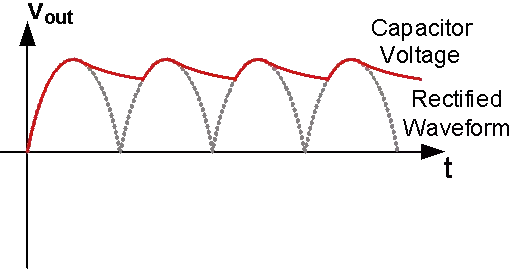


Het eerste slave prototype is gemaakt op een ‘breadboard’ en stuurt 4 leds aan, het maakt gebruik van een Arduino Nano. Het tweede slave prototype is gemaakt op een ‘perfboard’ en stuurt maximaal 8 relays (contacten) aan, het maakt gebruik van een Arduino Pro Mini.

### Stroommeting

De slaves zijn ook in staat om elektronische signalen in te lezen, dus hier wilden we ook zeker gebruik van maken. We hebben gezorgd door een systeem dat stoommetingen uitvoert, deze waardes kunnen dan gelezen worden door de master. Om dit te realiseren hebben we het volgende schema gebruikt:

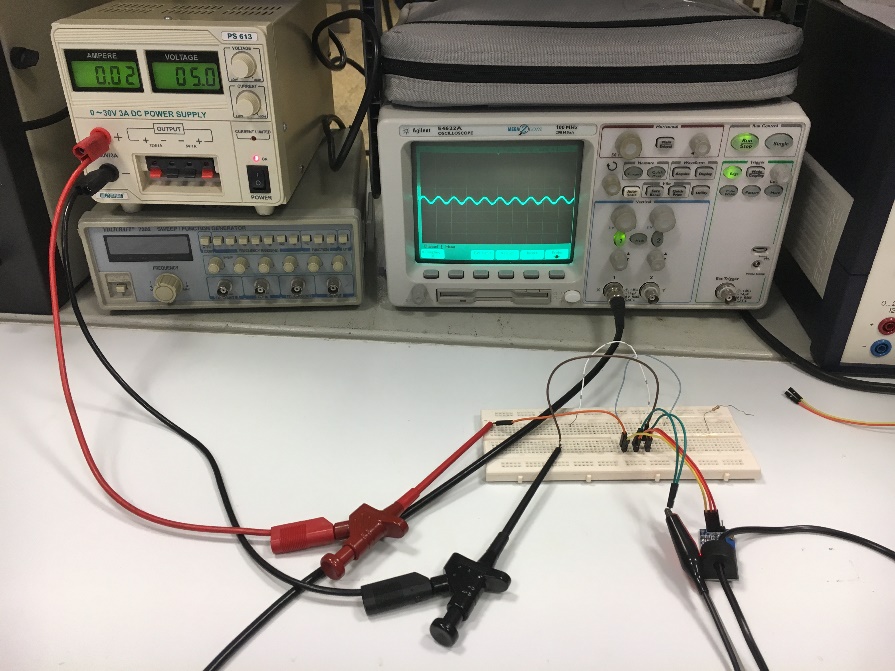


De belasting zetten we in serie met ingang van dit circuit en we krijgen een opgewekte spanning over de uitgang die dan ingelezen kan worden door een microcontroller. De transformator is heeft een ratio van 1 op 1000, dit betekend dat elke ampère over het net 1 milliampère induceert in ons circuit, deze stroom wordt hierna gelijkgericht van wisselspanning naar gelijkspanning. De condensator van 1 mF zorgt voor een agressieve afvlakking van het signaal (we willen zo min mogelijk rimpel op ons uitgangsignaal, anders wordt inlezen met de microcontroller een ramp). De spanning over de weestand wordt gemeten door de microcontroller via een analoge pin. (1 mA zorgt voor 1V over de weestand als de weestand 1 kilo ohm bedraagt)



Hier werd de stroommeting getest met een wisselspanningsbron en een gloeilamp, hiermee konden we onze microcontroller kalibreren door ook te meten met de nauwkeurige stoommeter. Onze waarnemingen:

|  |  |
| --- | --- |
| TODO |  |
|  |  |
|  |  |



### Componentenlijst

Na alle prototypes waren tot de conclusie dat we de volgende elektronische componenten gingen gebruiken in ons eindresultaat.

### Professioneel PCB

todo: gemaakt met eagle

we hebben 2 versies gehad

Todo: hongkong

# Design en afwerking

## Smartphone app

Todo

## Behuizing

Todo: 3d model, afmetingen

# Bronnenlijst

Hieronder een concrete lijst van internetbronnen die ik heb gebruikt bij het werken aan dit project.

* <https://tttapa.github.io/ESP8266/Chap10%20-%20Simple%20Web%20Server.html>
* <https://www.arduino.cc/en/Reference/EEPROM>
* <http://www.cplusplus.com/doc/tutorial/pointers/>
* <https://www.arduino.cc/en/Reference/softwareSerial>
* <https://stackoverflow.com/questions/3698043/static-variables-in-c>
* <https://randomnerdtutorials.com/esp8266-web-server/>
* <http://arduino.esp8266.com/stable/package_esp8266com_index.json>
* <https://en.wikipedia.org/wiki/Multicast_DNS>
* <https://en.wikipedia.org/wiki/Cyclic_redundancy_check#CRC-32_algorithm>
* <https://www.elecrow.com/download/HC-12.pdf>
* <https://en.wikipedia.org/wiki/Baud>
* <https://nl.wikipedia.org/wiki/Globally_unique_identifier>
* <https://en.wikipedia.org/wiki/Omnidirectional_antenna>
* <https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter>

Ik schrijf natuurlijk niet alle links op die ik bezocht heb, dat zou enorme tijdverlies zijn en het zou niemand interesseren, dit waren enkele belangrijke links die ik had genoteerd.

Alle flowcharts zijn door mij gemaakt met behulp van [https://www.draw.io](https://www.draw.io/).

Alle mijn bestanden (code, documentaties, presentaties, flowcharts, schema’s …) die ik heb gebruikt doorheen het jaar en zowel in deze bundel, staan op **GitHub** (auteursrechten zijn actief): <https://github.com/CodeStix/Home-Control-GIP>

# Logboek

Hier logboek invoegen

# Besluit en zelfreflectie

todo

# Bijlagen