2016

# Scriptie
# SMMS Project

**Afstudeerstage VMB Automation** **bestemt voor: Gecommitteerde**

| Hand tekening voor gezien: |
| --- |
| Naam: |
| Datum: |

**10059490**
**JORIS GROEN**
**25 AUGUSTUS 2016**

**HAAGSE HOGENSCHOOL | VMB besturingstechniek**

## Voorwoord

Beste lezer van deze scriptie. Deze scriptie heb ik geschreven tijdens mijn afstuderen bij VMB Automation. Tijdens mijn afstuderen heb ik gewerkt aan het Ship monitoring en management system. Hiervoor heb ik de software ontworpen. In dit verslag wordt ingegaan op de ontwerpstappen die ik doorlopen heb voor het opstellen van de software. Enige kennis van C programmeren is vereist voor het volledig begrijpen van deze scriptie. Enige bronvermelding wordt cursief en onderstreept en tussen haakjes weergegeven *(Bron)*

Ik wil graag Arie Verhoeven bedanken voor het bieden van de mogelijkheid om mijn afstudeerstage bij VMB te lopen. Daarnaast wil ik graag mijn collega's op de programmeerafdeling bedanken voor hun begeleiding en ondersteuning.

Ik wens de lezer veel plezier met het lezen van deze scriptie.

Met vriendelijke groet,

Joris Groen
Afstudeerder Elektrotechniek

## Woordenlijst

SMMS: Ship Monitoring en Management systeem
CM: Control module
EM: Equipment module
HCM: Hardware control Module

**Controller**

- DI: Digitale Input
- DO: Digitale Output
- AI: Analoge Input
- AO: Analoge Output
- IO: Inputs en Outputs. Een verzameling van alle DI, DO, AI en AO.

**Software termen**

- programma cycli: een programma cycli is als de syel controller 1 keer de RUN functie of bij het opstarten de INIT functie in het hoofdprogramma doorlopen heeft.
- FIFO: First in First out. Een buffer die FIFO gevuld is betekent dat het eerste binnengekomen element als eerste uit de buffer wordt gelezen. [1]

**Datatypen**

| Type | Size | Range |
|---|---|---|
| unsigned char byt(byte) | 8 bits | 0 to 255 |
| char | 8 bits | -128 to 127 |
| short int | 16 bits | -32,768 to 32,767 |
| unsigned int | 32 bits | 0 to 4,294,967,295 |
| int | 32 bits | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 32 bits | 0 to 4,294,967,295 |
| long | 32 bits | -2,147,483,648 to 2,147,483,647 |
| float | 32 bits | 1.17549435 * (10^-38) to 3.40282347 * (10^+38) |
| double | 64 bits | 2.2250738585072014 * (10^-308) to 1.7976931348623157 * (10^+308) |
| long double | 80 bits | 3.4 * (10^-4932) to 1.1 * (10^4932) |

[1] Voor meer info over FIFO: https://nl.wikipedia.org/wiki/Fifo

## Samenvatting

This report describes the development software of the Ship Monitoring and Management system.

The Transport sector is one of the biggest sectors in the Netherlands. Products are transported by road, rail and river. Most of the sectors in the transport sector try to reduce the environmental footprint.

However the inland ship transport sector is less innovative compared to the freight truck transport sector. This is partly because ship have a longer lifespan then trucks. Also the inland ship transport sector is known for its conservative nature. Because the ships have a longer live span the shipbuilders tend to choose for reliable but les innovative technical designs. This means that the captain only roughly knows what the most efficient engine speed is. Also the lubricating oil of the ship is changed based on the operation hours of the engine not on the quality of the oil. This leads to a waste of oil and fuel and higher maintenance on the engine.

Because of that the ships that transport goods these days have a big environmental footprint. Also the fuel and oil cost for the ship owners are higher than the could be. An system the measures the engine condition, the fuel consumption and the oil condition could solve this problem.

This is where the SMMS project comes in. The goal of the SMMS project is to develop that system. The goal of the thesis was to develop the software for the controllers that read the sensors measuring the fuel consumption an oil condition. The controllers also had to read the engine condition using a canbus J1939.

The following software components were developed during the thesis:

- Modules to manage the hardware of the controllers that have to read the sensors and engine data
- Modules to read the sensors an calculate the value of the flowrate etc.
- Modules to calculate the fuel consumption

After that the foundation of the web application was developed. The web application can read data that is send form the controllers and shod the data in text form. Also part of the function to store data in the database was developed.

Before the final presentation the following modules will be finished

- Storing data in a database

Before the final presentation the following modules might be finished

- A fully functional web application

VMB AUTOMATION

# Inhoud


Voorwoord ..... ii
Woordenlijst ..... iii
Samenvatting ..... iv
1 Probleemstelling (situatieschets) ..... 1
2 Afstudeeropdracht ..... 1
3 Afstudeerbedrijf ..... 2
4 Aanpak en projectgrenzen ..... 3
5 Eisen en wensen ..... 4
5.1 Eisen aan de hardware ..... 4
5.2 Eisen aan de software ..... 5
5.3 Eisen aan de Infrastructuur (netwerk) ..... 5
6 Op te leveren producten ..... 7
7 Opbouw hardware ..... 7
8 Opbouw software ..... 10
8.1 S88.01 ..... 10
8.2 Implementatie S88.01 ..... 12
8.3 Controle cycli ..... 14
9 Software Units ..... 15
9.1 HCM ..... 16
9.1.1 HCM IO ..... 16
9.1.2 HCM serieel ..... 19
9.1.3 HCM RSModbus ..... 20
9.1.4 HCM TCP/IP ..... 24
9.1.5 HCM CAN ..... 27
9.2 Control modules ..... 29
9.2.1 CM Flow sensor ..... 29
9.2.2 CM Pressure sensor ..... 32
9.2.3 CM Oil quality sensor ..... 34
9.2.4 CM J1939 ..... 36
9.2.5 CM Intern Com Send ..... 39
9.2.6 CM Intern Com Rec ..... 40
9.3 Equipment modules ..... 42
9.3.1 EM Fuel Consumption ..... 42
10 Webapplicatie ..... 44
10.1 Uitlezen van de serieele poort ..... 45
10.2 Weergeven van de data ..... 45

10.3 Lezen uit de database ..... 45
10.4 Schrijven naar data base en visualiseren ..... 45
11 Eind test ..... 46
12 Conclusie en aanbevelingen ..... 47
13 Bibliografie ..... 48
14 Bijlagen ..... I
14.1 Bijlage voorbeeld berekening Puls vs. Analoog dieselverbruik ..... III
14.2 Bijlage Test rapporten HCM, CM en EM ..... V
14.3 Bijlage Flowcharts HCM, CM en EM ..... VII
14.4 Bijlage vooranalyse ..... IX
14.5 Bijlage Wireless Communication onderzoek ..... XI
14.6 Bijlage VPN onderzoek ..... XIII
14.7 Bijlage broncode ..... XV
14.8 Bijlage VSB hoofdstuk 5 Software ..... XVII

# 1 Probleemstelling (situatieschets)

De Rotterdamse haven is de grootste haven van Europa. Miljoenen tonnen aan natte en droge bulkgoederen worden overgeslagen in de haven. Het grootste gedeelte van deze goederen worden verder Europa ingevoerd door binnenvaartschepen. Dit maakt de binnenwateren van Nederland een druk bevaart gebied door de professionele vaart en de pleziervaart. Dit brengt veel economische welvaart, maar heeft ook nadelige gevolgen voor het milieu door de CO2 uitstoot van de zware dieselmotoren. De meeste transportsectoren doen veel om hun CO2 afdruk en hun brandstofverbruik/kosten zo laag mogelijk te houden.

De scheepvaart sector blijft hier in achter doordat o.a. de levensduur van schepen met 20 tot 35 jaar, langer is dan die van bijvoorbeeld een vrachtwagen (12 tot 15 jaar). Hierdoor duurt het langer voordat nieuwe en schonere motoren algemeen gebruikt worden in de scheepvaart. De schipper moet zelf inschatten op welk toerental de motor het efficiëntste draait. De olie wordt op basis van het aantal bedrijfsuren van de motor ververst, maar de kwaliteit van de olie is van meer factoren afhankelijk. Hierdoor wordt soms te schone olie te vroeg vervangen of vieze olie te laat.

Dit alles resulteert in onnodig brandstof verbruik, onnodig olieverbruik, een kortere levensduur van de motor, hogere kosten voor de schipper en een onnodige aanslag op het milieu. Er zijn systemen die de oliekwaliteit of het brandstofverbruik of de motorcoditie meten. Een systeem die al deze gegevens meet en ze vervolgens kan combineren en m.b.v. deze gegevens o.a. het vervangmoment van de olie aangeeft bestaan nog niet of zijn nog in ontwikkeling. Het bedrijf VMB Automation wil graag zo'n systeem ontwikkelen en wil dit laten doen door een afstudeerder.

# 2 Afstudeeropdracht

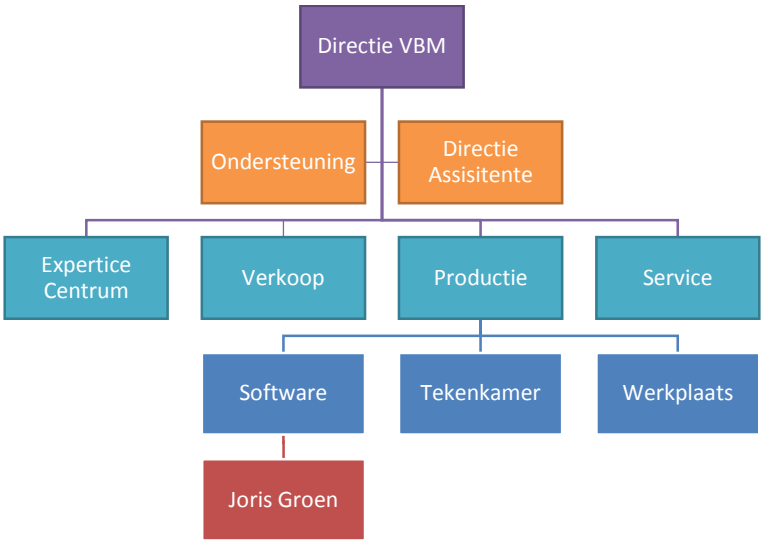
Het systeem bestaat uit sensoren en andere apparatuur die het brandstof verbruik, oliekwaliteit of motorkarakteristieken meten. Deze sensoren en apparatuur moet uitgelezen worden door verschillende controllers zogenaamde slave controllers. Deze slave controllers versturen de verzamelde data naar een centrale controller. De centrale controller stuurt de data naar een pc waar de data met een webapplicatie gevisualiseerd wordt.

Doel van de afstudeeropdracht is het schrijven van de software voor de controllers. Daarnaast moet de webapplicatie ontwikkeld worden. Het systeem voor het meten van het brandstofverbruik en de monitoren van de motorkarakteristieken zullen het standaardsysteem vormen. Op de standaard module kunnen dan de later de te ontwikkelen optionele systemen aangesloten worden. Bijvoorbeeld het systeem voor het meten van de oliekwaliteit.

De analyse fase van het project is grotendeels voorbij. In de weken voor de afstudeerperiode is al onderzoek gedaan naar de technische haalbaarheid van de verschillende systemen, maar er zijn nog geen ontwerptekeningen of programma's gemaakt. Het is nu de bedoeling dat er aan de hand van het analyse document, aangevuld met eigen bevindingen de standaard module ontworpen wordt. Verder is het de bedoeling dat de benodigde infrastructuur (data overdracht en stroomvoorziening) uitgezocht wordt. Het is de bedoeling dat deze infrastructuur zonder grote veranderingen(gaten boren e.d.) op het schip geplaats kan worden.

# 3 Afstudeerbedrijf

VMB is het afstudeerbedrijf waarvoor de afstudeeropdracht wordt uitgevoerd. VMB is een bedrijf dat zich heeft gespecialiseerd in de besturingstechniek. VMB is vooral werkzaam in de food industrie, productie industrie en de petrochemie/waterzuivering. VMB bied zijn klanten een totaal oplossing op het gebied van de besturingstechniek. D.w.z. dat de klant na het geven van de opdracht niet zelf de regie hoeft te voeren over het project.




**Figuur 3.1: Organogram VMB**

**Werkwijze**

VMB doorloopt zijn projecten volgens een vast stramien:

- Contact met de klant
- Analyse fase & Technische ontwerp
- Ontwikkelfase
- Testfase (FAT)
- Oplevering bij de klant (SAT)

Als eerste nemen de commercieel verkopers contact op met de klant. Als er een project voorhanden is gaat de technische verkoper langs bij de klant voor een eerste gesprek. Hier worden de eisen, wensen en grenzen (wat wil de klant niet) besproken. Zeker bij grotere project en bij R&D projecten zal er een analyse rapport geschreven worden. Dit wordt gedaan door de software en hardware engineers op basis van de gegevens van het gesprek van de technische verkoper. Als er uit het analyse rapport blijkt dat het project haalbaar is en de klant gaat akkoord dan wordt de ontwikkelfase opgestart. De hardware en software engineers ontwerpen de installatie volgens de ontwikkelmethode die zij in het analyserapport beschreven hebben. De installatie wordt doorontwikkeld totdat de installatie aan de eisen en wensen van de klant voldoet. Hierna volgt er een FAT test om te bewijzen dat de installatie aan de eisen en wensen van klant voldoet. Als de FAT test door de klant is goedgekeurd wordt de installatie geïnstalleerd bij de klant en wordt er een SAT gedaan om te zien of de installatie werkt bij de klant.

## 4 Aanpak en projectgrenzen

Na het inventariseren van de opdracht en het lezen van het analyse document moet onderzocht worden hoe de sensoren e.d. op de controllers wordt aangesloten. Er wordt gekeken hoe verschillende controllers met elkaar gaan communiceren en er wordt een inventarisatie gemaakt van de signalen en databussen die de controller moet kunnen verwerken. Op basis daarvan wordt bekeken welke controller er gebruikt gaat worden.

De software zal worden opgebouwd aan de hand van het S88.01. model. Dit is het standaard programeer model dat gebruikt wordt bij VMB Automation. Met dit model wordt het software programma opgesplitst in verschillende modules. Hierna wordt bekeken met welke programmeer omgeving er gewerkt gaat worden en van welke ingebouwde functies er gebruik gemaakt kan worden.

Op basis van deze gegevens zal bepaald wordt hoe de het software modules geïmplementeerd worden op basis van het S88.01. model.

Voor het programmeren is tijdens het afstudeer project gekozen om de software modules een voor een te ontwikkelen, van onderste naar hogere functionaliteit. Hierbij zal gestart worden met de hardware control modules die direct de hardware zullen besturen (IO data bussen e.d.). Daarna zullen de control modules ontwikkeld worden die de sensoren aangesloten op de hardware zullen besturen. Als laatst zullen equipment modules die bewerkingen moeten uitvoeren met behulp van control modules ontwikkeld worden.

Het ontwikkelen van de modules een voor een van onderste naar bovenste functionaliteit a.d.h.v. het S88.01. model levert de volgende voordelen op:

- Door de software in modules op te delen in modules wordt het software programma overzichtelijk. Hierdoor kunnen de modules onafhankelijk van elkaar ontwikkeld worden. Er kan dan makkelijk gewisseld tussen de ontwikkeling van modules op het zelfde niveau (HCM, CM of EM).
- De fundatie van het systeem (de HCM) worden als eerst ontwikkeld. Hierdoor kunnen CM en EM ontwikkeld worden met de beperkte/extra functionaliteit van de HCM in het achterhoofd.

Nadeel is dat sommige functionaliteit van een HCM pas nodig blijkt bij het ontwikkelen van een CM die de HCM gebruikt. Deze functionaliteit moet dan later in de HCM worden toegevoegd.

Ieder ontwikkelde module wordt getest op functionaliteit. De testrapporten zijn te vinden in de 14.2 Bijlage Test rapporten HCM, CM en EM.

Na het ontwikkelen van de software zal op een pc m.b.v. een webapplicatie de data gevisualiseerd worden. De webpagina wordt opgebouwd m.b.v. HTML, CSS, Javascript.

Er zal bekeken moeten worden op welke manier de data gevisualiseerd kunnen worden. Er zijn op het internet veel librarys te vinden waarmee m.b.v. javascript data gevisualiseerd kan worden m.b.v. meters en grafieken. Er zal een library gekozen moeten worden om de data te visualiseren. De uiteindelijke keuze van de library behoort niet tot de afstudeeropdracht. Het visuele gedeelte van de webapplicatie dient puur als demoversie.

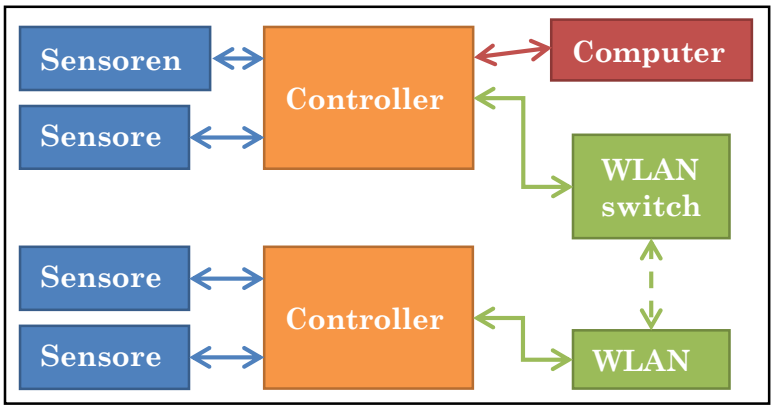
Daarna wordt het volledige systeem nog eenmaal getest en wordt gekeken of de data van de controller aankomt op de website en of de website de data opslaat in de database. Het testen van het systeem op een schip behoort niet tot de afstudeeropdracht.

# 5 Eisen en wensen

Aangezien het SMMS gebruikt gaat worden in de scheepvaart moeten de apparatuur geschikt zijn om in een industriële omgeving te gebruiken. Ook zullen de gewenste parameters van de motor, dieselverbruik e.d. gemeten moeten worden.

## 5.1 Eisen aan de hardware

De hardware bestaat uit een aantal flow, temperatuur en druk sensoren die uitgelezen moeten worden m.b.v. een controller. Daarnaast zal de controller de motorgegevens uitlezen. Dit kan bij oude motoren analoog, maar bij nieuwere motoren gebeurt dit meestal met een databus. Het systeem kan bestaan uit één of meerdere controllers. Deze moeten met elkaar kunnen communiceren zodat ze gegevens uit kunnen wisselen. In de stuurhut zal m.b.v. een computer de data gevisualiseerd worden en weergegeven worden m.b.v. een web applicatie op het beeldscherm. Een van de controllers zal de data van de sensoren uitgelezen door de andere controllers moeten versturen naar de computer.




**Figuur 5.1: Hardware principe schema**

**Sensoren**
De sensoren moeten van geschikt zijn om op een schip te gebruiken. Zij moeten tegen trillingen kunnen, tegen vuile omgevingen en moeten bijna onderhoudsvrij kunnen functioneren.

**Controller**
De controller moet genoeg IO mogelijkheden hebben voor het aansluiten van sensoren en databussen. Dit betekent dat de controller op zijn minst een CAN bus, RS232, RS485 en een ethernet aansluiting moet hebben. De controller moet al algemeen toegepast worden binnen VMB.

**Computer**
De PC mag een desktop pc zijn of een laptop. De pc/laptop moet de laatste versies van Internet explorer, chrome of modzilla firefox kunnen draaien. Daarnaast moet de pc genoeg geheugen hebben om de data, die gelogd wordt door de webapplicatie, op te slaan. Geheugenruimte is voor de testversie van het SMMS die tijdens deze afstudeerperiode ontwikkeld wordt nog niet belangrijk, aangezien de testversie niet jaren echter elkaar data zal loggen.

**Interne communicatie**
De apparatuur voor de communicatie moet net als de sensorengeschikt zijn om op een schip te gebruiken. Dit betekent dat zij jaren probleemloos moeten kunnen werken in een warme en vervuilde omgeving. De communicatie installatie moet eenvoudig en i.v.m.

brand veiligheid en waterdichtheid van de waterdicht compartimenten in het schip, met zo min mogelijk boorgaten geïnstalleerd kunnen worden. Dit houd in dat er zo veel mogelijk draadloos gecommuniceerd moet worden. De communicatie installatie moet makkelijk uitgebreid kunnen worden.

## 5.2 Eisen aan de software

De geschreven software moet overzichtelijk en makkelijk te begrijpen zijn voor de software programmeurs van VMB. De software moet makkelijk te onderhouden zijn en moet modulair worden opgebouwd. Op deze manier zijn losse onderdelen van de software ook op zich zelf te gebruiken. De software moet uitbreidbaarheid zijn, hierdoor kan het SMMS toegepast worden op schepen waar 1 motor uitgelezen moet worden, maar moet het zonder grote functionele aanpassingen mogelijk zijn om het SMMS toe te passen op schepen waar 3 of 4 motoren uitgelezen moeten worden.

In hoofdstuk 4 Aanpak en projectgrenzen is aangegeven dat bij het ontwerpen de software modules los van elkaar worden ontwikkeld. Tijdens elke ontwerp ronde zal een software modulen ontworpen worden. Om te bepalen welke software modules als eerst geïmplementeerd moeten worden is een MOSCOW model opgesteld zie Figuur 5.2. Het MOSCOW model geeft snel inzicht in welke eisen af moeten, af mogen en welke eisen geïmplementeerd kunnen worden in de toekomst of als er nog tijd over is[1]. Een project wordt gezien als gefaald als de eisen die af moeten niet geïmplementeerd zijn.

## 5.3 Eisen aan de Infrastructuur (netwerk)

De communicatie tussen de verschillende syel controllers moet betrouwbaar en gegarandeerd zijn.

Het netwerk waarmee de controllers onderling communiceren moet beveiligd zijn tegen invloeden van buitenaf. Hierbij moet wel rekening gehouden worden met het feit dat de databussen aangesloten op de controllers op TCP/IP na geen betrouwbare beveiliging hebben tegen invloeden van buiten af. Men zou de canbus van de ECU van de motor kunnen aansluiten op een eigen apparaat en kunnen voordoen dat de motor heel zuinig werkt. Zonder dat een ander systeem die door heeft. Als men op het schip aanwezig is kan men al invloed uitoefenen op de gemeten gegevens door de sensoren te beïnvloeden. Als men op het schip aanwezig is hoeft men dus niet het netwerk van de syel controllers te hacken. Daarom heeft beveiliging hier tegen geen prioriteit.

Wel moet het netwerk van de syel controllers beveiligd worden tegen wireless aanvallen van buiten af. Dit betekent dat het wireless netwerk van de syel controllers alleen gebruikt mag worden voor de syel controllers en niet bereikbaar mag zijn voor wireless clients (laptops e.d.).

[1] (*MoSCoW method, sd*)

## Must have

- HCM
  - Besturen van IO (AI, DI, DO en AO)
  - Besturen van serieele poort (RS232/RS485. Moet als transportkanaal kunnen dienen voor de externe communicatie
  - Besturen van RS485. Moet als transportkanaal kunnen dienen voor de modbus.
- Besturen van Ethernet controller (TCP/IP). Moet snel genoeg werken om als transportkanaal te dienen voor de interne communicatie
  - Besturen CAN bus. Deze HCM moet in staat zijn om alle J1939 data te ontvangen en plaatsen in een ontvangstbuffer
- CM
  - Analoog en m.b.v. puls signaal een flow sensor uitlezen
  - Uitlezen van J1939 data van de motor m.b.v. canbus
  - Interne communicatie module die data van verschillende syel controllers kan verzenden naar een centrale syel controller die verbonden is met het HMI (PC) in de stuurhut
  - Externe communicatie module voor het versturen van data in de centrale syel controller naar de HMI (PC).
- EM
  - Meten dieselverbuik van de motor m.b.v. flowsensoren
  - Monitoren van oliekwaliteit en oliedruk en hierop actie kunnen ondernemen (alarmen)
- Data verstuurd naar HMI web applicatie opslaan in database

## Should have

- Volledig functionerende webaplicatie waarbij overzichtelijk de gevisualiseerde data weergegevn wordt
- Met behulp van databus tussen motor en controller andere interessante gegevens uitlezen. Bijvoorbeeld:
  - Oliepijl in de motor, olie kwaliteit en andere parameters die interessant en beschikbaar zijn.
- software module voor het monitoren van de oliekwaliteit

## Could have

- Online beschikbaar maken van de uitgelezen gegevens

## Wont have

- software module voor:
  - uitlezen van de Emissie uitstoot van de motor, navigatie modulen voor het plannen van de vaarroute

**Figuur 5.2: Moscow Model**

## 6 Op te leveren producten

De volgende producten moeten aan het eind van het afstuderen opgeleverd worden:

- Brandstofmanagement module
- Motormanagement module
- Module voor het sturen van gegevens van modules naar het HMI systeem in de stuurhut waarop de informatie van de brandstofmanagement systeem en get motormanagement systeem weergegeven kan worden.
- Communicatie infrastructuur voor communicatie tussen het brandstofmanagement systeem, het motormanagement systeem en het HMI systeem
- Webapplicatie voor het ophalen en opslaan van data

De volgende producten mogen aan het eind van het afstuderen opgeleverd worden

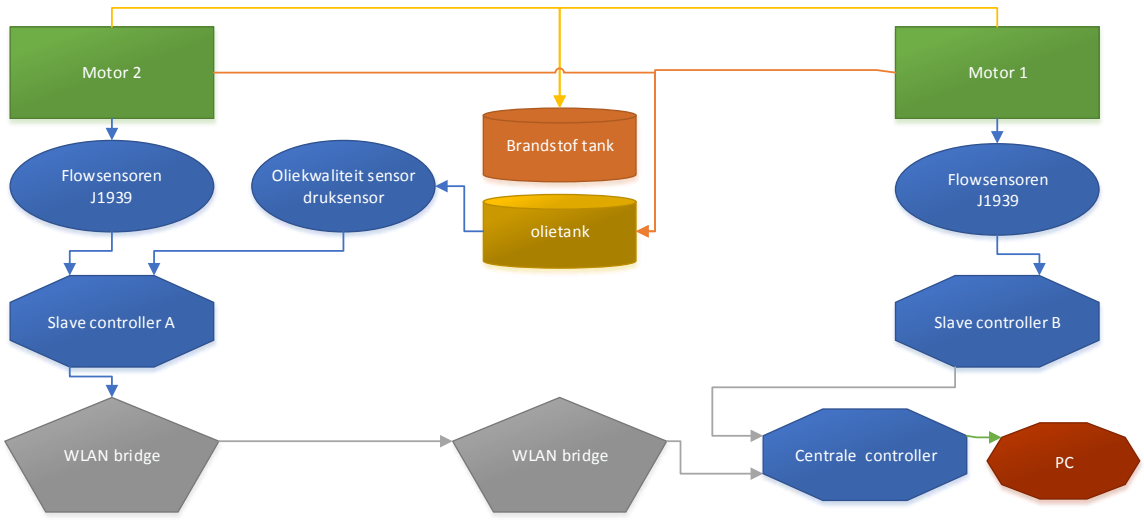
- Module om gegevens op de boot te communiceren naar de wal.
- Module om de oliekwaliteit te meten
- Volledig functionerende webapplicatie waarin overzichtelijk de data wordt weergegeven.

## 7 Opbouw hardware

Alhoewel alleen de software gemaakt hoeft te worden moet wel bepaald worden met welke controller er gewerkt moet worden en hoe die de signalen gaat meten. Daarnaast moet de manier waarop de controllers gaan communiceren uitgezocht worden. Enige hardware kennis is dus wel vereist.

De opbouw van de hardware met sensoren, controllers, een computer en de communicatie is verder uitgewerkt in Figuur 7.1.

De slave controllers verzamelen data van de motoren de de sensoren en sturen deze naar de centrale controller die via een com poort aangesloten is op een PC.




**Figuur 7.1: Hardware overzicht schema**

**Controllers**
Als controller zullen Syel controllers gebruikt worden. Syel controllers zijn opgebouwd rond een uController en zijn er in twee varianten compact control en industrial control. Compact control controllers hebben een lcd display, industrial controllers niet. Beide controller varianten zijn er in meerdere uitvoering waarbij het aantal databussen en io varieert. Elke controller die bij VMB gebruikt wordt is uitgerust met: zie Tabel 7.1

| Harware | Minimaal | Maximaal |
|---|---|---|
| DI, DO, | 4 | 16 |
| AI | 4 | 4 |
| AO | 2 | 2 |
| Encoders | 1 | 4 |
| RS232 | 1 | 2 |
| RS485 | 0 | 1 (op twee na) |
| CAN | 1 | 2 |
| ethernet | 1 | 1 |
| USB 2.0 | 1 | 1 |

**Tabel 7.1: IO en databussen op Syel controllers**

Syel controllers worden vaak toegepast bij VMB. De controllers zijn flexibel inzetbaar en worden in veel verschillende installaties van VMB gebruikt. Hierdoor zijn de meeste hardware engineers en software programmeurs bekend met de werking en beperkingen van de Syel controllers. De Syel controller beschikken op 2 versies na over de minimale hardware vereisten.

**Programmeer omgeving**
Als programmeer omgeving wordt Proteus gebruikt. Proteus is de standaard programmeer omgeving die gebruikt wordt voor het programmeren van syel controllers. Met behulp van deze programeer omgeving kan een syel controller programma geschreven worden in C of LD[1] (ladderdiagram). Als in de taal C geprogrammeerd wordt, zijn voor de meeste functionaliteit standaard functies aanwezig. Dit is niet het geval bij LD. Bijvoorbeeld voor het instellen van de verschillende databussen. Als van het S88.01. model gebruik gemaakt wordt zal m.b.v. deze standaard functie voor ieder stuk hardware een HCM ontwikkeld moeten worden. In de HCM kan functionaliteit geïmplementeerd worden die standaard functies niet bieden. De standaard functionaliteit m.b.t. de hardware in proteus

- Databussen: CAN, TCP/IP en serieel (RS232, RS485)
  - Instellen databus (snelheid, hardware poort)
  - Een bericht ontvangen
  - Een bericht verzenden
- IO: DI, DO, AI en AO
  - Ingangen lezen
  - Uitgangen schrijven en lezen

[1] Ladderdiagram is de simpelste programmeertaal om een controller/plc te programeren.

Voor het programma van het SMMS zal gebruik worden gemaakt van de taal C.

**Sensoren**
Zoals te zien is in Figuur 7.1 op pagina 7 worden voor dit project worden een aantal sensoren gebruikt.

- Een flowsensor voor het meten van het dieselverbruik
- Druksensor voor het meten van de oliedruk
- Oliekwaliteit sensor voor het meten van de olie kwaliteit, de olietemperatuur en de omgevingstemperatuur.
- Databus om motorgegevens uitgelezen

Voor het afstudeerproject is het alleen nuttig om te weten hoe de sensoren hun informatie aanleveren bij de syel controllers[1]:

- De flowsensor gebruikt een analoog 4-20ma signaal of een puls signaal om de gemeten stroomsnelheid door te geven aan de syel controller.
- De druksensor gebruikt een analoog 4-20mA signaal om de gemeten druk door te geven aan de syel controller.
- De oliekwaliteitssensor kan de gemeten olietemperatuur, kwaliteit doorgeven via een modbus verbinding, can verbinding en via twee analoge signalen. De gemeten omgevingstemperatuur kan alleen via de databussen doorgeven worden.
- De motorgegevens worden uitgelezen m.b.v. een canbus waarop berichten in een J1939 protocol worden verstuurd.

**Communicatie**
Verschillende syel controllers zullen draadloos met elkaar moeten communiceren. Aangezien de syel controllers zelf geen draadloze ethernet controller hebben zal dit gerealiseerd moeten worden met een WDS (wireless distriution system). Dit systeem bestaat uit een wireless bridge die allemaal draadloos met elkaar kunnen communiceren. Door de syel controllers aan te sluiten op een bridge wordt de syel controller onderdeel van het WDS. Als wireless bridge kunnen de WLAN modules van Weidmueller gebruikt worden. Deze modules kunnen als bridge werken met de extra functionaliteit dat de accespoint mode uitgeschakeld kan worden. Dit betekent dat het netwerk niet meer draadloos benaderd kan worden door clients van buitenaf. De installatie wordt daarnaast beveiligd met een WEP sleutel. Dit zorgt er voor dat WLAN modules die tot het netwerk behoren een WEP sleutel nodig hebben om te kunnen communiceren met andere WLAN modules. Voor een uitgebreider onderzoek naar de communicatie mogelijkheden voor  de interne communicatie zie hoofdstuk 14.5 Bijlage Wireless Communication onderzoek De centrale syel controller zal vervolgens via een com poort verzamelde data naar de webapplicatie sturen.

[1] Zie voor meer info zie 14.4 Bijlage vooranalyse

# 8 Opbouw software

In dit hoofdstuk wordt beschreven welk keuzes vooraf zijn gemaakt bij het ontwerpen van de software. Wat is de datastructuur, hoe worden verschillende modules aan elkaar gekoppeld. Met andere woorden: wat is de opbouw van de software?

De software bij VMB moet wordt opgebouwd volgens de standaard methode van VMB. Deze methode is terug te vinden in de VSB (VMB standaardisatie boek) en is gebaseerd op S88.01. S88.01 is een internationale standaard voor het beschrijven productie processen, maar kan ook gebruikt worden voor continue processen, zoals het verkrijgen en opslaan van informatie. S88.01 wordt gebruikt voor het modulair indelen van software voor processen[1]. De rede dat VMB deze standaardisatie gebruikt:

- Vereenvoudigen van overdragen software aan collega.
- Mogelijkheid om met meerdere programmeurs aan één project te werken.
- Eenduidige structuur waardoor het, oplossen van storingen wordt eenvoudiger.

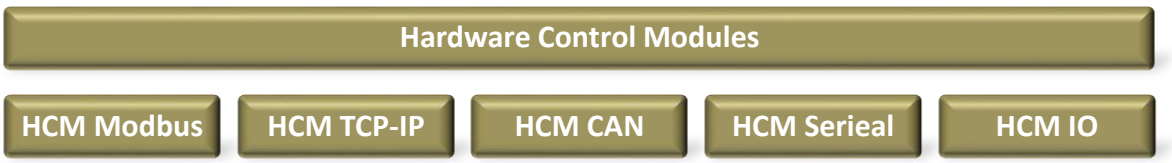
De modulaire indeling van S88.01 die van toepassing is op het SMMS project is weergegeven in Figuur 8.1. Door gebruik te maken van de standaardmethode binnen VMB is de software makkelijk te begrijpen voor de programmeurs binnen VMB de modulaire opzet maakt de software overzichtelijk. Door de modules zelfstandig te laten functioneren hebben modules geen complexe besturing nodig van de hun bovenliggende modules, behalve een simpel start of stopsignaal. Dit zorgt er voor herbruikbaar is. De losse modules kunnen daardoor ook voor andere doeleinden gebruikt worden.

**Figuur 8.1: indeling S88.01.**

## 8.1 S88.01.

**Hardware Control Module**

De Hardware Control Module (HCM) bevinden zich op het laagste niveau in het software model. De HCM sturen de verschillende hardware onderdelen van de controller aan. Er zijn HCM voor de encoder, RS232, RS485, Ethernet, CAN, en IO hardware. Bij HCM voor de communicatie via RS232, RS485, tcp/ip en Can wordt via variabelen, opgeslagen in de HCM, aan andere modules duidelijk gemaakt of zij klaar zijn om berichten te verzenden en of er nieuwe berichten zijn binnengekomen. Berichten die worden verzonden worden in een verzend buffer geplaatst. Berichten die ontvangen zijn worden in een ontvangst buffer geplaatst. De HCM voor de IO lezen de ingangen uit en plaatsen deze in de een array. De uitgangen worden aangestuurd aan de hand van waarden geplaatst in een andere array. De waarden van de ingangen en uitgangen worden iedere programma cycli bijgewerkt.




[1] (VSB (VMB standaardisatie Boek), 2016) zie 14.8 Bijlage VSB hoofdstuk 5 Software

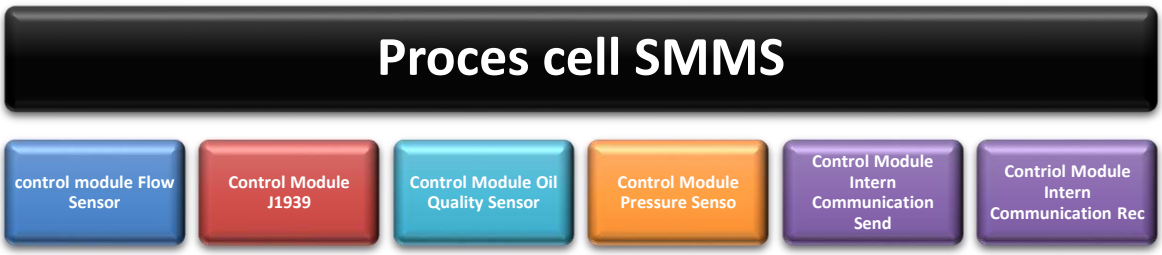
Figuur 8.2: Software overzicht SMMS HCM's

**Control Module**

Deze modules zitten op het een na laagste niveau in het software model. Zij worden gebruikt voor het realiseren van basic control:

- Uitlezen van een sensor
- Aansturen van een klep e.d.

Voor een overzicht van de Control Modules zie Figuur 8.3. De Control modules (CM) werken onafhankelijk van de HCM. Dit wil zeggen dat een CM niet direct bestuurd wordt door een HCM. De functies waarmee de CM, van bijvoorbeeld een flowsensor communiceert met de flowsensor, worden iedere programma cycli aangeroepen door het hoofdprogramma van de PLC.



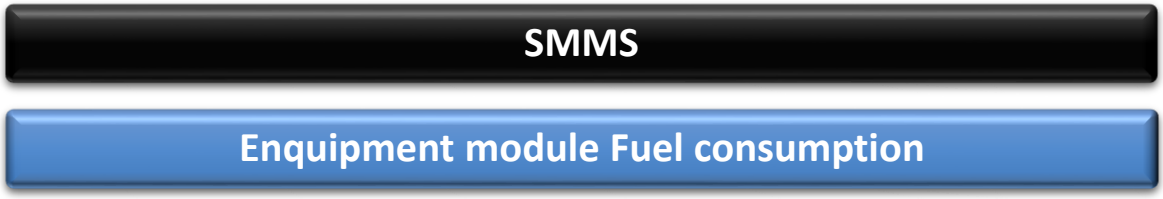

Figuur 8.3: Software overzicht SMMS.a

**Equipment module**

De equipment module bestaat uit 1 of meer control modules. Equipment modules zijn verantwoordelijk voor het controleren van een proces

- Bijhouden brandstofverbruik van 1 motor e.d. (zie Figuur 8.4)

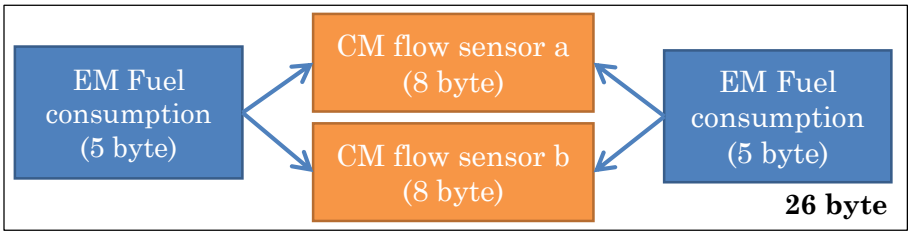
Er is voor gekozen om bij het SMMS project de equipment modules zelfstandig te laten functioneren. (zie kopje Control module).




Figuur 8.4: Software overzicht SMMS.b

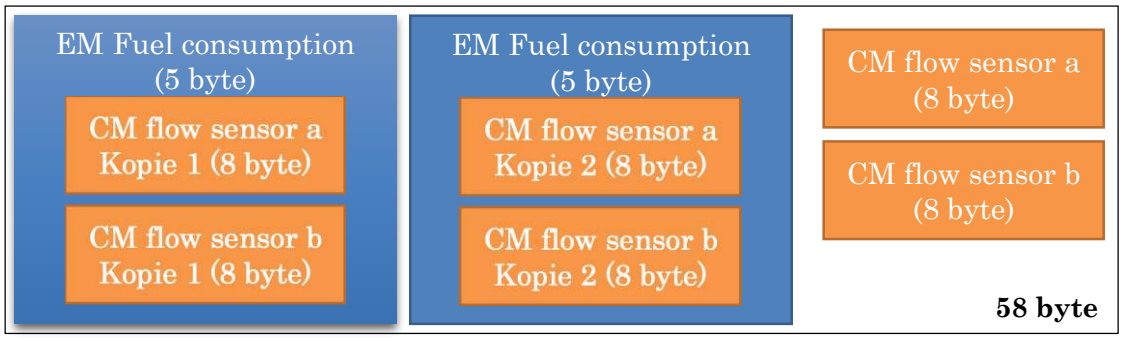
## 8.2 Implementatie S88.01.

Door alle settings en data voor een module, bijvoorbeeld een flowsensor control module, in een structure te plaatsen, kan door het definiëren van meerde objecten van dezelfde structure data voor meerdere flowsensoren opgeslagen worden. Modules worden aan elkaar gekoppeld d.m.v. pointers (zie Figuur 8.5) i.p.v. de CM structure in de EM structure te definiëren (zie Figuur 8.6) hierdoor is het mogelijk om de het zelfde HCM of CM object te gebruiken voor meerdere CM's of EM's. Dit scheelt veel geheugen gebruik en de waardes van de originele CM hoeven niet elke programma cycli gekopieerd te worden naar de "kopie CM's" in de EM´s. Door gebruik te maken van pointer hebben de EM's die gebruik maken van een CM altijd de actuele waarden van de CM's.




**Figuur 8.5: Data structuur met pointers**

Een HCM, CM of EM wordt aangemaakt door een object te definiëren van de struct van de desbetreffende module. Met behulp van een init functie, gemaakt voor de module kan de module ingesteld worden. De waardes van de instellingen worden meegeven als functieargument van de init functie. In alle functie gedefinieerd voor een module wordt altijd het adres van het object van de betreffende module meegegeven[1]. Hierdoor is het mogelijk om de zelfde init functie te gebruiken voor twee verschillende CM Flow Sensor modules. Het functieargument met het adres geeft aan welke CM Flow Sensor ingesteld moet worden. Zogenaamde hulpfuncties hebben meestal geen functie argument met het adres van de module waarop de functie uitgevoerd worden. Dit omdat die geen waardes van de module veranderen. Hulpfunctie worden vooral gebruikt om bijvoorbeeld een waarde te berekenen, of om bijvoorbeeld twee chars (8 bit lange variabelen) samen te voegen tot een 16 bit lange variabele.

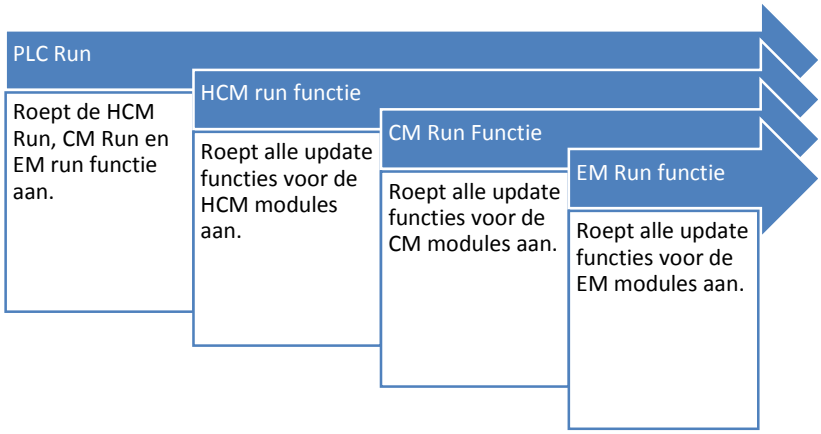



**Figuur 8.6: Data structuur zonder pointers**

De functie waarmee HCM´s , CM´s en EM´s hun data updaten of uitsturen (aansturen van een klep, temperatuurwaarde van de temperatuurmeter uitlezen) worden aangeroepen in het hoofdprogramma van de PLC. M.a.w. een EM Fuel Consumption hoeft geen opdracht te geven aan de CM Flow Sensor om de flowsensor uit te lezen, dit gebeurd

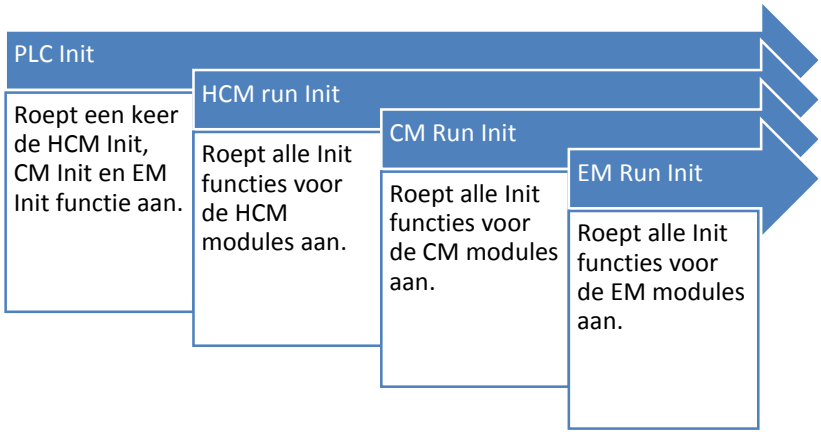
[1] dit functie argument zal dan ook niet meer benoemd worden bij de uitleg van de functies van de HCM CM en EM. Wel zullen eventuele extra functieargumenten benoemd worden.

iedere programma cycli in het hoofdprogramma. Hiermee wordt voorkomen dat een CM (of HCM) die door meerder EM's (of CM's) gebruikt wordt twee keer een sensor gaat uit lezen.

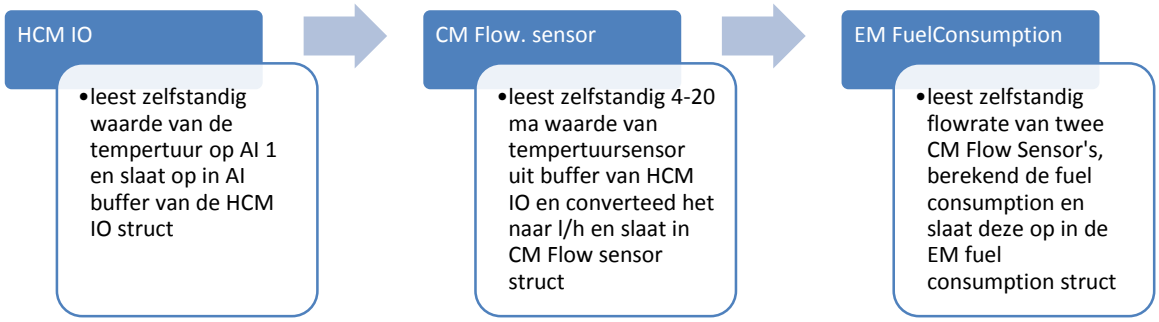



**Figuur 8.7: PLC Run Program flow**

De waardes van de uitgelezen sensor die opgeslagen worden in de struct van de CM , of de waarde waarmee de klep aangestuurd wordt (open of dicht), kunnen door iedere EM (of CM) die verbonden is aan de CM (of HCM) uitgelezen of beschreven worden. De communicatie tussen de verschillende modules gebeurd d.m.v. de variabelen in de struct niet d.m.v. functies. Uitzondering hierop is als een waarde in een buffer (array of 2d array) geplaatst moet worden. Een EM die een waarde in een buffer in de struct van een CM wil plaatsen, roept hiervoor een functie gemaakt voor de desbetreffende CM aan om de waarde in de buffer te plaatsen. Dit om te voorkomen dat een buffer verkeerd beschreven wordt. In het geval van bijvoorbeeld een modbus buffer is de betekenis van de waarde plaatsgebonden. De character op plaats 1 in de buffer bepaald hoe de ontvanger van het bericht zal reageren. Als deze waarde op de verkeerde plaats in de buffer wordt ingevuld krijg je onvoorspelbaar gedrag. Door een functie te gebruiken wordt voorkomen dat er een fout gemaakt wordt bij het opstellen van het modbus bericht. De functie standaardiseert de procedure voor het opstellen van het modbus bericht. Bovendien kan de functie feedback geven (return waarde). Aan de hand van de feedback kan bepaald worden of het plaatsen van het modbus bericht gelukt is. Het uitlezen van een buffer gebeurd wel direct (zonder functie).




**Figuur 8.8: PLC Init Program flow**




**Figuur 8.9: Informatie flow van sensor naar EM module**

Door de data structuren en de functies of deze manier op te stellen is de software makkelijk uit te breiden. Voor een tweede CM Flow Sensor hoeft alleen een tweede struct object van de CM Flow Sensor aangemaakt te worden en een init en een update functie van CM Flow Sensor aangeroepen te worden in PLC init en PLC run. Hiermee is voldaan aan alle eisen voor de software beschreven in hoofdstuk 5.2 Eisen aan de software op bladzijde 5.

## 8.3 Controle cycli

Er zijn twee cycli die van toepassing zijn op de controllers van het SMMS project. De Init cycli en de Run cycli. In de Init cycli worden de CM, en EM objecten geïnitialiseerd. In de run cycli worden alle objecten gerund. In deze cycli worden bij de de run functies aangeroepen gedefinieerd in het hoofdprogramma van de HCM, CM en EM. Die roepen weer alle update functies van de HCM, Cm of EM aan.

# 9 Software Units

De software is opgedeeld in verschillende Units. Per unit zullen de bijbehorende EM's en CM's besproken worden. Als eerste zal de algemene datastructuur en aansturing van de software componenten besproken worden. Voor alle besproken functie onder de kopjes functie zijn flowcharts gemaakt. De flowcharts zijn te vinden in 14.3 Bijlage Flowcharts HCM, CM en EM. in deze bijlage is een hoofdstuk indeling gemaakt die overeenkomt met de hoofdstuk indeling van hoofdstuk 9 Software Units. De test rapporten van de desbetreffende HCM, CM of EM in 14.2 Bijlage Test rapporten HCM, CM en EM.

In alle functie gedefinieerd voor een module wordt altijd het adres van het object van de betreffende module meegegeven, dit functie argument zal dan ook niet meer benoemd worden bij de uitleg van de functies van de HCM CM en EM. Wel zullen eventuele extra functieargumenten benoemd worden.

## 9.1 HCM

in dit blok staat de functies waarmee alle initialiseer en update functie van de HCM's aangeroepen worden. De functie worden voor ieder object een keer aangeroepen per programma cycli. Dus als er twee seriële HCM struct objecten zijn gedefinieerd (A en B) wordt twee keer de InitSer functie aangeroepen 1 keer met adres A en een keer met adres B.

### 9.1.1 HCM IO

Deze HCM wordt gebruikt voor het aansturen van DO en AO en het uitlezen van de DI en AI. In tegenstelling tot alle andere HCM waar er voor iedere com poort, tcp/ip socket en Canbus aansluiting een aparte HCM wordt aangemaakt. Is voor de IO van de syel controller één HCM IO die alle IO aantuurt. Dit is gedaan om het aansturen van de IO overzichtelijk te houden en op deze manier kunnen de waarden van de IO opgeslagen worden in array's waardoor alle IO achter elkaar is in te lezen en te beschrijven zijn. De HCM IO ondersteund de mogelijkheid om Remote IO van syel te gebruiken. De IO op de remote IO worden gewoon doorgeteld. M.a.w. als een syel controller 4 ingangen heeft en het remote IO station 8. Dan zit op de syel controller DI 1 t/m 4 en op de remote IO 5 t/m 12. In de software wordt geen onderscheid gemaakt tussen de IO op de controller en de IO op de remote IO. De HCM IO ondersteund de mogelijkheid om de DI van de syel controller te gebruiken als puls ingangen.[1]

**Struct**

struct hcmIOData: bevat alle gegevens van een HCM IO

- bool bDI[]: array met waarde van de DI. lengte afhankelijk van het aantal DI
- bool bDO[]:array met waarde van de DO. lengte afhankelijk van het aantal DO
- unsigned short int usiAI[]:array met waarde van de AI. lengte afhankelijk van het aantal AI
- unsigned short int usiAO[]:array met waarde van de AO. lengte afhankelijk van het aantal AO
- unsigned char NrOfDI: aantal DI op de controller
- unsigned char NrOfDO: aantal DO op de controller
- unsigned char NrOfAI: aantal AI op de controller
- unsigned char NrOfAO: aantal AO op de controller
- unsigned char Device: geeft aan welke syel controller er gebruikt wordt.
- unsigned char ucModuleNr: het ID nummer van de HCM IO
- unsigned short int usiCounterRisingEdge[]: hulpvariabelen voor het berekenen van de usiCounterRisingEdgePerSec[].
- unsigned short int usiCounterRisingEdgePerSec[]: array gevuld met variabelen die het aantal rising edges per seconde op de DI bijhouden. legnte afhankelijk van het aantal DI op de syle controller.
- bool bMark[]: hulp variabele om te zorgen dat bij iedere DI in de interupt service routine gecontroleerd wordt of er een rising edge op de DI aanwezig was.
- bool bChangePerSec: hulpvariabele voor bChangePerSecInd.
- unsigned char ucChangePerSecOld: hulpvariabele voor bChangePerSecInd.
- bool bChangePerSecInd: geeft aan dat er een nieuwe risingedge/sec is berkend. Aan de hand van deze variabele kunnen andere CM en EM die aan de HCM IO gekoppeld dit zien.

**Figuur 9.1: struct hcmIOData**

[1] Op dit moment alleen nog voor het controller type V4L

**Interrupt**

De syel controllers kunnen maximaal 500 keer per seconde de ingangen uitlezen[1] als de ingangen uitgelezen worden in het hoofdprogramma. Dit is te weinig aangezien de meeste flowsensoren die de puls ingang gaan gebruiken meer dan 500 pulzen per seconde afgeven. Daarom moeten de puls ingang op interrupt basis werken. Hierdoor kan er tijdens het lopen van het programma het aantal pulsen op de DI geteld worden, waardoor er veel meer pulsen geteld kunnen worden.

Meestal gebeurd het uitlezen van de IO door de uController in de syel controller d.m.v. een databus ingebouwd in de uController. Om de rising edge op een DI op interrupt basis uit te lezen zal gebruik gemaakt moeten worden van een speciaal circuit in de uController die de DI en DO "direct" uitleest en aanstuurt. Voor het instellen en detecteren van een rising edge interrupt op de DI worden de volgende registers gebruikt:

- SCS: met het 1e bit van dit 32 bit brede register word bepaald of de uController de IO uitleest door de databus (0) of direct uitleest via een speciaal circuit (1)
- EXTINT: Het 4e bit van dit register genaamd EINT3 wordt 1 als er aan de eisen van een interrupt voldaan is, in ons geval een rising edge op een van de DI. Als EINT3 1 is wordt de interrupt service routine (ISR) opgestart en wordt de functie die aan deze ISR is gekoppeld uitgevoerd.
- EXTMODE: het 4e bit in dit register bepaald waar de EINT3 op reageert. De twee mogelijkheden zijn niveau en edge. Bij niveau wordt er een interrupt gegenereerd als een DI 1 of 0 is bij edge wordt er een interrupt gegenereerd als de DI 1 of 0 wordt. Dit register wordt zo ingesteld dat de interrupt gegenereerd wordt bij edge.
- IO0_INT_EN_R: dit register bepaald bij welke DI EINT3 reageert. Het is een 32 bit breed register in het geval van de V4L corresponderen bit 8, 9, 14 en 15 met DI een, twee, drie, en vier. Door deze bitjes op één te zetten genereren rising edge of de waarde (afhankelijk van EXTMODE) van DI een t/m vier een interrupt op EINT3. Bit 8, 9, 14 en 15 van dit register worden op één gezet.
- IO0_INT_STAT_R: dit register geeft aan of er een interrupt heeft plaatsgevonden door een rising endge op DI een t/m vier. Als alle register bij een rising edge een interrupt genereren. Dan wordt bij een rising edge op DI één bit acht van IO0_INT_STAT_R 1. Aangezien bij elke DI dezelfde interrupt genereerd wordt kan met dit register bepaald worden welke DI de interrupt veroorzaakte. De DI corresponderen met de zelfde bitjes als bij IO0_INT_EN_R. Als tijdens het uitvoeren van de ISR van de interrupt van DI één ook bij DI twee een rising edge binnenkomt genereert dit niet opnieuw een interrupt en zou de puls niet meegeteld worden. Door tijdens de ISR altijd te controleren op bit 8, 9, 14 of 15 hoog is worden er geen pulsen gemist.
- IO0_INT_CLR : door een één te schrijven naar een bit in dit register kan een bit in register IO0_INT_STAT_R gereset worden. Pas als een bit in IO0_INT_STAT_R gereset is kan er opnieuw een interrupt gegenereerd worden door de ingang waarmee het bit in IO0_INT_STAT_R correspondeert.

[1] Zie 14.2 Bijlage Test rapporten HCM, CM en EM hoofdstuk testraapport HCM IO

**Functies**

Voor de werking en initialisatie van de HCM IO zijn een aantal functies gedefinieerd. Met behulp van de _InitIO functie wordt de HCM IO geïnitialiseerd. Als extra functie argument wordt een bool variabele meegegeven die aangeeft of er gebruik gemaakt wordt van remote IO. Tijdens de _InitIO functie worden alle uitgangen gereset daarnaast wordt de init_gpio_EXT_IRQ functie aangeroepen. Deze functie is verantwoordelijk voor het instellen van de SCS, EXTINT, EXTMODE en IO0_INT_EN_R. Daarnaast start init_gpio_EXT_IRQ een aparte thread met als functie _MeassureCounterPerSec.

Deze functie heeft als taak om per seconde het aantal pulsen van alle DI te tellen. Hierdoor weet je hoeveel pulsen er per seconde zijn binnengekomen. De rede dat dit in een aparte thread wordt gedaan is dat het meten van de tijd (100ms) in het hoofdprogramma niet precies genoeg is. Afhankelijk van de duur van het hoofdprogramma varieerde de tijd algauw tussen de 80 en 120 ms. De mogelijkheid om een thread precies 1 keer per 100ms uit te voeren gaf preciezere resultaten. De tijd varieerde bij het gebruik van de thread tussen de 99ms en de 101 ms.

De _SignalNewVal functie is gemaakt om aan andere HCM, CM of EM aan te geven dat er een nieuwe waarde (pulsen/seconde) gemeten is. Op bediening van de _MeassureCounterPerSec functie wordt een variabele genaamd bChangePerSecInd een programma cycli hoog gemaakt.

Het uitlezen en aansturen van de IO is gelukkig een stuk simpeler. De waarde van de DI, DO, AI en AO zijn in vier apparte arrays opgeslagen. Twee arrays met booleans voor de DI en de DO en twee met unsigned short int voor de AI en de AO. Met de functie _ReadAndWriteIO worden alle AI en DI ingelezen en de waardes in de array geplaatst en worden alle waardes voor de AO en DO uit de array gelezen en doorgezet naar de AO en de DO. Andere CM en EM kunnen dan de waarde van de DI en AI uitlezen uit de arrays en als ze de AO of DO willen aansturen de waardes plaatsen in de arrays.

De _UpdateIO functie wordt aangeroepen in het HCM hoofdprogramma. In de _UpdateIO functie worden de _ReadAndWriteIO en de _SignalNewVal functie aangeroepen.

### 9.1.2 HCM serieel

HCM serieel object heeft als taak een seriële port aan te sturen (bijvoorbeeld seriële RS232 port A). Het HCM serieel object zal ontvangen berichten in een buffer te plaatsen en als de zendbuffer gevuld is een bericht gevuld met de data uit de zendbuffer verzenden. De grote van de berichten die verzonden en ontvangen kunnen worden, zijn beperkt door de grote van de zend en ontvangstbuffer. De datastructuur van het HCM serieel struct is te zien in Figuur 9.1.

| struct hcmDataSetSer: bevat alle gegevens van een HCM Serieel. |
|---|

- COM *pPort: Pointer naar COM port hardware
- long int liBaud: Baudrate instelling van de comport
- unsigned char ucTXBuffer[256]: Zendbuffer
- unsigned char ucRXBuffer[256]: Ontvangstbuffer
- unsigned char ucTXBufferNr: houd bij hoeveel berichten er moeten worden verzonden
- unsigned char ucRXBufferNr: houd bij hoeveel berichten er zijn ontvangen
- bool RXNewMessage: geeft aan dat er een nieuw bericht is ontvangen

**Tabel 9.1: struct hcmDataSetSer**

**Functionaliteit**

Voor iedere seriële com port op de syel controller kan een object van het HCM serieel struct gedefinieerd worden. De pPort pointer verbind de com port hardware van de Syel controller met het object van de HCM serieel struct. De HCM ondersteun de mogelijkheid om berichten van maximaal 16383 byte te maken, verzenden en ontvangen d.m.v. een com poort. De buffer lengte kan voor geheugen efficiëntie makkelijk verkleind worden. standaard is de maximaal berichtlengte ingesteld op 256 byte. De HCM Serieel zal d.m.v. een variabele aangeven of er een bericht is ontvangen.

**Functies**

M.b.v. de InitSer functie worden alle waarden van het HCM serieel object geïnitialiseerd. De run functie word iedere programma cycli aangeroepen en verzorgt het ontvangen en verzenden van berichten. Hierbij wordt de com poort aan de HCM serieel gekoppeld en wordt de zendsnelheid ingesteld.

Als een object waaraan de HCM serieel object gekoppeld is een bericht wil verzenden plaatst dit ander object een bericht in de verzend buffer van het HCM serieel object. Dit kan met functie _FillTxBufferSer. Het andere object kan een bericht van maximaal 256 karakters in de zend buffer plaatsen.

Met de _SendBufferSer functie wordt een bericht geplaatst in de buffer verzonden. Met de _ReceiveSer functie wordt gecontroleerd of er een bericht is ontvangen en worden deze geplaatst in de ontvangstbuffer. De _UpdateSer functie roept de _ReceiveSer en de _SendBufferSer functie aan.

Daarnaast is er nog een hulpfunctie _SetParStop kunnen de stopbit en pariteit instellingen van de com poort gekoppeld aan de HCM Serieel veranderd worden.

De _SendCharSer is een functie die speciaal gemaakt is om berichten groter dan 256 byte te verzenden. Het maximaal aantal byte wat hiermee verzonden kan worden is 16383 byte.

### 9.1.3 HCM RSModbus

HCM RSModbus wordt gebruikt om data te verzenden over een seriële databus ingepakt in een modbus protocol.

**Modbus**

Modbus is een protocol dat gebruikt wordt om op afstand appraten te besturen en gegevens uit te lezen door het schrijven en lezen van registers. Je hebt input register die alleen maar uitgelezen mogen worden met de modbus en holding register die beschreven en uitgelezen mogen worden via modbus. Een modbus netwerk bestaat uit een master met een of meerdere slaves ieder met een eigen uniek adres. Modbus is een zogenaamd request en receive protocol. Dit betekent dat een slave niks doet totdat een master een request naar een slave toestuurt. Het modbus protocol heeft twee versies namelijk de modbus RTU en de modbus ANCII. Beide versies hebben geen verschil in functionaliteit maar verschillen wel in de manier waarop zij de data in de modbus berichten uitdrukken en de manier waarop zij de berichten controleren.[1] Met modbus kunnen enkele of meerdere registers beschreven worden. De functiecode die met het bericht wordt meegestuurd bepaald wat de ontvanger van het bericht doet. De in de HCM RSModbus ondersteunde functie codes:

- 3: opvragen inhoud meerdere holding register
- 4: opvragen inhoud meerdere input registers
- 6: beschrijven inhoud 1 holding register.

Als voorbeeld wordt de berichten van de communicatie tussen een modbus master en een modbus slave gegeven, er wordt gebruik gemaakt van het modbus RTU protocol. De master stuurt een request naar de slave met adres 17. In dit request vraag de modbus om de waarden in register 30006 t/m 30008 ofwel input register 5 , 6 en 7.

Het bericht verstuurd naar de slave in hexadecimale waarden: 11 04 00 05 03 01 23 AB

- 11 : adres van de slave (17)
- 04: functie code
- 00 05: adres van register 1e opgevraagde register (5 + ofset:30001 = 30006). Deze waarde wordt voor het modbus bericht opgesplitst in twee waarden. AdresHI (00) en AdresLO (05)
- 0003: aantal registers dat opgevraagd wordt (3). Deze waarde wordt opgesplitst voor het modbus bericht in twee waarden. NrRegHI en NrRegLO.
- 23AB: CRC (cyclic redundancy check) de crc wordt opgesteld op basis de inhoud van het bericht. De ontvanger zal zelf opnieuw een crc code genereren op basis van de inhoud van het bericht. Deze moeten het zelfde zijn. Op deze manier kan gecontroleerd worden of het bericht zonder fouten is overgekomen.[2] Ook deze waarde wordt opgesplitst in twee waarden

[1] Voor meer informatie over modbus zie http://www.simplymodbus.ca/ASCII.htm
[2] Voor meer informatie over CRC zie https://nl.wikipedia.org/wiki/Cyclic_redundancy_check

Het bericht de slave terugstuurt naar de master: 11 04 06 00 0A 00 44 00 42 F8 F4

- 11: adres van de slave
- 04: functiecode
- 06: bytes die nog volgen (8 – crc bytes = 6)
- 00 0A: waarde van register 30006 ofwel inputregister 5. Deze waardes komt binnen als twee losse char variabelen (8bit) en moet samengevoegd worden tot een int (16 bit)
- 00 44: waarde van register 30007 ofwel inputregister 6.
- 00 42: waarde van register 30008 ofwel inputregister 7
- F8 F4: crc

**Struct**

| struct hcmDataSetRSMod: bevat alle gegevens van een HCM RSModbus |
|---|
| • hcmDataSetSer *pTXRXPort: pointer naar adres van een HCM serieel |
| • ucParityStopMode: bedoel om de setting van de parity en stop bits van de serieele poort op te slaan |
| • ucTXMessage[20]: één dimensionale buffer voor het opslaan van één modbus bericht |
| • ucTXMessageLength: variabelen die aangeeft hoe lang de te verzenden message is in chars |
| • bExpectReqMessage: bit dat geset wordt als er een reply van een slave wordt verwacht |
| • bExpectReqNoMessage: bit dat wordt geset als er geen reply van een slave wordt verwacht |
| • bExpectRecMessageError: bit dat geset wordt als er geen reply komt van een slave terwijl dit wel zou moeten |
| • bPremToSend: bit dat intern aan de HCM RSModbus aangeeft of er een nieuw modbus bericht aangemaakt mag worden (pas als het oude modbusbericht verzonden is) |
| • bNewTXMessage: bit dat geset wordt als er een nieuw modbus bericht is aangemaakt |
| • bNewRXMessage: bit dat geset wordt als er een modbus bericht is binnengekomen |
| • ulRXDelayState: hulp variabele voor het bRXDelay |
| • bRXDelay: variabele waar iedere 100ms een puls op komt te staan als er geen reply van de slave ontvangen wordt. |
| • ucRXdelayCounter: variabele die opgehoogd wordt als er een puls op bRXDelay staat. Als ucRXdelayCounter groter of gelijk is aan 5 (500ms) wordt bExpectRecMessageError geset |
| • ulTXDelayState: hulp variabele voor het bTXDelay |
| • bTXDelay; variabele waar iedere 10ms een puls op komt te staan als de functie om een modbus bericht te verzenden wordt aangeroepen; |
| • ucTXdelayCounter: variabele die opgehoogd wordt als er een puls op bTXDelay staat. als ucTXdelayCounter groter of gelijk is aan 3 (30ms) dan mag er een nieuw modbus bericht verzonden worden. |
| • ucRxBufferLength: variabele die aangeeft hoever de usiRXRegAddress usiRxRegValue |
| • ucTXDevAddress: variabele die het adres van de modbus slave bewaard van het laatst verzonden modbusbericht |
| • ucTXFunctionCode: variabele die de functie code in het laatst verzonden modbus bericht bewaard |
| • usiRXRegAddress[]: array waarin de adressen opgevraagde of beschreven registers van het laatst verzonden modbusbericht opgeslagen worden. |
| • usiRxRegValue[]: array waarin de waarden van de register opgeslagen in usiRXRegAddress geplaatst worden. deze waarden komen binnen via het reply bericht van een modbusslave |

**Tabel 9.2: struct hcmDataSerMod**

**Functionaliteit**
De HCM RSModbus ondersteund de mogelijkheid om als modbus master te functioneren. De HCM kan modbus berichten van maximaal 16383 byte lang genereren, versturen en ontvangen. De bericht lengte kan voor geheugen efficiëntie makkelijk aangepast worden. Standaard staat de maximale bericht lengte ingesteld op 256 byte. Er kan per programma cycli maar één modbus bericht verstuurd en ontvangen worden. Voor alle verzonden modbus berichten krijgt de master een response van de slave met daarin de opgevraagde data van of beschreven data in de registers. Daarom wordt ieder modbus bericht na controle uitgepakt en de waardes van de register opgeslagen in een regValarray, parallel daaraan loopt een regAdres array waarin bij het versturen van de request al de adressen van de opgevraagde of beschreven registers zijn geladen. Andere CM of EM verbonden aan HCM RSModbus kunnen in deze array zien welke waarden de registers hebben van een slave. Het slave adres en functie code kunnen de CM en EM vinden in de eerste twee charters van het ontvangen modbus bericht. Met een variabele bNewRXMessage kan de HCM RSModbus aangeven of er een nieuw modbus bericht is aan de CM en EM die verbonden zijn met HCM RSModbus.

**Functies**
Voor het initialiseren van HCM RSModbus en het aanmaken, zenden en ontvangen van modbus berichten maakt de HCM RSModbus gebruik van vier functie en heeft daarnaast nog een aantal hulp functies. Als functieargumenten van de HCM RSMModbus worden als extra functieargumenten een adres naar een HCM Serieel opgegeven en een variabele die aangeeft wat de stopbit en partiteits instellingen zijn voor de HCM Serieel. De HCM Serieel wordt gebruikt om de modbus berichten te versturen over een seriële databus (RS232 of RS485).

Andere CM en EM die verbonden zijn aan de HCM RSModbus kunnen een modbus bericht aanmaken door de _CreateMessageMod functie aan te roepen. Deze functie heeft de volgende extra functie argumenten:

- unsigned char SlaveAddress: adres van de slave waarnaar het bericht verzonden moet worden
- unsigned char RFunction: functie code kan 3, 4, of 6 zijn andere functie codes worden niet ondersteund.
- unsigned char StartAdressHi & unsigned char StartAdressLo: bevatten het adres 1e register en in het geval van functiecode 6 het enige register waarvan gegevens opgevraagd worden of in geval van functiecode 6 beschreven worden.
- unsigned char NumberOfRegHi & unsigned char NumberOfRegLo: geeft in geval van functie code 3 en 4 aan van hoeveel registers na 1e register nog meer waardes opgevraagd worden. In het geval van functie code 6 bevatten deze functieargumenten de waarde die naar het register moeten worden geschreven.

De _CreateMessageMod kan een modbusbericht aangemaakt worden en opgeslagen worden in een zendbuffer. De _CreateMessageMod controleert of die een bericht mag aanmaken, als dit niet kan, omdat bijvoorbeeld de buffer al gevuld is met een modbus bericht, stuurt _CreateMessageMod een returnwaarde nul. De CreateMessageMod roept tijdens opstellen van het bericht de hulpfunctie functie _CreateCrcMod aan.

_CreateCrcMod berekend en stuurt als returnwaarde de CRC voor het modbus bericht. Als laatste vult de _CreateMessageMod de usiRXRegAddress array met de adressen waarvan de waarde opgevraagd of beschreven moeten worden. Ook vult _CreateMessageMod de variabelen ucTXDevAddress en ucTXFunctionCode met respectievelijk het adres van de modbus slave en de functiecode in het verzonden bericht.

Hierdoor kunnen andere CM en EM de functiecode en het slave adres bepalen zonder kennis te hebben van de opbouw van een modbus bericht. Als het bericht is opgesteld en in de buffer is geplaatst worden de variabele bExpectReqMessage en bNewTXMessage op één gezet, bPremToSend op nul en stuurt _CreateMessageMod een returnwaarde één. Hiermee wordt respectievelijk aangegeven dat een antwoord verwacht wordt van de slave, dat er een nieuwe modbus bericht is aangemaakt om te verzenden, dat er niet opnieuw een modbus bericht aangemaakt mag worden tot die is verzonden en dat het gelukt is een nieuw modbus bericht aan te maken.

Met de _SendMessageMod kan een modbus bericht verzonden worden. Aan de hand van de bNewTXMessage controleert de _SendMessageMod of er een nieuw modbus bericht in de zend buffer zit. Als dit zo is dan wordt er 50ms seconde gewacht en dan wordt de _FillTxBufferSer van de HCM Serieel aangeroepen. De HCM Serieel handelt dan het verzenden van het bericht af.

De _ReadMessage is een functie die een binnenkomend modbus bericht controleert en daarna de waardes van de registers die beschreven of uitgelezen zijn, plaatst in de usiRxRegValue array. Met de _CreateCrcMod functie wordt de crc van het ontvangen modbus bericht berekend en vergeleken met de crc in het ontvangen modbus bericht. Als dit het zelfde zijn wordt gecontroleerd of het slave adres en de functie code in het ontvangen modbus bericht van de slave gelijk is aan die in het verzonden modbus bericht. Als dit het geval is kan de HCM RSModbus er van uit gaan dat die een compleet en kloppend modbus bericht heeft ontvangen. Daarna zal de _ReadMessage functie de waardes van de opgevraagde of beschreven array in de usiRxRegValue array plaatsen. Als geen bericht is ontvangen terwijl er wel een modbus bericht verzonden is naar een modbus slave dan zal er na 500ms een bExpectRecMessageError op één gezet worden.

Als laatst is er nog de _UpdateMod functie deze functie word aangeroepen in het hoofdprogramma van de HCM. De _UpdateMod roept de _SendMessageMod en _ReadMessage aan.

### 9.1.4 HCM TCP/IP

De HCM TCP/IP wordt gebruikt voor het aansturen van de ethernet controller, om op die manier d.m.v. tcp/ip te kunnen communiceren met andere Syel controllers. De HCM TCP/IP ondersteund de mogelijkheid om berichten tot 200 byte breed te verzenden.

struct hcmHardwareDataSetTcpIp: bevat alle gegevens van een hcmHardwareDataSetTcpIp

- *pTcpHost: pointer naar een HOST variabele. dit is een speciale variabele waarin de instellingen van een socket opgeslagen worden.
- *cpTcpIp: in deze variabele wordt het ip adres waarmee de socket verbinding moet maken opgeslagen.
- bMode: bepaald of de variabele in client of server mode werkt
- usiSendPort: poort nummer waarop TCP/IP berichten verzonden worden
- usiRecPort: poort nummer waarop TCP/IP berichten ontvangen worden
- ucSendMessages: hulpvariabele voor ucSendMessagesFreq
- ucRecMessages: hulpvariabele voor ucRecMessagesFreq
- ucSendMessagesFreq: variabele voor het meten van het aantal verzonden berichten per seconde
- ucRecMessagesFreq: variabele voor het meten van het aantal ontvangen berichten per seconde
- ucMessageLength: variabele die aangeeft hoe lang een message maximaal kan zijn
- ucBufferSize: variabele die aangeeft hoeveel berichten er in de ucTXBuffer en ucRXBufferbuffer passen.
- ucTXBuffernumber: varaibele die aangeeft hoeveel berichten er in de ucTXBuffer zitten
- ucRXBuffernumber: varaibele die aangeeft hoeveel berichten er in de ucRXBuffer zitten
- ucTXBufferSendNr: hulpvariabele.
- ucTXBuffer[][]: twee diemensionale buffer voor het opslaan van te verzenden berichten
- ucRXBuffer[][]: twee diemensionale buffer voor het opslaan van ontvangen berichten
- ucRXBufferMessageLength[]: buffer die paralel loopt aan de ucRXBuffer. in deze buffer wordt opgeslagen hoe lang het bericht in de ucRXBuffer is.
- bSendTimer: variabele die om de 1ms een puls geeft. pas als er een puls gegeven is kan er een bericht verzonden worden. hiermee kan de zendsnelheid geregeld worden
- uliSendTimer; hulp variabele
- bTXBufferSending: variabele die aangeeft of de ucTXBuffer verzonden wordt
- bMessageFreqTimer: variabele die om de 1 seconde een puls geeft. word gebruikt voor ucSendMessagesFreq en ucRecMessagesFreq.
- uliMessageFreqTimer: hulp variabele voor bMessageFreqTimer
- cStatusTekst[25]: string waarin een tekst geprint wordt afhankelijk van TcpIpStatus
- TcpIpStatus: bevat de status van de tcp/ip verbinding
- bMakeConnection: variabele die om de 1 seconde een puls geeft. als er een puls is zal _AutoConnectieTcpIp verbinding proberen te maken met het ip adres in cpTcpIp
- uliTimerMakeConnection: hulp variabele voor bMakeConnection

**Figuur 9.2: struct hcmHardwareDataSetTcpIp**

Ontvangen en te versturen berichten worden eerst geplaatst in een twee dimensionale buffer waarin in zowel de ontvangst als de zend buffer plaats is voor 100 tcp/ip berichten met een lengte van maximaal 200 bytes[1]. Een HCM TCP/IP kan één tcp/ip socket ondersteunen. Deze socket heeft een ontvangt en zend poort en een ip adres waarmee die verbinding maakt en waarnaar hij berichten kan zenden. De HCM TCP/IP kan in twee mode werken server en client. In client mode zal de socket van de HCM TCP/IP automatische verbinding proberen te maken met het ip adres waarmee die verbinding moet maken. In server mode wacht de HCM TCP/IP af tot er een verbinding request binnen komt.

De HCM TCP/IP houd bij hoeveel berichten er zijn ontvangen en moeten verzonden. Op deze manier kunnen CM en EM die verbonden zijn aan een HCM TCP/IP zien of er berichten zijn binnengekomen. En weet de HCM TCP/IP of die nog berichten moet

[1] Zie hoofdstuk HCM TCP/IP in 14.2 Bijlage Test rapporten HCM, CM en EM

verzenden. Bij het plaatsen van berichten in de zend buffer geeft de functie die dit doet feedback of dit gelukt is (dit lukt bijvoorbeeld niet als de buffer vol zit). Op deze manier weten CM en EM die verbonden zijn aan de HCM TCP/IP of het plaatsen van het bericht gelukt is en welke vervolg actie zij kunnen ondernemen als het niet gelukt is. Daarnaast worden er nog bijgehouden hoeveel berichten er per seconde verzonden worden.

**Functies**
De HCM TCP/IP heeft een groot aantal functie. Hieronder volgt een overzicht van de functies. De drie belangrijkste zullen nader toegelicht worden.

- _InitTcpIp: hiermeer wordt de HCM TCP/IP geinitialiseerd. Er wordt in de functie argumenten opgegeven wat de zend en ontvangspoort is, het IP adress waarmee de socket moet verbinden en de mode (client of server). Deze functie roept ook de functie _CreateServerHostTcpIp aan.
- _CreateServerHostTcpIp: functie waarmee de socket aangemaakt wordt.
- _ConnectTcpIp: fuctie waarmee de HCM TCP/IP de socket opdracht geeft verbinding te maken met een socket op het IP adres waarmee hij verbinding moet maken.
- _DisconnectTcpIp: functie waarmee HCM TCP/IP de socket opdracht geeft de verbinding te verbreken. Waarbij de verbinding netjes wordt afgesloten. De client of server geeft aan de verbinding te verbreken en wacht tot de ander ook de verbinding verbreekt.
- _AbortTcpIp: functie waarmee HCM TCP/IP de socket opdracht de verbinding zonder overleg te verbreken.
- _DestroyTcpIp: functie waarmee HCM TCP/IP de socket kan verwijderen
- _SendTcpIp: functie waarmee een array van chars verzonden kan worden in een tcp/ip bericht. Het adres van de HCM TCP/IP die het bericht moet verzenden, de array en de lengte van de array worden opgegeven in de functie argumenten.
- CheckStatusTcpIp: is een functie waarmee de status van de socket wordt uitgelezen. Afhankelijk van de status wordt een string bestaande uit een array van chars in de HCM TCP/IP struct gevuld met een tekst. Vb: socket status is 2. Dit betekent dat de sockt verbinding heeft als client. De sting wordt gevuld met de tekst: Verbinding als Client. Verder wordt in deze functie berekend hoeveel berichten er per seconde ontvangen of verzonden zijn.
- _AutoConnectieTcpIp: fuctie die om de seconde controleert of de socket nog een verbinding heeft als client. Als dit niet het geval is zal deze functie automatische de _ConnectTcpIp functie aanroepen. Deze functie wordt alleen gebruikt als de HCM TCP/IP in client mode werkt.
- _UpdateTcpIp: deze functie roept de _SendTXbufferTcpIp, _ReceiveTcpIp, v en als de HCM TCP/IP in client mode werk ook de _AutoConnectieTcpIp functie aan. De _UpdateTcpIp functie wordt aangeroepen wordt in het hoofdprogramma van de HCM.

De laatste drie functies worden nader toegelicht. Met de _ReceiveTcpIp functie wordt gecontroleerd of er nog nieuwe berichten zijn binnen gekomen. Als dit het geval is zullen alle berichten die binnengekomen voor 1 programma cycli geplaatst worden in de ontvangstbuffer. Er kan niet gedetecteerd worden wat de lengte van het ontvangen TCP/IP bericht is. Dit zal de ontvanger van te voren moeten weten of moeten zien in het ontvangen bericht. De variabele ucRXBuffernumber geeft aan hoeveel berichten er ontvangen zijn. Aan de hand van die variabele kunnen andere CM en EM verbonden aan de HCM TCP/IP zien of en hoeveel berichten er zijn ontvangen. De variabele ucRecMessages gebruikt om ook bijhoud hoeveel berichten ontvangen zijn. Die variabele

wordt gebruikt door functie CheckStatusTcpIp gebruikt voor het berekenen van de ontvangen berichten per seconde.

Met de _FillTXbufferTcpIp kan een CM of EM die aan de HCM TCP/IP verbonden is berichten plaatsen in de zend buffer van de HCM TCP/IP. Er kunnen alleen berichten in de zendbuffer geplaatst worden als de zendbuffer nog niet vol zit en de functie _SendTXbufferTcpIp niet bezig is de zendbuffer te verzenden (bTXBufferSending==0). Als het gelukt is om een bericht in de zendbuffer te plaatsen geeft de _FillTXbufferTcpIp functie een return waarde 1 en wordt de variabele ucTXBuffernumber met 1 verhoogd. Als het plaatsen niet gelukt is geeft de _FillTXbufferTcpIp functie een return waarde 0. Aan de hand van de return waarde weet de CM of EM die aan de HCM TCP/IP gekoppeld en de functie _FillTXbufferTcpIp aanriep of het plaatsen van het bericht gelukt is.

Met de _SendTXbufferTcpIp functie worden berichten geplaatst in de zendbuffer verzonden. Als bij het aanroepen van de _SendTXbufferTcpIp functie de zendbuffer gevuld is met berichten, wordt een variabele genaamd bTXBufferSending op één gezet. Daarna verzend _SendTXbufferTcpIp één bericht per programma cycli. Pas als de buffer weer leeg is zal de variabele bTXBufferSending op nul gezet worden. zolang de variabele bTXBufferSending op één staat kan de _FillTXbufferTcpIp geen berichten plaatsen in de buffer.

### 9.1.5 HCM CAN

Met de HCM CAN kunnen CAN berichten verstuurd en ontvangen worden via een CAN bus.

**CAN bus**

Canbus is een bussysteem dat zijn oorsprong vind in de autotechniek. Vroeger werden alle elektronische systemen parallel met elkaar doorverbonden d.m.v. grote dikke draadbomen. Dit nam veel ruimte in, daarom werden die vervangen door een bus systeem. Via CAN bus kunnen verschillende apparaten met elkaar communiceren. In tegenstelling tot modbus werk CAN bus niet met een master en slave principe. Alle apparaten kunnen zelfstandig berichten verzenden over een CAN bus. De prioriteit wordt bepaald door de eerste elf bits van het ID bericht.

| Eerste 11 bits van een CAN ID | | | |
|---|---|---|---|
| 00000000000 | 00001111111 | 00000100000 | 11111111111 |
| Hoogste prioriteit | Hoge prioriteit | lagere prioriteit | Laagste prioriteit |

**Tabel 9.3: voorbeeld prioriteiten van hoog naar laag**

Berichten met een hogere prioriteit krijgen automatische voorrang. Er zijn twee CAN versies die algemeen toegepast worden: CAN 2.0A en CAN 2.0B. het grootste verschil tussen deze twee is dat het ID van het bericht bij versie A 11 bit breed is en bij versie B 29 bit breed. Een CAN bericht bestaat uit drie onderdelen die door de zender ingevuld moeten worden.

- ID: get ID van het bericht. Aan de hand van het ID kan een ontvanger bepalen wat die met het bericht moet doen en wat het bericht voor informatie bevat. Tevens geeft de ID de prioriteit van het bericht aan. Het
- Data lengte: geeft aan hoeveel byte data er verzonden wordt in de Data Field.
- Data Field: dit veld dat 1 tot 8 byte groot kan zijn bevat de data die verzonden moest worden.

**Struct**

| struct hcmDataSetCan: bevat alle gegevens van een HCM CAN |
|---|
| •uiBitRate: steld de snelheid op het CAN netwerk in. ((128k, 250k, 500k of 1M) |
| •CANBASE_MSG cmCanTxMessages[]: buffer voor te zenden CAN messages |
| •CANBASE_MSG cmCanRxMessages[]: buffer voor ontvangen CAN messages |
| •ucTxbufferNr: geeft aan hoeveel berichten de cmCanTxMessages bevat |
| •ucRxbufferNr: geeft aan hoeveel berichten de cmCanRxMessages bevat |
| •ucTxbufferLength: geeft aan hoeveel berichten er passen in cmCanTxMessages maximaal |
| •ucRxbufferLength: geeft aan hoeveel berichten er passen in cmCanRxMessages maximaal |
| •bNewRXMessage: geeft of er nieuwe berichten zijn ontvangen |
| •bCanVersion: geeft aan welke CAN versie er gebruikt wordt (2.0A of 2.0B) |
| •bCanPort: geeft aan welke canpoort er gebruikt wordt (poort 1(0) of poort 2(1)). |
| •usiRecMessage: wordt opgehoogd als er een nieuw bericht is ontvangen. |

**Tabel 9.4: struct hcmDataSetCan**

**Functionaliteit**
De HCM CAN ondersteund de mogelijkheid om canbus berichten te maken verzenden en ontvangen. De HCM CAN ondersteund CAN 2.0A en CAN 2.0B en kan werken op snelheden 128kbit/sec t/m 1000kbit/sec.

De HCM CAN houd bij en hoeveel berichten er zijn ontvangen. Op deze manier kunnen CM en EM die gekoppeld zijn aan een HCM CAN zien of er nieuwe CAN berichten zijn binnengekomen.

**Functie**
Met de _InitCan functie kan een HCM CAN geïnitialiseerd worden. Als functie argument worden het adres van de HCM CAN die geïnitialiseerd moet worden opgegeven, de can poort waarmee de HCM CAN de canberichten gaan verzenden en ontvangen, de snelheid waarmee de can poort gaat werken (uiBitRate), en de welke can versie er gebruikt gaat worden. Afhankelijk van de waarde in de functie argumenten wordt can poort 1 of can poort 2 geïnitialiseerd met en zal die zal werken met de opgegeven snelheid.

Met de _FillTxBufferCan kan een can bericht aangemaakt en opgeslagen worden in de zendbuffer van de HCM CAN. Bij deze functie worden als extra functie argumenten het ID van het CAN bericht opgegeven deze is afhankelijk van de can versie 11 of 29 bit breed, een array van chars waarin de data van het canbericht staat(1-8 byte), de hoeveel data die verzonden moet worden (1-8). Alle waarden worden geplaatst in een canbericht opgeslagen in de 1e vrije plaats in de zendbuffer. De _FillTxBufferCan kan geen berichten opslaan als de buffer al vol is. De functie stuurt dan een nul als returnwaarde zodat de CM of EM die de functie aanriep weet dat het aanmaken van een CAN bericht mislukt is.

Met de _SendTxBufferCan kan de zendbuffer verzonden worden. Bij deze functie worden geen extra functie argumenten meegegeven. In tegenstelling tot de HCM TCP/IP wordt de zendbuffer van de HCM CAN wel in een programma cycli verzonden. Als er tijdens een programma cycli 9 canberichten zijn aangemaakt dan zullen die in een keer verzonden worden zo gauw de _SendTxBufferCan functie wordt aangeroepen.

Met de _ReadMessageCan fucntie kan wordt gecontroleerd of er een canbericht is ontvangen op de canpoort waarmee de HCM CAN werkt. Als een canbericht is ontvangen zal die geplaatst worden in de ontvangstbuffer van de HCM CAN. En wordt een variabele genaamd bNewRXMessage in de HCM CAN struct op 1 gezet. Hiermee kunnen CM en EM die verbonden zijn aan de HCM CAN controleren of er een nieuw canbericht is binnengekomen.

De _UpdateCan functie wordt iedere programma cycli aangeroepen in het HCM hoofdprogramma. De _UpdateCan roept de _SendTxBufferCan en de _ReadMessageCan functie aan.

## 9.2 Control modules

In het hoofdprogramma van de CM worden tijdens het initialiseren alle _INIT functies van alle CM aangeroepen waarvan objecten zijn gedefinieerd. Tijdens iedere volgende programma cycli worden de _UPDATE functies van de CM aangeroepen. Daarnaast is er nog wat extra functionaliteit ingebouwd voor de CM OILQUA. Zie hiervoor hoofdstuk

### 9.2.1 CM Flow sensor

Met de CM Flow sensor kan een flow sensor uitgelezen worden. Op deze manier kan m.b.v. een flow sensor bijgehouden worden hoeveel diesel er door een leiding loopt. Het is nog niet helemaal duidelijk welke flowsensor er gebruikt gaat worden. Er zijn veel verschillende soorten en maten van aansluitingen. Ook het maximale dieselverbruik speelt een rol en dit is sterk afhankelijk van de dieselinstallatie en motor op het schip. Uit het vooronderzoek is gebleken dat de informatie terugkoppeling van de flowsensor meestal op drie manieren gebeurd

- Modbus: wordt meestal gebruikt bij de duurdere flowsensoren. Aangezien prijs een belangrijke factor is bij het kiezen van de flowsensor is het niet waarschijnlijk dat modbus gebruikt zal worden. Modbus wordt daarom niet geïmplementeerd in de CM Flowsensor.
- Puls signaal: Iedere puls vertegenwoordigd een hoeveelheid diesel in liters. Door het aantal pulsen per seconde te tellen kun je berekenen wat de stroomsnelheid is. Door het totaal aantal pulsen te tellen kan berekend worden wat het totaal verbruik is.
- Analoge signaal (4-20ma): het signaal vertegenwoordigd een stroomsnelheid in l/h of l/min. Hiermee kan gemeten worden wat de stroomsnelheid is. Het is belangrijk dat CM Flow Sensor snel achter elkaar (minimaal om de seconde) de waarde van het analoge signaal gemeten wordt anders kunnen er meetfouten ontstaan bij het berekenen van het totale dieselverbruik.

Het bereken van het (totale) diesel verbruikt wordt gedaan door de EM Fuel Consumption.

struct cmFlowSens: bevat alle gegevens van een CM Flow Sensor, bestaat uit een data en een settings struct

- stSet: bevat alle settings van een cmFlowSens
  - hcmIOData *IO: pointer naar HCM IO
  - cmInternComSend* pInternComSend: pointer naar CM Intern Com Send
  - ucModuleNr: id nummer van het CM Flow Sensor object
  - ucInputNr: bepaald welke DI of AI er gebruikt wordt om de flow sensor uit te lezen
  - ucInputMode: bepaald op welke manier de flowsensor uitgelezen wordt (puls of analoog)
  - fConversion: wordt gebruikt voor het bereken van de stroomsnelheid
  - fOffset: wordt gebruikt voor het bereken van de stroomsnelheid
- stData: bevat alle data van de cmFlowSens
  - float fFlow: hierin wordt de uitgelezen stroomsnelheid van de flowsensor opgeslagen in l/h
  - bConnectionError: variabele die op één gezet wordt als er geen verbinding meer is met de flow sensor

**Figuur 9.3 struct cmFlowSens**

**Functionaliteit**
Zoals eerder aangegeven ondersteund de CM Flow Sensor alleen de mogelijkheid om de flowsensor uit te lezen m.b.v. een puls signaal of een analoog signaal. Voor het analoge signaal wordt iedere 100ms gemeten wat het de waarde van het analoge signaal is. Het puls signaal wordt ook iedere 100ms uitgelezen als er opnieuw berekend is hoeveel pulsen er zijn binnengekomen in de afgelopen 100ms. Doormiddel van een conversie en een offset variabele in de struct kan de analoge waarde van het ingangsignaal of het aantal pulzen omgerekend worden naar l/s. Als er gemeten wordt m.b.v. een analoog signaal kan bekeken worden of de sensor nog steeds verbonden is aan de controller als gedetecteerd wordt dat dit niet meer het geval is zal een error variabele op één gezet worden. Daarnaast is het mogelijk om data van de CM te versturen naar een andere controller. Op deze manier kan data centraal worden opgeslagen en verstuurd naar de PC waar de data gevisualiseerd en gelogd wordt.

**Functies**
Met de _InitFlowSens functie kan een CM Flow Sensor geïnitialiseerd worden. Als extra functie argumenten worden meegegeven

- ModuleNr: uniek id nummer van de CM Flow Sensor
- hcmIOData *: Pointer naar HCM IO wordt gebruikt om een analoog of pulssignaal uit te lezen.
- cmInternComSend *: Pointer naar CM Intern Com Send. Een CM die de data van een CM of EM kan versturen naar een andere controller.
- Mode: Bepaald of de CM Flow Sensor werkt met een analoog of een puls signaal
- Input: bepaald op welke plaats in de DI puls of AI array respectievelijk de waarde van het aantal pulsen of de waarde van het analoge signaal worden opgeslagen.
- Conversion: wordt gebruikt om de pulsen of de analoge waarde om te zetten naar l/s.
- Offset: in l/h. wordt gebruikt om de pulsen of de analoge waarde om te zetten naar l/s.

Met de _ReadFlowSens functie kan de CM Flow Sens om de 100ms het analoge signaal uitlezen of bekijken hoeveel pulzen er de afgelopen 100ms zijn binnengekomen. Bij de pulzen wordt de stroomsnelheid in l/s berekend met de volgende formule.

$$Stroomsnelheid = (pulzen\ perseconde * conversie\ variable) + offset\ variabele$$

De conversie variabele is het aantal liters dat een puls vertegenwoordigd. Offset is bijna altijd nul en eigenlijk niet gebruikt.

Bij een analoog signaal wordt de waarde eerst omgezet naar mA en daarna wordt de stroomsnelheid in l/h berekend aan de hand van de volgende formule.

$$Stroomsnelheid = (meetwaarde(mA) * conversie\ variable) + offset\ variabele$$

Het berekenen van de conversie variabele is hier wat lastiger en moet de gebruiker zelf doen. De range van het analoge signaal (4-20ma) is 16 (20-4) vertegenwoordigd een bepaalde range van de flowmeter en wordt berekend met de volgende formule

$$conversion = \left(\frac{(x\ l/s\ @20mA) - (x\ l/s\ @4mA)}{20mA - 4mA}\right) * 3600$$

$$voorbeeld: \left(\frac{(0{,}5\ l/s\ @20mA) - (0\ l/s\ @4mA)}{20mA - 4mA}\right) * 3600 = 0{,}03125 * 3600 = 112{,}5\ {}^{l}/_{h}/ma$$

Als de stroomsnelheid berekend is zal die opgeslagen worden in de fFlow variabele van de CM Flow Sensor struct. Als er van een analoog meetsignaal gebruik gemaakt wordt kan de _ReadFlowSens detecteren of het signaal binnen het 4-20mA meetbereik blijft. Als die er onder de 4mA zakt is er iets mis en is de waarde van het analoge signaal niet meer betrouwbaar en wordt de variabele bConnectionError op één gezet. De bConnectionError wordt gereset als het signaal weer een betrouwbare waarde binnenkrijgt (binnen de 4-20mA band).

Met de _SendDataFlowSens functie wordt de data struct in de CM Flow Sensor struct omgezet in een array van chars die verzonden kan worden over een tcp/ip verbinding m.b.v de CM Intern Com Send. Controller die de array ontvangt zet die weer om naar de data struct van de CM Flow Sensor. Op deze manier kan de data van alle CM en EM centraal opgeslagen worden. De inhoud van de functie wordt alleen uitgevoerd als een variabel genaamd pInternComSend in de CM Intern Com Send op één gezet is. De eerste drie bitjes van de array geven aan welke datastuct in de array zit.

Met de _UpdateFlowSens functie worden de _ReadFlowSens en de _SendDataFlowSens functie aangeroepen. De _UpdateFlowSens wordt iedere programma cycli aangeroepen door het hoofdprogramma van de CM.

### 9.2.2 CM Pressure sensor

Deze module wordt gebruikt op een druksensor uit te lezen m.b.v. analoge ingangen op de controller. De druksensor verstuurd deze waarde d.m.v. een analoog signaal 4-20mA.

**Struct**

struct cmPress: bevat alle gegevens van een CM Pressure sensor, bestaat uit een data en een settings struct

- stSet: bevat alle settings van een cmPress
  - *pIO: pointer naar de struct van HCM IO. hiermee kunnen de waardes van de AI uitgelezen worden.
  - ucAInr: bevat het nummer van de analoge ingang waarop de druksensor is aangesloten.
  - ucModuleNr: het ID nummer van de CM Press
  - float fConversion;
  - float fOffset;
- stData: bevat alle data van de cmPress
  - bConnectionError: is 1 als de analoge ingang lager dan 4ma meet
  - fPressure: bevat de waarde van de oliedruk

**Figuur 9.4: struct cmPress**

**Functionaliteit**

De CM Press ondersteund de mogelijkheid om zelfstandig de druk af te lezen m.b.v. de analoge ingangen op de controller. Daarnaast controleert de CM Press of het analoge meetsignaal binnen de vastgestelde meetband van 4-20 mA blijft. Bij een waarde onder de 4ma wordt de bConnectionError variabele in de struct van de CM Press op 1 gezet. Hiermee kan de gebruiker gewaarschuwd worden dat de sensor die door de CM uitgelezen wordt niet goed werkt.

**Functie**

Met de _InitPress wordt de CM Press geïnitialiseerd. Als extra functie argumenten worden opgegeven:

- ModuleNr: id nummer van het CM Press object
- hcmIOData*: pointer naar HCM IO. Hiermee worden de AI uitgelezen
- cmInternComSend*: pointer naar CM Intern Com Send. Deze CM kan de datastuct van de CM Press verzenden over een tcp/ip verbinding.
- AInr: variabele die aangeeft op welke AI de het signaal van de druksensor binnenkomt
- Conversion: wordt gebruikt om de analoge waarde om te zetten naar bar.
- Offset: wordt gebruikt om de analoge waarde om te zetten naar bar.

Met de _ReadFlowSens functie kan de CM Press Sensn het analoge signaal uitlezen Bij een analoog signaal wordt de waarde eerst omgezet naar mA en daarna wordt de druk in bar berekend aan de hand van de volgende formule.

$$Druk = (meetwaarde(mA) * conversie\ variable) + offset\ variabele$$

Het berekenen van de conversie variabele moet de gebruiker zelf doen. De range van het analoge signaal (4-20ma) is 16 (20-4) en vertegenwoordigd een bepaalde range van de drukmeter en wordt berekend met de volgende formule

$$conversion = \left(\frac{(x\ bar\ @20mA) - (x\ bar\ @4mA)}{20mA - 4mA}\right)$$

$$voorbeeld: \left(\frac{(24\ bar\ @20mA) - (8\ bar\ @4mA)}{20mA - 4mA}\right) = 1\ bar/mA$$

Het berekenen van een eventuele offset gaat met de volgende formule

$$offset = x\ bar\ @4mA$$

$$voorbeeld(8\ bar\ @4mA):\ \ offset = 8\ bar$$

Als de druk berekend is zal die opgeslagen worden in de fPress variabele van de CM Press struct. De _ReadFlowSens kan detecteren of het signaal binnen het 4-20mA meetbereik blijft. Als die er onder de 4mA zakt is er iets mis en is de waarde van het analoge signaal niet meer betrouwbaar en wordt de variabele bConnectionError op één gezet. De bConnectionError wordt gereset als het signaal weer een betrouwbare waarde binnenkrijgt (binnen de 4-20mA band).

De CM Press heeft een SendDataPress fuctie. Deze functie heeft de zelfde functionaliteit als de SendDataFlowSens. Voor de functionaliteit van de SendDataFlowSens functie zie hoofdstuk 9.2.3 CM Oil quality sensor.

Met de _UpdatePress worden de _ReadDataPress en de _SendDataPress aangeroepen. _UpdatePress wordt iedere programma cycli aangeroepen in het hoofdprogramma van de CM.

### 9.2.3 CM Oil quality sensor

Deze CM OilQua wordt gebruikt om een oliekwaliteitssensor uit te lezen. De oliekwaliteitssensor meet de oliekwaliteit, de olietemperatuur en de omgevingstemperatuur rond de oliekwaliteitssensor. De gegevens kunnen uitgelezen worden d.m.v. een modbus verbinding over een RS485 verbinding, via een CAN verbinding en via twee analoge signalen. Bij het gebruik van de analoge signalen kunnen alleen de oliekwaliteit en de olietemperatuur uitgelezen worden. Bij de CM OilQua en de Cm Press hoort ook een EM die de data van deze CM weergeeft op het scherm van een syel controller. Deze EM wordt niet gebruikt in het SMMS en wordt dus niet verder behandeld in deze scriptie. Wel nam het ontwikkelen van de CM OilQua, de CM Press en de EM meer tijd in beslag dan voorzien. Hierdoor is door tijd nood de webapplicatie nog niet volledig ontwikkeld.

**Struct**

struct cmOilQua: bevat alle gegevens van een OilQua CM, bestaat uit een data en een settings struct

- stSet: bevat alle settings van een OilQua CM
  - *pModBus: pointer naar een struct van HCM Modbus. Hiermee worden alle modbus bericht verzonden en uitgelezen
  - *pIO: pointer naar de struct van HCM IO. hiermee kunnen de waardes van de AI uitgelezen worden.
  - struct cmInternComSend *pInternComSend: pointer naar een struct van cmInternComSend hiermee kan de datastruct van de cmOilQua worden verzonden ;
  - ucDevNr: device nummer van de oliekwaliteitssensor wordt gebruikt voor de modbus berichten
  - ucAInr[2]: de nummers van de analoge ingangen waarop de oliekwaliteitssensor is aangesloten.
  - bUpdateDone: geeft aan of een cmOilQua klaar is met het uitlezen van een oliekwaliteitssensor via modbus
  - bMessageSend: geeft aan een message is verzonden via modbus voor het uitlezen van de sensor.
  - ucModuleNr: indetificatie nummer van de cmOilQua
  - ucErrorCount: telt het aantal keer dat er geen verbinding gemaakt kon worden via modbus.
- stData: bevat alle data van de OilQua CM
  - bConnectionError: is 1 als er te vaak geen verbinding gemaakt kon worden via modbus. of als de analoge ingangen lager dan 4 ma meten
  - fOilTemp: bevat de waarde van de olietemperatuur
  - fOilQuality: bevat de waarde van de oliekwaliteit
  - fAmbientTemp: bevat de waarde van de omgevings temperatuur.

**Figuur 9.5: struct cmOilQua**

**Functionaliteit**

De CM ondersteund de mogelijkheid om de gegevens d.m.v. modbus en analoge ingangen uit te lezen. De CAN wordt nog niet ondersteund door deze CM. De CAN stond als laagste op de prioriteiten lijst voor deze CM, aangezien deze databus ook gebruikt wordt voor het uitlezen van de motor en het aansturen van remote IO. Met maximaal twee CAN aansluitingen is er een grote kans dat deze databussen al in gebruik zijn.

De uitgelezen waardes worden opgeslagen in 3 float variabelen in de struct van de CM. Door de wel of niet initialiseren van de pointer naar een HCM Modbus kan aangegeven worden of de CM d.m.v. modbus of analoge signalen de waardes van de sensor uitleest. Het is mogelijk om de datastruct van de CM Oilqua struct te verzenden. Hierdoor kan de data van alle datastructs centraal opgeslagen worden. Hiervoor moet bij het initialiseren van de CM Oilqua wel de cmInternComSend pointer gedefinieerd worden. Als die niet gedefinieerd is wordt de data niet verstuurd.

**Functies**

Met behulp van de Init functie wordt een struct voor een CM OilQua geïnitialiseerd. Hierbij wordt de ID nummer van de CM doorgegeven en wordt gekozen of de CM aan de HCM IO of aan een HCM Modbus gekoppeld word. Verder wordt bij uitlezen d.m.v.

modbus, aangeven wat het device nummer van de sensor is. In het geval van analoge uitlezing worden de nummers van de analoge ingangen opgegeven. Functie argument die niet gebruikt worden zijn nul. Alle gegevens die niet d.m.v. functie argumenten geïnitialiseerd worden zijn 0. Behalve de bUpdateDone die wordt geïnitialiseerd op 1 aangezien deze variable aangeeft of de CM meetgegevens mag opvragen via de Modbus.

Met behulp van de _ReadDataOilQua functie worden meetgegevens van de oliekwaliteitssensor uitgelezen via de analoge ingangen. De CM controleert op dezelfde manier als de CM Press of het signaal binnen de afgesproken meetwaarde blijft van 4-20 ma. Uitlezen van de meetgegevens via de modbus gaat m.b.v. twee functies.

De eerste functie _ReqDataOilQua verzend een modbus verzoek bericht naar de oliekwaliteitssensor. Dit gebeurd alleen als in het CM hoofdprogramma is aangegeven dat de CM OilQua een request mag verzenden. Dit wordt aangegeven door de variabele bUpdateDone op nul te zetten. De functie controleert of het bericht succesvol in de berichten buffer van de HCM modbus is geplaatst. Als de oliekwaliteitssensor dit modbus bericht ontvangt zal die een bericht terugsturen met daarin de waarden van de olietemperatuur, omgevingstemperatuur en de oliekwaliteit.

Met de _RecDataOilQua functie bekijkt de CM OilQua of de HCM modbus een bericht ontvangen heeft en controleert of het bericht van de juiste oliekwaliteitssensor komt en of het bericht de juiste modbusfunctie (4) bevat. Als het bericht klopt worden de waarden van de registers 0 t/m 2 van de oliekwaliteitssensor geconverteerd en geplaatst in de juiste variabelen in de struct van de CM OilQua. De variabele bUpdateDone wordt dan op nul gezet en een volgende CM OilQua mag een request verzenden.

De RecDataOilQua detecteert ook als er geen antwoord op een verzonden bericht ontvangen wordt. Als er geen bericht ontvangen wordt zal de bUpdateDone variabele op nul worden gezet en mag een volgende CM OilQua een request verzenden. Als er 5x geen antwoord ontvangen wordt door de CM OilQua bij het versturen van een request gebeurd wordt de variabele bConnectionError op 1 gezet. Hiermee wordt aangegeven dat er geen verbinding meer gemaakt kon worden met de oliekwaliteit sensor. De gebruiker van het SMMS systeem kan dan hierop actie ondernemen.

De CM OilQua heeft een SendDataOilQua functie. Deze functie heeft de zelfde functionaliteit als de SendDataFlowSens. Voor de functionaliteit van de SendDataFlowSens functie zie 9.2.1 CM Flow sensor.

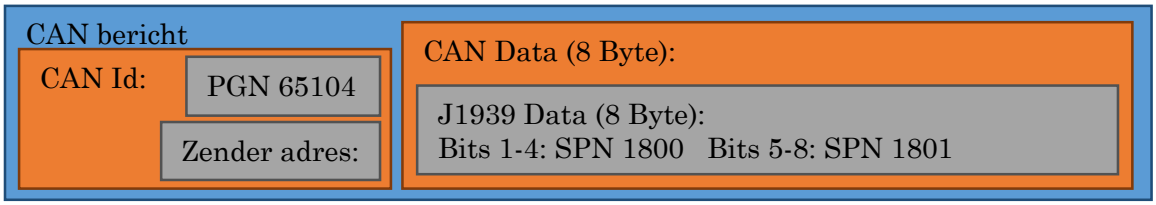
De _UpdateOilQua roept de _ReadDataOilQua, _ReqDataOilQua, _RecDataOilQua en _SendDataOilQua functie aan. De _UpdateOilQua wordt iedere programma cycli in het hoofdprogramma van de CM aangeroepen.

### 9.2.4 CM J1939

Met de CM J1939 kunnen motorgegevens die in in een J1939 protocol zijn verstuurd over een canbus uitgelezen worden.

**J1939**

Het J1939 protocol is het berichtformaat waarin de gegevens van de motor worden verstuurd. Het J1939 protocol wordt verpakt in een CAN bericht. Ieder J1939 bericht heeft een PGN (Parameter Groep Number) de PGN geeft aan welk bericht er verstuurd/ontvangen is bijvoorbeeld PGN 65104 staat voor de batterij (accu) temperatuur. Iedere J1939 data uit een aantal SPN (Suspect Parameter Number) de PGN 65104 bevat de SPN 1800 en 1801 deze SPN staan voor de batterij temperatuur 1 en voor batterij temperatuur 2. In het CAN id wordt ook het adres van de zender mee gegeven.




**Figuur 9.6 Can Frame met J1939 data**

**Struct**

De meeste struct van de CM en EM zijn opgedeeld in twee delen een data en een settingstruct. In de datastruct zit nog een array van struct waarin alle data van de verschillende SPN in opgeslagen. De SPN struct is ook weer opgedeeld in een setting en data struct. In de settingstruct van de SPN struct wordt aangegeven op welke bit van een PGN zich de data van de SPN bevind en een variabele waarmee de binaire waarde geconverteerd kan worden naar zijn echte waarde. Bijvoorbeeld een binaire getal waarin de waarde zit van een druk meter. In de datastruct van de SPN struct wordt de echte waarde opgeslagen en wordt aangegeven bij welke pgn en spn de waarde hoort. Verder wordt de scource opgeslagen, op die manier kan terug gevonden worden van welk apparaat de waarde afkomstig is.

struct cmJ1939 : bevat alle gegevens van een CM J1939 , bestaat uit een data en een settings struct

- stSet: bevat alle settings van een CM J1939
  - hcmDataSetCan * pCanbus: pointer naar HCM CAN waarmee de J1939 berichten worden ontvangen.
  - cmInternComSend *pInternComSend: hiermee wordt de datastuct van de CM J1939 verstuurd
  - SPNBufferWidth: legte van SPM buffer
  - ucModuleNr: id nummer van het CM J1939 object
- stData: bevat alle data van de CM J1939
  - struct SPN[]: array van SPN struct .

**Figuur 9.7 struct cmJ1939**

struct PSN: bevat alle gegevens van een SPN, bestaat uit een data en een settings struct

- stSet: bevat alle settings van een SPN
  - ucStartByte: variabele waarmee berekend wordt in welke bits van de PGN de data van de SPN zit
  - ucEndByte: variabele waarmee berekend wordt in welke bits van de PGN de data van de SPN zit
  - ucOffset: variabele waarmee de binaire data omgerekend kan worden naar een decimaal getal
  - fMultiply: variabele waarmee de binaire data omgerekend kan worden naar een decimaal getal
- stData: bevat alle data van de SPN
  - usiPGN: PGN nummer waarin de data in deze SPN struct staat.
  - usiSPN: SPN nummer waarin de data van deze SPN struct staat
  - ucScource: scource nummer waarvan de data afkomstig is
  - fValue: decimale representatie van de data van de SPN

**Figuur 9.8 struct SPN**

### Functionaliteit

De CM J1939ondersteun de functionaliteit om J1939 berichten te maken, ontvangen en verzenden. De CM J1939 kan zelfstandig de SPN data uit een J1939 bericht halen en opslaan in de array van SPN in de struct van CM J1939. De SPN struct array in de struct van de CMJ1939 kan gezien worden als het naslag werk voor de CM J1939. Aangezien er een paar honderd SPN bestaan in het J1939 protocol zijn zij niet allemaal gedefinieerd in de SPN array. Alleen SPN die de gebruiker vooraf heeft gedefinieerd in de SPN array worden herkend door de CM J1939. SPN die niet gedefinieerd zijn worden genegeerd. Om een SPN te definiëren moet de gebruiker vooraf de settings en data struct van de SPN struct invullen m.u.v. de variable fValue. De gegevens die ingevuld moeten worden kunnen gevonden worden in de meeste datasheets die het J1939 protocol behandelden. Als een J1939 bericht ontvangen wordt zal de decimale representatie van de SPN data ingevuld worden in de fValue variabele in de PGN struct die hoort bij de PGN in het ontvangen bericht.

Net als alle andere CM en EM heeft de CM J1939 de mogelijkheid om de data te verzenden naar een centraal opslagpunt. De implementatie hiervan verschild iets van de andere CM en EM.

De CMJ1939 herkend in ieder geval de volgende SPN nummers:

- SPN 190: voor het meten van de RPM van de motor
- SPN 100: voor het meten van de Oliedruk in de motor
- SPN 110: voor het meten van de koelwater tempreratuur in de motor
- SPN 174: voor het meten van de diesel temperatuur in de motor

### Functies

Met de _InitJ1939 wordt een CM J1939 geïnitialiseerd. Als extra variabelen worden opgegeven:

- modelNr : is het id nummer van het CM j1939 object en wordt geplaatst is ucModuleNr van de CM J1939 struct
- hcmDataSetCan*: bevat een adres van een HCM CAN waarmee de J1939 berichten geplaatst ontvangen of verzonden worden. adres wordt geplaatst in hcmDataSetCan van de CM J1939 struct
- cmInternComSend*: bevat een adres van een CM Intern Com Send waarmee data van de CM J1939 verstuurd kan worden. Het adres wordt geplaatst in cmInternComSend van de CM J1939 struct

Verder roept de _InitJ1939 functie de _InitSPN functie aan. Met die functie wordt alle door de gebruiker gedefinieerde SPN data ingeladen in de SPN array. M.b.v. die data kan de CM J1939 SPN herkennen in J1939 berichten.

Met de _WriteMessageJ1939 functie kan een J1939 berichten opgesteld worden en geeft de _WriteMessageJ1939 functie en roept de _FillTxBufferCan functie van de HCM CAN aan die dit bericht in de zendbuffer van HCM CAN zal plaatsen. De HCM CAN handelt het verzenden van het bericht verder af.

Met de _ReadMessageJ1939 wordt een J1939 bericht uit de ontvangstbuffer van HCM CAN gehaald. Vervolgens wordt het canbericht uitgepakt en bekeken wat de scource en het PGN nummer van het canbericht is en wordt data van het canbericht in een tijdelijke array gestopt. Daarna worden de SPN struct array in de J1939 struct doorgekeken en word bekeken of de PGN en scource uit het bericht overeenkomen met de voor gedefinieerde PGN en scource waarde die de gebruiker in de SPN struct array heeft ingevoerd. Als dit het geval is wordt de _CalcValueJ1939 functie aangeroepen die de data van de SPN uitleest uit de tijdelijke array. En de fValue berekend van de SPN.

De _CalcValueJ1939 is een hulpfunctie functie die de waarde van de fValue van de SPN berekend aan de hand van de gegevens in de setting struct van de SPN struct. Als vastgesteld is welke bitjes de waarde van de SPN representeren worden deze bitjes geconverteerd naar één binair getal. Hiermee wordt de fValue berekend met de volgende formule:

$$fValue = (data\ van\ SPN(binair) + ucOffset(binair)) * fMultiply$$

De _SendDataJ1939 functie heeft grotendeels de zelfde implementatie als de SendDataFlowSens beschreven onder het kopje functies in hoofdstuk 9.2.1 CM Flow sensor. Alleen de _SendDataJ1939 stuurt niet de datastruct van de struct van de CM J1939. Dit zou bij een grote SPN array in de CM J1939 struct algauw meer dan 200 bytes lang worden. Dan kan de HCM TCP/IP niet meer garanderen dat de verstuurde berichten intact aankomen. Het doel van de CM J1939 is het verzamelen van data van de motor (SPN) daarom hoeft alleen de SPN data centraal opgeslagen te worden. Daarom worden alleen de datastruct van de SPN struct verzonden in aparte berichten. Een CM J1939 met een SPN struct array van negen SPN zal dan ook negen aparte TCP/IP berichten verzenden met in ieder bericht één datastruct van de SPN struct.

De _UpdateJ1939 functie roept de _ReadMessageJ1939 functie en de _SendDataJ1939 functie aan.

### 9.2.5 CM Intern Com Send

De module die de communicatie verzorgt tussen twee syel controllers is gesplitst in twee delen. Dit is handig want dat hoeven niet alle slave syel controllers de functionaliteit mee te dragen voor het ontvangen van berichten van andere syel controllers.

Deze CM wordt gebruikt voor het managen van het versturen van de data structs van CM en EM naar de centrale controller.

| struct cmInternComSend: bevat alle gegevens van een CM Intern Com Send, bestaat een settings struct |
|---|
| • stSet: bevat alle settings van een CM Intern Com Send<br>• cmHardwareDataSetTcpIp* TCPIP: pointer naar een HCM TCP/IP;<br>• bPremToSend: waarde die aan andere CM en EM aangeeft dat zij hun datastruct mogen verzenden; |

**Tabel 9.5 struct cmInternComSend**

**Functionaliteit**

De CM Intern Com Send is de eerste CM die de communicatie tussen de slave en de centrale syel controller verzorgt. De CM Intern Com Send verzorgt het versturen van berichten naar de centrale controller. De CM Intern Com Send is gekoppeld aan een HCM TCP/IP. De centrale controller stuurt een startbericht naar slave controller. De HCM Intern Com Send ontvangt dit bericht m.b.v. HCM TCP/IP. De CM Inter Com Send geeft dan aan dat alle CM en EM in de slave controller hun datastruct kunnen zenden. Als dit gebeurd is stuurt de CM Intern Com Send een eind bericht.

**Functies**

Met de _InitInternComSend wordt de CM Intern Com Send geinitialiseerd. Hierbij wordt de HCM TCP/IP aan de CM Inter Com Send gekoppeld.

De _SendDataInternComSend fuctie wordt door iedere CM en EM aangeroepen die zijn datastruct verzend. Deze functie plaatst de datastruct in de vorm van een array in de zendbuffer van HCMn TCP/IP. De HCM TCP/IP regelt vervolgend het verzenden

Met _RecDataInternComSend bekijkt de CM Com Intern Send of er een startbericht ontvangen is (drie nullen). Dan zal CM Intern Com Send aan de andere CM en EM aangeven dat zijn hun datastruct kunnen zetten. Als dit gebeurd zend de CM Intern Com Send een eindbericht (weer drie nullen).

Met de _UpdateInternComSend wordt de RecDataInternComSend fuctie aangeroepen.

### 9.2.6 CM Intern Com Rec

De CM Intern Com Rec is de tweede module die de communicatie verzorgt tussen twee syel controllers. Daarnaast verorgt die de communicatie tussen de centrale controller en de PC.

| struct cmInternComRec: bevat alle gegevens van een CM Intern Com Rec, bestaat een settings struct |
|---|
| • stSet: bevat alle settings van een CM Intern Com Rec<br>  • pressData stPressData[10]: bevat alle verzamelde datastructs CM Press<br>  • OilQuaData stOilQuaData[10]:bevat alle verzamelde datastructs CM OilQua<br>  • flowData stFlowData[10]: bevat alle verzamelde datastructs CM Flow Sensor<br>  • SPNData stSPNdata[2][20]: bevat alle verzamelde datastructs de SPN in de J1939 structs<br>  • FuelConsumptionData stFuelConsumptionData[10]: bevat alle verzamelde datastructs EM Fuel Consumption<br>  • ucDataString[5000: bevat alle data van de bovenstaande structs in JSON string vorm.<br>• stData: bevat alle data van de CM Intern Com Rec<br>  • cmHardwareDataSetTcpIp* TCPIP[15]: array met alle HCM TCP/IP die de verbinding met de slavecontrollers managen<br>  • hcmDataSetSer* pSer: pointer naar HCM Serieeel waarmee data naar de PC wordt verzonden.<br>  • ucNrOfConnections: geeft aan hoeveel HCM TCP/IP er gebruikt worden (1-15)<br>  • ucX: hulp variabele<br>  • bRXDelay: geeft een puls als er 50ms verstreken zijn na het laatste uitlezen van de variabele. wordt gebruikt voor het geforceerd starten van een nieuw request als een slave controller geen antwoord geeft.<br>  • ulRXDelayState: hulp variable<br>  • ucRXdelayCounter: hulp variabele |

**Tabel 9.6: struct CM Intern Com Rec**

**Functionaliteit**

Met de CM Intern Com Rec kunnen na het zenden van een startbericht antwoordbericht ontvangen worden. Deze antwoordberichten bevatten data van een datastruct van een CM of EM. Aan de hand van de eerste drie chars van de array wordt vastgesteld welke datastruct er ontvangen is. De derde char van de array bevat de unieke modelNr van de CM of EM van zijn soort. Als vastgesteld is welke datastruct er ontvangen is wordt de array geconverteerd naar de datastruct. Als de data struct bijvoorbeeld van een CM Flowsensor is en modelNr drie heeft. Dan wordt deze datastruct opgeslagen in stFlowData array op plaats twee. Want modelNr worden genummer van 1 tot oneiding en de array loopt van plaats nul tot 9.

Als de CM Intern Com Rec een eindbericht ontvangt van een slave controller zal die de alle datastuct array in de datastuct van CM Intern Com Rec omzetten naar een JSON string en verzenden naar de PC

**Functies**

Met de _InitInternComRec functie wordt de CM Intern Com Rec geinitialiseerd. Alle HCM TCP/IP die een verbinding met een slavecontroller managen worden gekoppeld aan de CM Intern Rec. Ook word de HCM Serieel gekoppeld aan de CM Intern Com Rec.

Met de _ReqDataInternComRec functie wordt een start bericht verstuurd naar een slavecontroller. Als van die slavecontroller weer een eindbericht wordt ontvangen wordt de naar de volgende controler een start bericht gestuurd.

Met de _RecDataInternComRec wordt van de slave controller waarnaar een startbericht is gestuurd alle zijn verstuurde data ontvangen. Er wordt bekeken welke datastruct is verstuurd en deze wordt geplaatst op e juiste plaats in de juiste datastruct array. Als er een eindbericht wordt ontvangen zal de _SendPcDataInternComRec functie aangeroepen worden en mag de _ReqDataInternComRec functie een nieuw startbericht sturen.

Met _SendPcDataInternComRec wordt alle data in de datastrcuct array in de datastruct van de CM Intern Com Rec als JSON string verzonden naar de PC.

De _RunInternComRec functie roept de _ReqDataInternComRec en de _RecData-InternComRec functie aan.

## 9.3 Equipment modules

In het hoofdprogramma van de EM worden tijdens het initialiseren alle _INIT functies van alle EM aangeroepen waarvan objecten zijn gedefinieerd. Tijdens iedere volgende programma cycli worden de _UPDATE functies van de EM aangeroepen.

### 9.3.1 EM Fuel Consumption

De EM Fuel Consumption wordt gebruikt om het dieselverbruik van de motor te berekenen. Het meten van het dieselverbruik van een motor gaan m.b.v. twee CM Flow Sensor. Aangezien de dieseltoevoer naar en de dieselafvoer van de motor gemeten moet worden. Het verschil tussen de twee is het dieselverbruik van de motor.

**Struct**

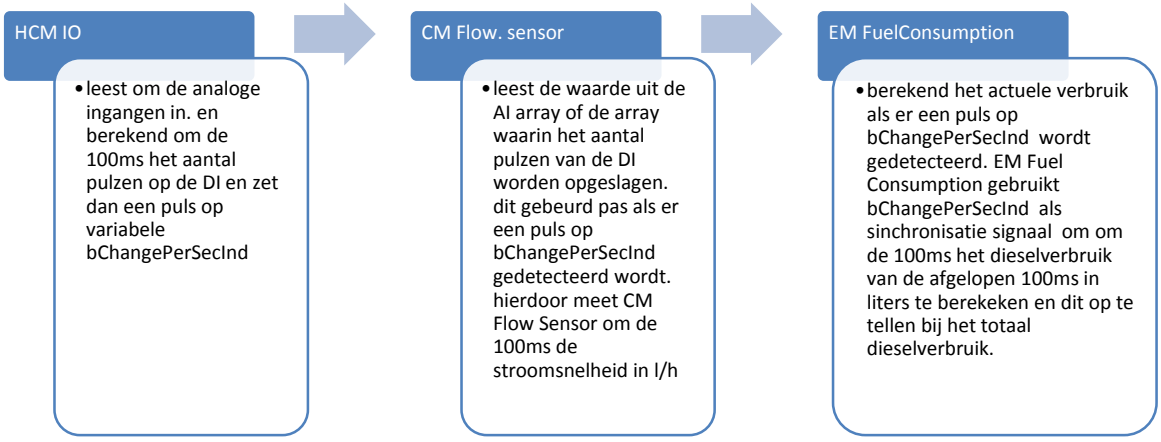
struct emFuelConsumption: bevat alle gegevens van een EM Fuel Consumption , bestaat uit een data en een settings struct

- stSet: bevat alle settings van de EM Fuel Consumption
  - ucModuleNr: id nummer van het EM Fuel Consumption object
  - cmFlowSens* pInFlow: pointer naar een CM Flow Sensor die de diesel toevoer meet
  - cmFlowSens* pOutFlow: pointer naar een CM Flow Sensor die de diesel afvoer meet. is 0 als er maar met 1 flowsensor gemeten wordt
  - cmInternComSend *pInternComSend: hiermee wordt de datastuct van de CM J1939 verstuurd
  - bMode: bepaald of de Em Fuel Consumption met één of met twee CM Flow Sensor werkt.
- stData: bevat alle data van deEM Fuel Consumption
  - fFuelConsumption: bevat het actuele dieselverbruik
  - dTotalFuelConsumed: bevat het totale dieselverbruik

**Figuur 9.9 struct emFuelConsumption**

**Functionaliteit**

De EM Fuel Consumption ondersteund de mogelijkheid om met twee CM Flow Sensoren het actueel dieselverbruik en het totaal dieselverbruik te berekenen. De EM Fuel Cosumption heeft ook de mogelijkheid om met één CM Flow Sensor het dieselverbruik te meten. Dit wordt gebruikt als er alleen een dieseltoevoer is naar de motor en er maar met één flowmeter gemeten wordt. Het berekenen van het dieselverbruik gebeurd om de 100ms.Op het zelfde moment dat de flowsensor is uitgelezen en door CM Flow Sensor het de stroomsnelheid van de diesel door de flowsensor is bepaald zie Figuur 9.1: struct hcmIOData.




**Figuur 9.10**

**Functies**
Met de _InitFuelConsumption functie wordt een EM Fuel Consumption geïnitialiseerd. Als extra variabele worden meegegeven:

- ModuleNr: bevat het unieke id nummer van het EM Fuel Consumption object
- cmFlowSens* Inflow: bevat het adres van de CM Flowsensor die gebruikt gaat worden voor het meten van de dieselinvoer.
- cmFlowSens* Outflow: bevat het adres van de CM Flowsensor die gebruikt gaat worden voor het meten van de dieselafvoer. Is nul als er maar met een CM Flow Sensor gewerkt wordt.
- Mode: bevat een nul of een één en bepaald of de EM Fuel Consumption gebruikt gaat worden met één of twee CM Flowsensor.

De _ InitFuelConsumption roept ook de _ResetFuelConsumption functie aan. Deze zet de variabele fFuelConsumption en dTotalFuelConsumed op nul en reset hiermee het totaal dieselverbruik. Deze functie kan in de toekomst gebruikt worden. om vanaf de webapplicatie het dieselverbruik te resetten. Dit is door tijd gebrek nog niet geïmplementeerd.

Met de _ProcessFuelConsumption wordt het actuele dieselverbruik berekend en het totaal dieselverbruik bijgewerkt. Het dieselverbruik wordt bij twee flowsensoren berekend met de volgende formule:

$$actueel\ dieselverbruik(l/h) = dieselinvoer(l/h) - dieseluitvoer(l/h)$$

Bij één flowsensor

$$actueel\ dieselverbruik(l/h) = dieselinvoer(l/h)$$

Totaal dieselverbruik wordt berekend met de formule en wordt per 100ms uitgevoerd:

$$totaal\ dieselverbruik\ +=\ dieselverbruik\ in\ afgelopen\ 100ms$$

$$totaal\ dieselverbruik\ +=\ (actueel\ dieselverbruik)/(3600 * 10)$$

De EM Fuel Consumption heeft een _SendDataFuelConsumption fuctie. Deze functie heeft de zelfde functionaliteit als de SendDataFlowSens. Voor de functionaliteit van de SendDataFlowSens functie zie 9.2.1 CM Flow sensor

De _UpdateFuelConsumption functie roept de _ProcessFuelConsumption en de _SendData-FuelConsumption functie aan. _UpdateFuelConsumption wordt iedere programma cycli aangeroepen in het hoofdprogramma.

## 10 Webapplicatie

Met de webapplicatie wordt de data verzameld door het SMMS gevisualiseerd op een scherm en opgeslagen in een database. De webapplicatie wordt geschreven m.b.v.

- HTML: hiermee wordt de webpagina opgebouwd
- CSS: bepaald de stijl van de pagina (lettertype, kleur e.d.)
- Javascript: dit wordt gebruikt om diverse elementen (meters en grafieken) te creeëren . wordt gebruikt voor client side scripts.
- ASP.NET: is de code die op de server draai en zorgt er voor dat de server de juiste informatie naar de client toestuurt.
- Voor de communicatie met de database wordt SQL en C# gebruikt.
- Node JS: wordt gebruikt om de data die binnenkomt op de serieel poort door te sluizen naar de webpagina.

De webapplicatie bestaat uit een webpagina waarop de data wordt weergegeven en een database waar de data wordt opgeslagen.

Doordat de afstudeerder nog nooit met HTML, CSS, ASP.NET en Javascript gewerkt heeft zal de webpagina geen technische hoogstandje worden. De prioriteit is vooral de webapplicatie functioneel werkend te krijgen. Daarna zal het visuele gedeelte (meters en grafieken) ingebouwd worden. Pas dan wordt er gekeken naar eventuele optimalisatie. De webpagina zal ontwikkeld worden met behulp van het programma webmatrix. Dit is een programma waarmee beginnende ontwikkelaars een webpagina kunnen bouwen.

Er is voor gekozen om alleen javascrip librarys te gebruiken die offline beschikbaar gemaakt kunnen worden. Hier is voor gekozen omdat er geen vaste internet verbinding op een schip aanwezig is. De schipper moet voor iedere Mb dataverbruik betalen.

Door tijdgebrek is de webapplicatie nog niet voltooid. De volgende onderdelen zijn voltooid:

- De database (met hierin tabellen voor iedere CM en EM datastruct, in de tabbellen zijn kolommen gedefinieerd voor de variabelen in de datastructs)
- Lezen van een serieele poort op de pc/laptop. *(ITP Physical Computing)*
- Doorsluizen van de data naar de webpagina *(ITP Physical Computing)*
- Weergeven van de data (als tekst) en gebruiken van de data op de webpagina
- Verbinding maken met database en data ophalen van de database *(joeriks)*

Nog niet voltooid:

- Opslaan van data in de database
- Data weergeven m.b.v. meters en grafieken

## 10.1 Uitlezen van de serieele poort

De serieele poort wordt uitgelezen m.b.v. een node JS programma. Node JS is een uitbreiding variant van javascript. Met een node JS kunnen applicatie onderling verbonden worden.

De Node JS bestuurt com poort 1 van de PC/laptop. Het node JS programma opent de com poort en luistert naar een binnenkomend bericht. Daarnaast opent het node JS programma een websocket. De node JS wordt via een websocket verbonden met de Webpagina Als de syelcontroller de datastring met afsluitende \n zend naar de PC detecteerd het node JS programma dit en roept een functie aan om de string via de websocket naar de webpagina te zenden.

## 10.2 Weergeven van de data

Zoals eerder beschreven is het visuele gedeelte van de webpagina nog niet af. Het is wel mogelijk om de string gestuurd door het node JS programma via de websocket weer te geven in de webapplicatie. Hiervoor is een HTML object aangemaakt waarin m.b.v. javascript de ontvangen string wordt weergegeven.

## 10.3 Lezen uit de database

Met een ander javascript kan m.b.v. een C# asp.net script een data uit een database gelezen worden. het javascript roept met behulp van een GET functie het asp.net script aan. Het asp.net script draait op de server en zoek in de database verbonden aan de server naar de opgevraagde data. De data wordt met een response functie in de vorm van JSON terug gestuurd naar het javascript. Als het bericht binnenkomt wordt de JSON data in een javascript object gezet en kan de data gebruikt worden.

**JSON**

JSON staat voor JavaScript Object Notation. JSON is de structuur waarmee javascriptobjecten worden opgebouwd. De sytax:

```
Var javascriptObjectarray = [{ObjectVarId : ObjectVarVal},
{ObjectVarId : ObjectVarVal }]
```

Objecten worden geplaatst tussen {} en arrays tussen [ ]. Objecten hebben één of meerdere variabele id's met daaraan een waarde gekoppeld. Een manier om de tweeden ObjectVarVal te benaderen is

```
Var Val = javascriptObjectarray[1]. ObjectVarId
```

## 10.4 Schrijven naar data base en visualiseren

Voor het schrijven naar de database zal een functie worden aangeroepen die een string kan sturen naar een asp.net c# script. Het script zal de string die alle actuele data van het SMMS bevat in JSON omzetten naar een c# variabele en dan alle data plaatsen op de juiste plaats in de database.

Het visualiseren zal met behulp van een javascrip libary gaan die JSON data kan visualiseren in een grafiek. Er is hiervoor al een library gevonden die als demo versie gebruikt kan worden.[1]

[1] http://www.jqwidgets.com/

# 11 Eind test

In de eindtest in de laatste week van de stage is het systeem dat tot dan toe af was getest.

Er wordt gewerkt met drie syel controllers. Hiervan werken er twee werken als slave controllers. De andere werkt als centrale controller die alle data verzameld van de slave controllers en via een compoort doorstuurt naar de pc. Op de pc draait de gedeeltelijk voltooide webaplicatie die de data in tekstvorm zal printen op het scherm.

Op slave controller A zijn drie analoog signaal generatoren aangesloten. Zij simuleren twee flowsensoren die het dieselverbuik van een motot meten. De derde analoge ingang simuleert een druksensor die de oliedruk meet

Op de tweede controller zijn vier oliekwaliteit sensor aangesloten via een modbus. Daarnaast is er een cabus aangesloten op de controller. Een testaplicatie op de pc genereerd met een usb to can kabel canberichten met j1939 die herkend wordt door de CMJ1939.

De twee slave controllers sturen hun data via de ethernet verbinding naar de centrale controller.

**Te testen functionaliteit**

- Bewegen de waardes geprint in de webapplicatie mee als de waardes van de sensoren of J1939 veranderd?

**Resultaten**

| Sensor | Waarde beweegt mee in webapp | Geslaagd? |
|---|---|---|
| Flowmeter 1 | Ja binnen 1 seconde | V |
| Flowmeter 2 | Ja binnen 1 seconde | V |
| Berekend brandstofverbruik | Ja binnen 1 seconde | V |
| Druksensor | Ja binnen 1 seconde | V |
| Oliekwaliteitssensor 1 | Ja binnen 1 seconde | V |
| Oliekwaliteitssensor 2 | Ja binnen 1 seconde | V |
| Oliekwaliteitssensor 3 | Ja binnen 1 seconde | V |
| Oliekwaliteitssensor 4 | Ja binnen 1 seconde | V |
| SPN data RPM | Ja binnen 1 seconde | V |
| SPN data Oliedruk | Ja binnen 1 seconde | V |
| SPN data temperatuur ambiant | Ja binnen 1 seconde | V |
| SPN data temperatuur diesel | Ja binnen 1 seconde | V |

Tabel 11.1: Overzicht functie testen en resultaat

# 12 Conclusie en aanbevelingen

Doel van de afstudeeropdracht was het schrijven van de software voor de controllers. Daarnaast moest er een webapplicatie ontwikkeld worden. Kijkend naar de op te leveren producten en de must punten in het Moscow zie Figuur 5.2: Moscow Model op bladzijde 6. Kunnen we concluderen dat de afstudeeropdracht grotendeels geslaagd is.

De volgende onderdelen zijn ontwikkeld

- HCM voor het managen van de hardware op de controllers
- CM voor het uitlezen van motordata en het uitlezen van de flowsensoren
- EM voor het berekenen van het brandstofverbruik

Daarnaast is er is er ook een CM ontwikkeld voor het meten van de oliekwaliteit, olie temperatuur en druk. Doordat dit meer tijd in nam dan vooraf was voorzien heeft het project hierdoor wel enige vertraging opgelopen.

Daarna is begonnen met het ontwikkelen van de webapplicatie. De volgende onderdelen van de webapplicatie zijn ontwikkeld:

- Lezen van data op een com poort
- Data gelezen op de com poort doorsluizen naar de website
- Opzetten van een database

Deze modules zijn met succes getest in een eindtest in de laatste week van de afstudeerstage. Het enige onderdeel in het must gedeelte van het MOSCOW model dat niet ontwikkeld is:

- Opslaan van data in de database

Door de tijdsdruk aan het einde van het project is niet meer gelukt deze functie volledig te implementeren Dit komt vooral doordat het ontwikkelen van de oliekwaliteit modules enige vertraging was opgelopen.

Voor de verdediging worden de volgende functionaliteit geïmplementeerd:

- Opslaan van data in de database

Voor de verdediging hoop ik de volgende functionaliteit te hebben geïmplementeerd:

- Volledig functionerende webapplicatie

## 13 Bibliografie


*Cyclic redundancy check* . (sd). Opgehaald van Wikipedia: https://nl.wikipedia.org/wiki/Cyclic_redundancy_check

Groen, J. (2016). *VPN onderzoek.* Hendrik Ido Ambacht: VMB.

Groen, J. (2016). *Wireless Communication onderzoek.* Hendrik Ido Ambacht: VMB.

*ITP Physical Computing*. (sd). Opgehaald van Lab: Serial Communication with Node.js: https://itp.nyu.edu/physcomp/labs/labs-serial-communication/lab-serial-communication-with-node-js/

*joeriks*. (sd). Opgehaald van coding in the cloud: https://joeriks.com/2011/01/30/5-min-webmatrix-jquery-ui-json-jquery-templates/

Joris Groen, E. G. (2016). *vooranalyze SMMS.* -: -.

*MoSCoW method*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/MoSCoW_method

Reference manual ANSI-C (Helpfiles Proteus ES). (sd).

*Spiral model*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Spiral_model

*TCP Connection Termination* . (sd). Opgehaald van The tcp/ip guide: http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm

*tutorial HTML, Javascrip, SQL, JSON en jQuery*. (sd). Opgehaald van W3Scvhools: http://www.w3schools.com/

*VSB (VMB standaardisatie Boek).* (2016). Hendrik Ido Ambacht: VMB.

# 14 Bijlagen

## 14.1 Bijlage voorbeeld berekening Puls vs. Analoog dieselverbruik

# 1. Onderzoek meetfout

In dit voorbeeld zal gedemonstreerd worden hoe een meetfout kan optreden bij het berekenen van het totaal dieselverbruik. De fout gedemonstreerd in dit voorbeeld is alleen van toepassing bij het meten van de stroomsnelheid m.b.v. een analoog signaal 4-20ma.

De meetfout zal aangetoond worden m.b.v. een tabel en grafieken. De getallen in de situatie gelden puur als voorbeeld.

**Situatie**

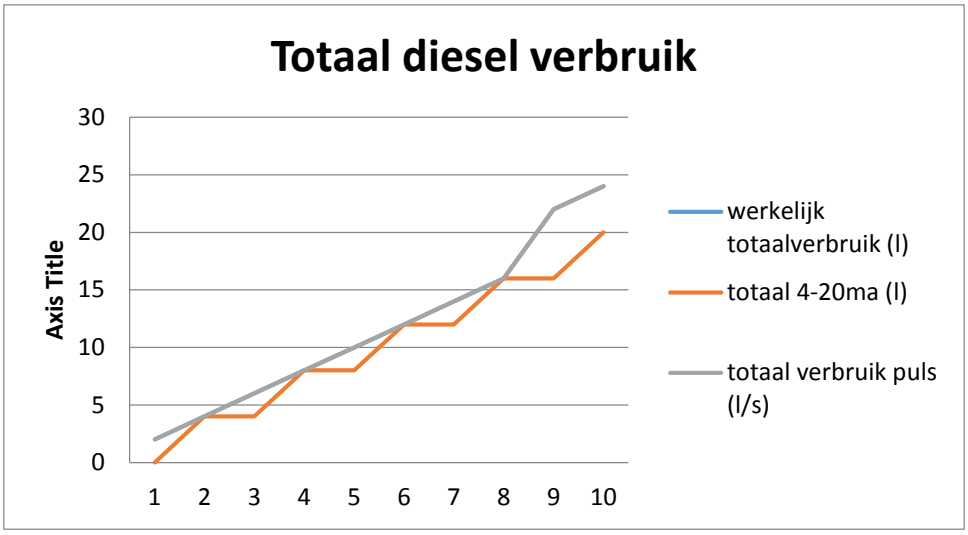
Een flowsensoren meet het dieselverbruik van de een motor. De flowsensor verzend een puls signaal. Eén puls vertegenwoordig 1 liter diesel  Iedere puls wordt door de controller geteld. Daarnaast verstuur de meter een analoog signaal die om de twintig  seconde gemeten wordt door de controller.

| tijd (s) | werkelijk verbruik (l/s) | Werkelijk totaalverbruik (l) | verbruik gemeten 4-20ma (l/20s) | totaal 4-20ma (l) | verbruik puls (l/s) | totaal verbruik puls (l/s) |
|---|---|---|---|---|---|---|
| 10 | 2 | 2 | NULL | 0 | 2 | 2 |
| 20 | 2 | 4 | 4 | 4 | 2 | 4 |
| 30 | 2 | 6 | NULL | 4 | 2 | 6 |
| 40 | 2 | 8 | 4 | 8 | 2 | 8 |
| 50 | 2 | 10 | NULL | 8 | 2 | 10 |
| 60 | 2 | 12 | 4 | 12 | 2 | 12 |
| 70 | 2 | 14 | NULL | 12 | 2 | 14 |
| 80 | 2 | 16 | 4 | 16 | 2 | 16 |
| 90 | 6 | 22 | NULL | 16 | 6 | 22 |
| 100 | 2 | 24 | 4 | 20 | 2 | 24 |

Zoals te zien is in de tabel mist de controller bij het analoge signaal de verandering in de stoomsnelheid die plaatst vind op seconde 90. Als de controller het analoge signaal sneller (vaker) zou uitlezen zou deze verandering wel meegenomen worden in de berekening van het totale dieselverbruik.

**Conclusie**

Met behulp van het puls signaal kan betrouwbaarder het totale dieselverbruik gemeten worden. In dit voorbeeld is het effect uitvergroot. De controller leest in werkelijkheid om de 100ms het signaal in. Met de snellere meetmoment wordt de kans klein dat een verhoging van de stroomsnelheid niet gemeten wort. Dit maakt het effect van de meetfout bij het meten van de stroomsnelheid m.b.v. een analoog signaal te verwaarlozen.

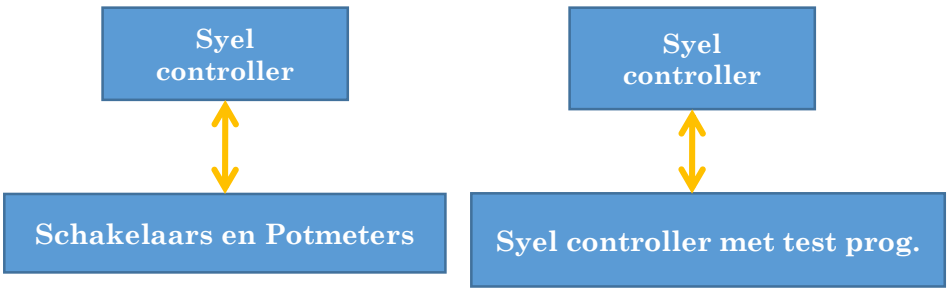

Totaal diesel verbruik
Axis Title
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
werkelijk totaalverbruik (l)
totaal 4-20ma (l)
totaal verbruik puls (l/s)

## 14.2 Bijlage Test rapporten HCM, CM en EM

# 1 Testrapport HCM IO

Dit is het testrapport voor de HCM IO. In deze testen werd de functionaliteit van de HCM IO getest.

**Testopstelling**




In de opstelling wordt één V4L syelcontroller gebruikt waarop schakelaars op de DI, potmeters op de AI is aangesloten. Voor het meten van de DO en de AO wordt een universeel meter gebruikt.

De signalen voor de AI en DI, gemeten door de syel controller waarop de HCM IO draait, worden weergegeven op het scherm van de syel controller. De waarde van de DI komt op de DO te staan en de waarde van de eerste twee AI op de twee AO.

Een tweede opstelling met een syel controller wordt gebruikt voor het testen van de risingedge counter van de HCM. Het testprogramma op de syel controller kan tot 2000 pulsen per seconde genereren.

**Te testen functionaliteit:**

- Analoog signaal wordt gemeten op de syel controller
- Digitaal signaal wordt gemeten op de syel controller
- Analoog uitgangsignaal wordt gegenereerd door de syel controller
- Digitaal uitgangsignaal wordt gegenereerd door de syel controller.
- Testen rising edge counter

**Te meten**

- Maximaal te meten aantal pulsen/seconde

**Resultaten**

Uit de test bleek dat de HCM IO alle functionaliteit bezat. De HCM IO detecteerde welke DI hoog af laag waren en zette op basis daarvan DO hoog of laag. Het signaal op de AI werd ingelezen en bewoog mee met de waarde van het AI signaal. Bij 0 mA was de waarde 0, bij 20 mA was de waarde tussen de 14870 en de 14890 (20*744).

Op de analoge uitgangen wordt de zelfde waarde gemeten als op de eerste twee analoge ingangen. De analoge uitgangen doen het dus.

Bij het testen van de rising edge konden maximaal 1800 pulzen per seconde geteld worden.

# 1 Testrapport HCM RSModbus en HCM Serieel

Dit is het testrapport van de HCM RSModbus.

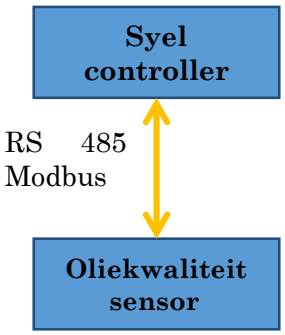
**Te testen functionaliteit:**

- Verbinding maken met een andere sensor
- Verbinding maken met een pc waarop een modbus slave applicatie draait
- Register lezen van modbus slave applicatie op pc en van sensor
- Register schrijven van modbus slave applicatie op pc en van sensor

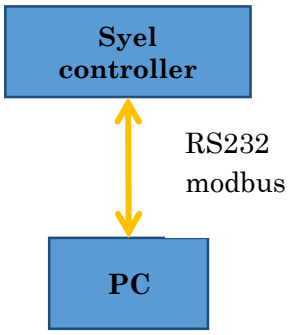
**Meetopstelling**

Voor de meetopstelling zijn twee opstellingen gebruikt. Een voor het testen van de modbus communicatie tussen een syel controller en een oliekwaliteitssensor. De tweede opstelling is voor het testen van modbus communicatie tussen een syel controller en een pc.

Opstelling




Opstelling




**Resultaten**

Bij het testen van de modbus communicatie werkte de communicatie met de modbusslave applicatie op de pc direct. Het opzetten van modbus communicatie met de oliekwaliteitsensor koste wat meer moeite. Het bleek dat en pariteit en stopbit instellingen van de compoort van de syel controller verkeerd stonden. Nadat deze instellingen goed waren ingesteld kon er een modbus communicatie opgezet worden. Bij de opstelling met de sensor werd het lezen en schrijven van registers getest. Dit bleek te werken. Ook bij de modbusslave applicatie konden registers door de syel controller uitgelezen en beschreven worden.

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Te testen functionaliteit: | | V |
| Verbinding maken met een andere sensor | - | V |
| Verbinding maken met een pc waarop een modbus slave applicatie draait | - | V |
| Register lezen van modbus slave applicatie op pc en van sensor | - | V |
| Register schrijven van modbus slave applicatie op pc en van sensor | - | V |

# 2 Testrapport HCM TCP/IP

Dit is het testrapport van de HCM TCP/IP.

**Te testen functionaliteit:**

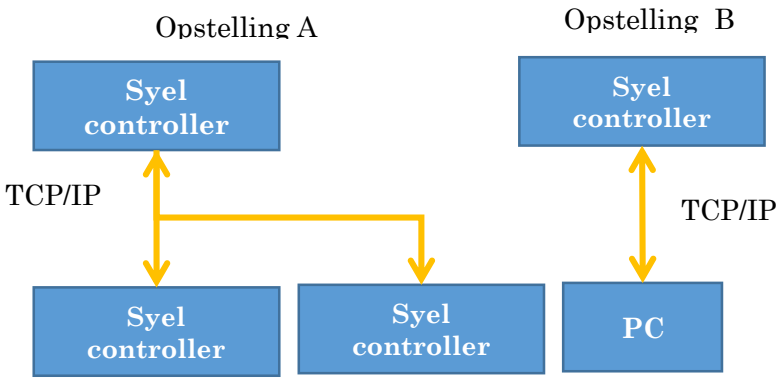
- Verbinding maken met een andere controller
- Verbinding maken met een pc werkend
- Meerdere Berichten versturen naar PC en syel controller
- Meerdere Berichten ontvangen van PC en syel controller

**Te meten**

- Maximale bericht grote HCM TCP/IP voor de syelcontroller
- Maximale verbinding snelheid syel controller

**Meetopstelling**

Voor de meetopstelling zijn twee opstellingen gebruikt. Een voor het testen van de communicatie tussen een syel controller werkend als server en twee syel controllers werkend als client. De tweede opstelling is voor het testen van communicatie tussen een syel controller en een pc.




**Resultaten**

Bij beide opstellingen lukte het om een tcp/ip verbinding op te bouwen. Ook het versturen van berichten ging goed. Ieder tcp/ip bericht ontvangen door de syel controller wordt geplaatst in de ontvangst buffer en ieder bericht verzonden door de syel controller wordt geplaatst in de zend buffer. Als er meerdere berichten in één programma cyli binnen komen op bij de syel controller worden deze FIFO geplaatst in de ontvangstbuffer waarbij het eerste ontvangen bericht op de eerste plaats in de ontvangstbuffer staat. Als er meerdere berichten verzonden worden in één programma cycli worden die FIFO in de buffer geplaatst en FIFO verzonden.

Bij het stapsgewijs verhogen van de bericht grote bleek dat de syel controller die de berichten ontvangt berichten groter dan 200 bytes niet meer volledig ontving. Dit betekent dat voor communicatie tussen twee syel controllers dat berichten niet groter mogen zijn dan 200 byte.

Bij het stapsgewijs verhogen van de zendsnelheid bleek dat het de ontvangende syel controller berichte miste als er meer dan 40 berichten per seconde werden verzonden. Voor deze test werden door de zendende syel controller 200 berichten verzonden. Deze berichten moesten allemaal door de ontvangende controller worden ontvangen. Bij een zendsnelheid groter dan 40 berichten/ sec kwamen twee berichten niet aan bij de ontvangende controller. Voor betrouwbare gegevens overdracht moet na het verzenden van een bericht 25ms gewacht worden voordat een tweede bericht verzonden mag worden. bij het verzenden van berichten naar een PC bleek dat er tot 2000 bytes verzonden konden worden.

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| Verbinding maken met een andere controller | - | V |
| Verbinding maken met een pc werkend | - | V |
| Meerdere Berichten versturen versturen naar PC en syel controller | - | V |
| Meerdere Berichten ontvangen van PC en syel controller | - | V |
| Te meten | | V |
| Maximale bericht grote HCM TCP/IP voor de syel controller | 200 byte | V |
| Maximale verbinding snelheid syel controller | 40 berichten/sec (800byte/sec) | V |

# 3 Testrapport HCM CAN

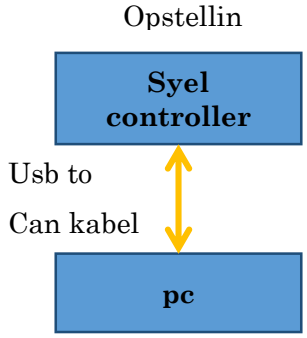
Dit is het testrapport van de HCM CAN.

**Te testen functionaliteit:**

- Verbinding maken met een pc waarop een CAN applicatie draait
- Berichten ontvangen van CAN applicatie op PC
- Berichten schrijven naar CAN applicatie op PC

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. De testopstelling bestaat uit een PC verbonden met een Syel controller d.m.v. een USB to can kabel. De can aansluiting van de kabel is aangesloten op de syel controller en de usb aansluiting op de pc. Met een Can applicatie kan een can bericht verstuurd worden naar of ontvangen worden van een syel controller.




**Resultaten**

Bij het testen van de HCM CAN bleek dat er berichten konden worden verstuurd naar en ontvangen van de can applicatie op de pc. dit is getest bij meerdere Baudrates van de HCM CAN en de can slave applicatie op de pc.

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| Verbinding maken met een pc waarop een CAN applicatie draait | - | V |
| Berichten ontvangen van CAN applicatie op PC | - | V |
| Berichten schrijven naar CAN applicatie op PC | - | V |

# 4 Testrapport CM Press

Dit is het testrapport van de CM PRESS.

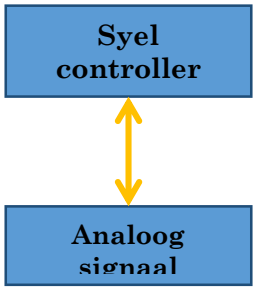
**Te testen functionaliteit:**

- Meten van de druk bij een veranderend analoog signaal
- Error bij signaal < 4mA
- 4 bar bij 0mA
- 25 bar bij 20 mA

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. De testopstelling bestaat uit een analoog signaal generator (4-20ma) aangesloten op een AI van de Syel controller.

Opstellin




De waarde in bar uitgelezen door de sensor en berekend door de CM Press wordt geprint op het syel scherm.

**Resultaten**

| resultaten | Opmerking | Geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| Meten van de druk bij een veranderend analoog signaal | Waarde veranderd mee in het scherm. | V |
| Error bij signaal < 4mA | Error variabele wordt op 1 gezet bij analoog signaal < 4ma | V |
| 0 bar bij 4mA | - | V |
| 25 bar bij 20 mA | - | V |

# 5 Testrapport CM OilQua

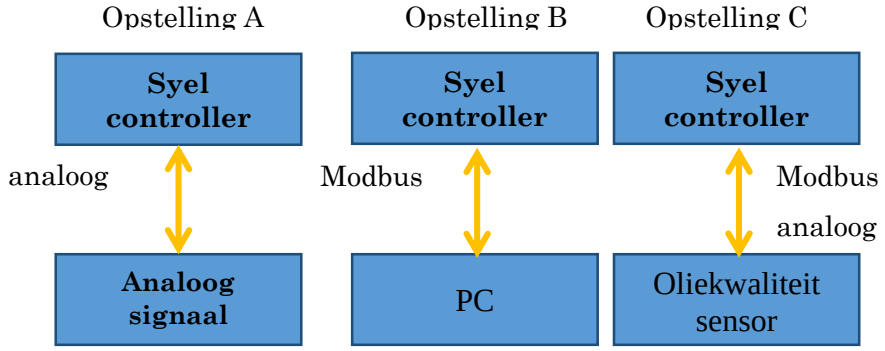
Dit is het testrapport van de CM OilQua.

**Te testen functionaliteit:**

- Verbinding met echte oliekwaliteitssensor
- Error bij signaal < 4mA
- Temperatuur -30°C @4mA, 130°C @20mA
- Oliekwaliteit -20% @4mA, 60% @20mA
- Oliekwaliteit en temperatuur zijn gelijk aan register waarde van modbus slave * 100.

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. Testopstelling A en B testen de CM OilQua respectievelijk m.b.v. een analoog signaal en een modbus slave applicatie op een pc. Testopstelling C bestaat uit een oliekwaliteitssensor aangesloten op een AI en modbus van de Syel controller.




De waarde in bar uitgelezen door de sensor en berekend door de CM Press wordt geprint op het syel scherm.

**Resultaten**

Bij beide opstellingen werkte de CM OilQua. Bij opstelling C werd vooral gekeken of er een modbus verbinding kon worden opgezet tussen de syel controller en de oliekwaliteitssensor. Bij opstelling A en B werd vooral gekeken of de op het scherm geprinte waarde van de oliekwaliteit, temperatuur en omgevingstemperatuur overeenkwamen met de aangeboden waarden van het analoog signaal en de modbus slave applicatie.

| resultaten | Opmerking | Geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| Verbinding met echte oliekwaliteitssensor | Er wordt verbinding gemaakt en er worden gegevens doorgeven door de sensor | V |
| Error bij signaal < 4mA | - | V |
| Temperatuur -30°C @4mA, 130°C @20mA | - | V |
| Oliekwaliteit -20% @4mA, 60% @20mA | - | V |
| Oliekwaliteit en temperatuur zijn gelijk aan register waarde van modbus slave * 100. | - | V |

# 6 Testrapport CM Flow Sensor

Dit is het testrapport van de CM Flow Sensor.

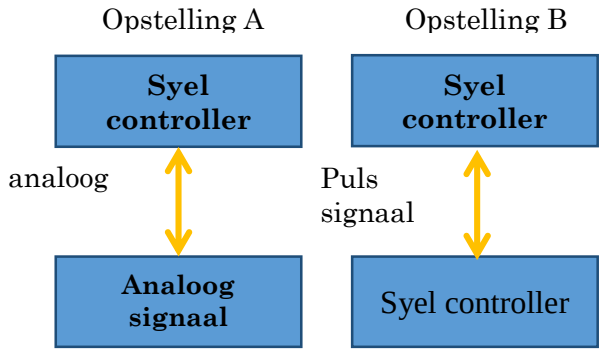
**Te testen functionaliteit:**

- flowsensor geeft juiste flow weer bij analoog signaal
- error bij signaal < 4 ma
- flowsensor geeft flow weer bij puls signaal

**Meetopstelling**

De CM Flow Sensor is ingesteld met offset van 0 en vermenigvuldigingsfactor van 2. Dit betekent dat bij een analoog signaal van 6 er 4l/h door de flowmeter stroomt. Als er 1 puls (is 1l) per 100ms gemeten worden dat er 72000 l/h door de flowmeter stroomt.

Er worden twee meetopstellingen gebruikt. Opstelling a test de analoge mode van CM Flowsensor en opstelling B de puls ingang mode van de CM Flow Sensor.




**Resultaten**

Bij beide opstellingen werkte de CM Flow Sensor.

| resultaten | Opmerking | Geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| flowsensor geeft juiste flow weer bij analoog signaal | Bij een veranderd analoog signaal beweegt de berekende waarde mee. | V |
| Error bij signaal < 4mA | - | V |
| flowsensor geeft flow weer bij puls signaal | Bij een veranderd aantal pulsen/100ms beweegt de berekende waarde mee. | V |

# 7 Testrapport CM J1939

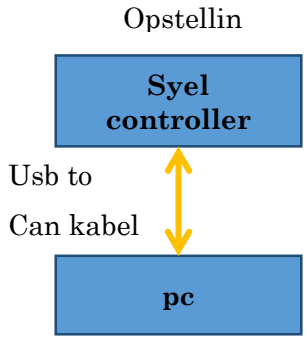
Dit is het testrapport van de CM J1939

**Te testen functionaliteit:**

- Verbinding maken met een pc waarop een CAN applicatie draait
- Berichten ontvangen van CAN applicatie op PC in J1939 formaat
- J1939 berichten uitpakken en waardes zetten in juiste SPN
- SPN die niet zijn ingevoerd worden genegeerd
- J1939 berichten versturen

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. De testopstelling bestaat uit een PC verbonden met een Syel controller d.m.v. een USB to can kabel. De can aansluiting van de kabel is aangesloten op de syel controller en de usb aansluiting op de pc. Met een can applicatie kan een can bericht in J1939 formaat verstuurd worden naar of ontvangen worden van een syel controller.




**Resultaten**

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| Verbinding maken met een pc waarop een CAN applicatie draait | - | V |
| Berichten ontvangen van CAN applicatie op PC in J1939 formaat | - | V |
| J1939 berichten uitpakken en waardes zetten in juiste SPN | - | V |
| SPN die niet zijn ingevoerd worden genegeerd | - | V |
| J1939 berichten versturen | - | V |

# 8 Testrapport CM Intern Com Send

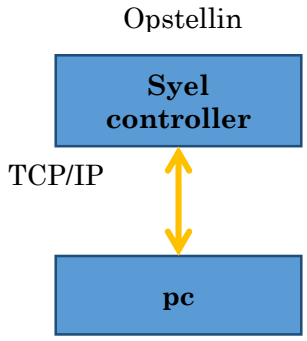
Dit is het testrapport van de CM Intern Com Send

**Te testen functionaliteit:**

- Verbinding maken met een pc met TCP/IP applicatie
- Start Bericht ontvangen van TCP/IP applicatie op PC
- Alle datastructs van de CM en EM worden in het juiste format verzonden.
- Verstuurt stopbericht op afloop

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. De testopstelling bestaat uit een PC verbonden met een syel controller d.m.v. een Ethernet. Met een TCP/IP applicatie kan een bericht verstuurd worden naar of ontvangen worden van een syel controller.




**Resultaten**

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Verbinding maken met een pc met TCP/IP applicatie | | |
| Start Bericht ontvangen van TCP/IP applicatie op PC | - | V |
| Alle datastructs van de CM en EM worden in het juiste format verzonden. | Alle structs worden verzonden. De eerste drie variabele van de berichten corresponderen met de structs die zij bevatten | V |
| Verstuurt stopbericht op afloop | - | V |

# 9 Testrapport CM Intern Com Rec

Dit is het testrapport van de CM J1939

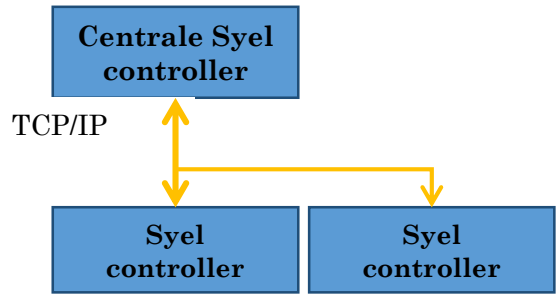
**Te testen functionaliteit:**

- Verbinding maken met slave controller A en B
- Start Bericht versturen naar Slave controller A
- Alle datastructs van de CM en EM worden in het juiste format ontvangen van slave controller A
- Ontvangt stopbericht op afloop van slave controller A
- Verstuurt nieuw start bericht naar volgende slave controller (B)

**Meetopstelling**

Voor de meting is één testopstellingen gebruikt. De testopstelling bestaat uit een twee slave controllers verbonden aan een centrale controller via een de ethernet. De centrale controller zal bij beide slave controllers de data opvragen

Opstellin




**Resultaten**

| resultaten | Opmerking | geslaagd |
|---|---|---|
| Verbinding maken met slave controller A en B | Met beide controller werd een tcp/ip verbinding opgebouwd | V |
| Start Bericht versturen naar Slave controller A | - | V |
| Alle datastructs van de CM en EM worden in het juiste format ontvangen van slave controller A | Alle structs worden ontvangen. De eerste drie variabele van de berichten corresponderen met de structs die zij bevatten | V |
| Ontvangt stopbericht op afloop van slave controller A | - | V |
| Verstuurt nieuw start bericht naar volgende slave controller (B) | - | V |

# 10 Testrapport CM Flow Sensor

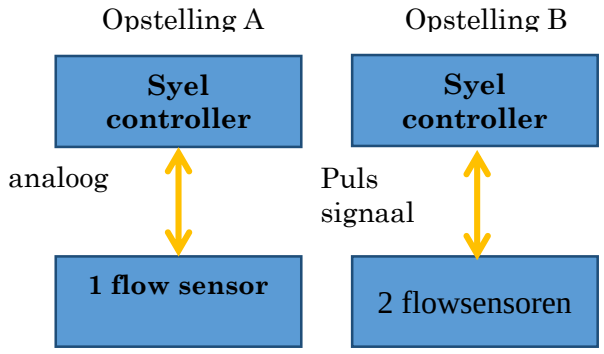
Dit is het testrapport van de CM OilQua.

**Te testen functionaliteit:**

- EM Fuel Consumption berekend juiste brandstof verbruik in beide modes
- EM Fuel Consumption berekend op tijd nieuw totaal brandstof verbruik in beide modes.

**Meetopstelling**

Er worden twee meetopstellingen gebruikt. Opstelling A test de EM Fuel Consumption met 1 flowsensor Opstelling B met twee flowsensor.




**Resultaten**

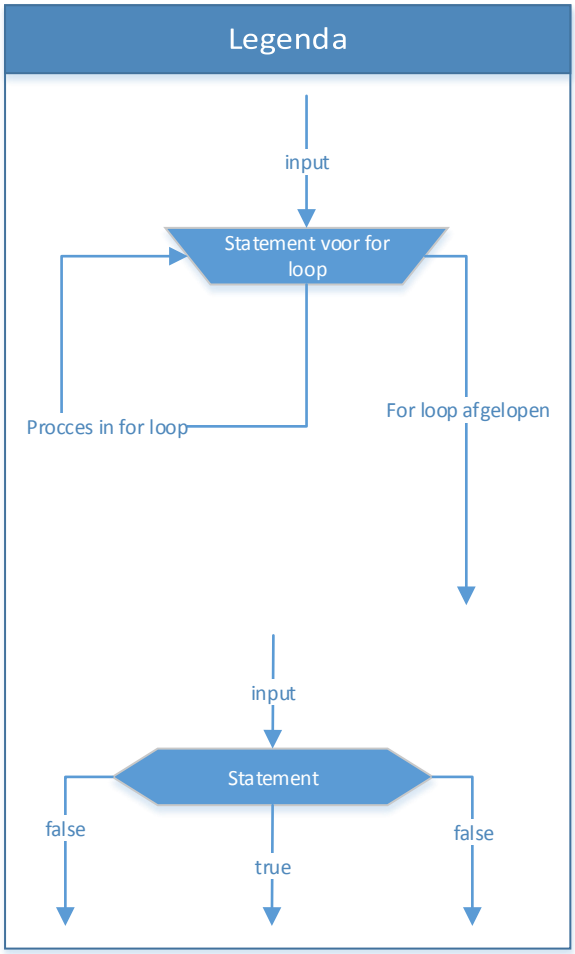
Bij beide opstellingen werkte de CM Flow Sensor.

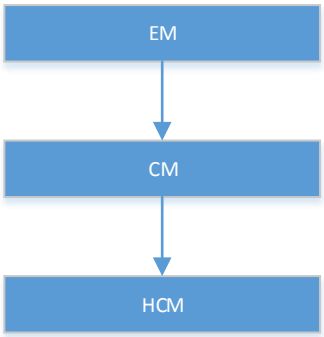
| Tijd | Gemeten actueel brandstof verbruik | | Berekend totaal brandstofverbruik | |
|---|---|---|---|---|
| | A | B | A | B |
| 100 ms | 2 l/100ms | 2 l/100ms | 2 l | 2 l |
| 200 ms | 2 l/100ms | 1 l/100ms | 4 l | 3 l |
| 300 ms | 4 l/100ms | 1 l/100ms | 8 l | 4 l |
| 400 ms | 4 l/100ms | 2 l/100ms | 12 l | 2 l |

| resultaten | Opmerking | Geslaagd |
|---|---|---|
| Te testen functionaliteit: | | |
| EM Fuel Consumption berekend juiste brandstof verbruik in beide modes | Bij beide opstellingen ging het berekenen van het actuele brandstofverbruik goed | V |
| Error bij signaal < 4mA | - | V |
| EM Fuel Consumption berekend op tijd nieuw totaal brandstof verbruik in beide modes. | Bij beide opstellingen ging het berekenen van het actuele brandstofverbruik goed | V |

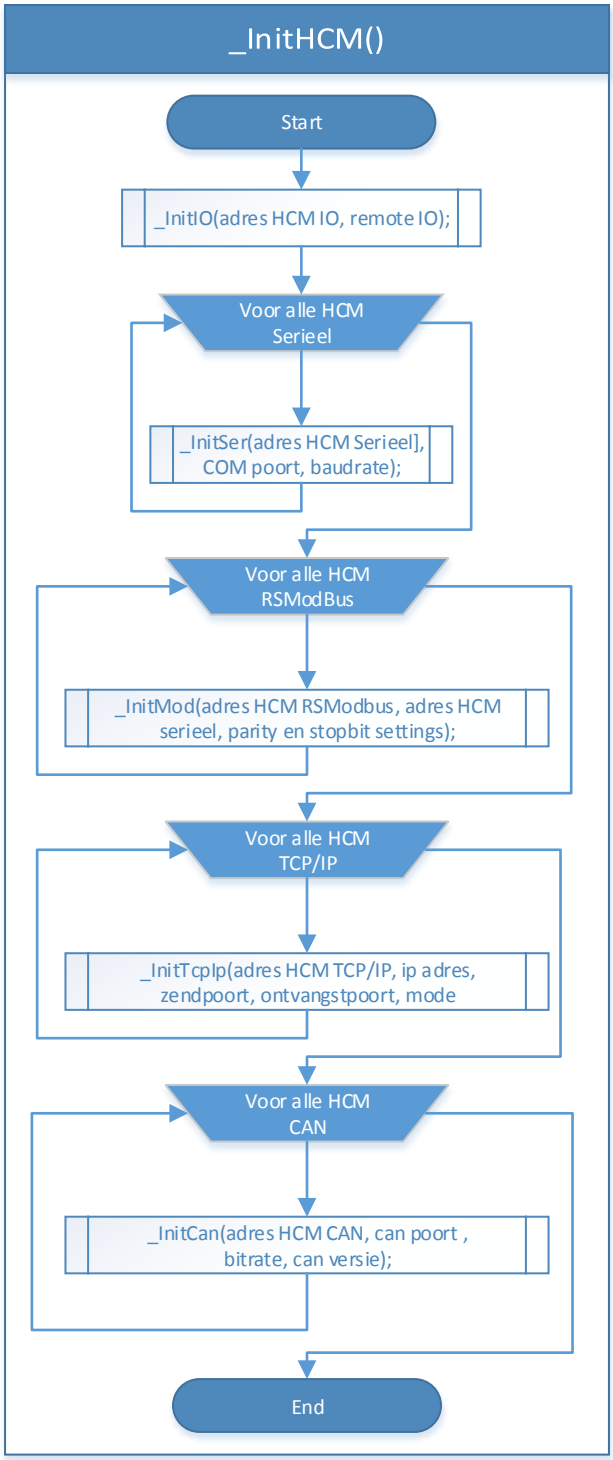
## 14.3 Bijlage Flowcharts HCM, CM en EM

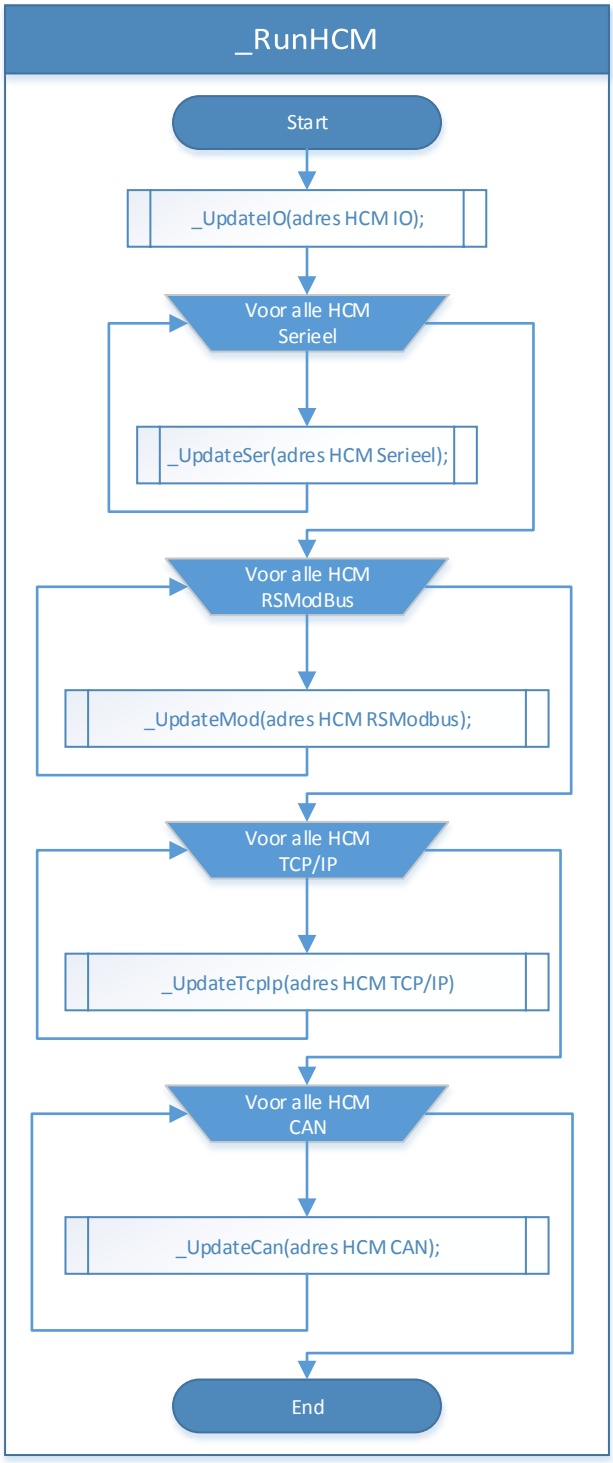
# 1 Software Units Flowcharts



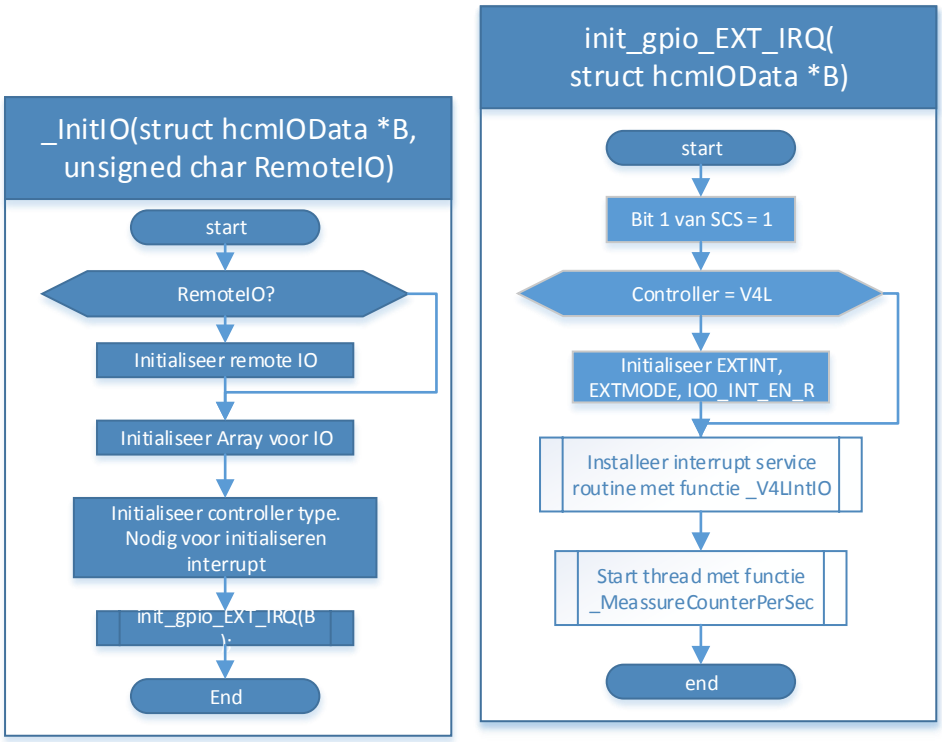



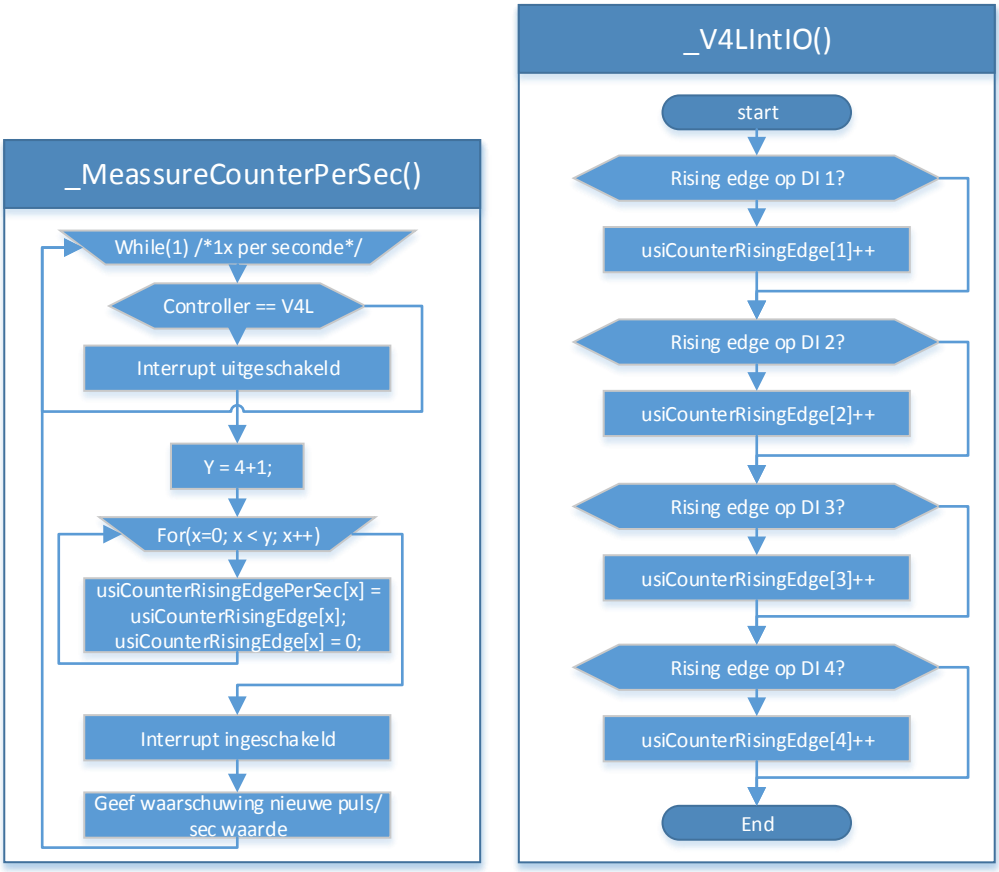
## 1.1 HCM

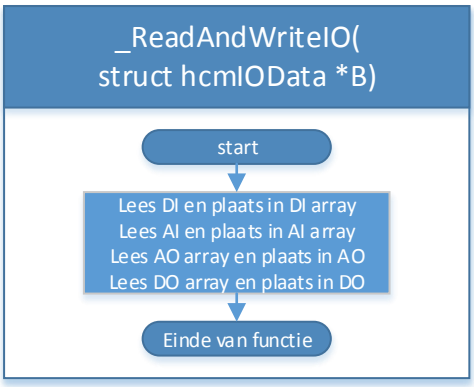

_InitHCM()
Start
_InitIO(adres HCM IO, remote IO);
Voor alle HCM Serieel
_InitSer(adres HCM Serieel], COM poort, baudrate);
Voor alle HCM RSModBus
_InitMod(adres HCM RSModbus, adres HCM serieel, parity en stopbit settings);
Voor alle HCM TCP/IP
_InitTcpIp(adres HCM TCP/IP, ip adres, zendpoort, ontvangstpoort, mode
Voor alle HCM CAN
_InitCan(adres HCM CAN, can poort , bitrate, can versie);
End

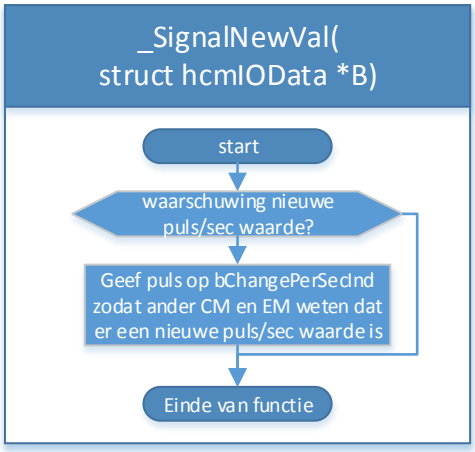

_RunHCM
Start
_UpdateIO(adres HCM IO);
Voor alle HCM Serieel
_UpdateSer(adres HCM Serieel);
Voor alle HCM RSModBus
_UpdateMod(adres HCM RSModbus);
Voor alle HCM TCP/IP
_UpdateTcpIp(adres HCM TCP/IP)
Voor alle HCM CAN
_UpdateCan(adres HCM CAN);
End

### 1.1.1 HCM IO

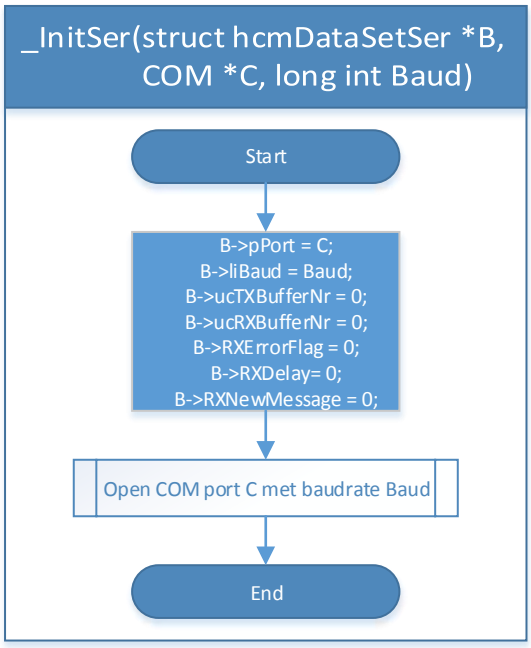

_InitIO(struct hcmIOData *B, unsigned char RemoteIO)
start
RemoteIO?
Initialiseer remote IO
Initialiseer Array voor IO
Initialiseer controller type. Nodig voor initialiseren interrupt
init_gpio_EXT_IRQ(B
End
init_gpio_EXT_IRQ( struct hcmIOData *B)
start
Bit 1 van SCS = 1
Controller = V4L
Initialiseer EXTINT, EXTMODE, IO0_INT_EN_R
Installeer interrupt service routine met functie _V4LIntIO
Start thread met functie _MeassureCounterPerSec
end

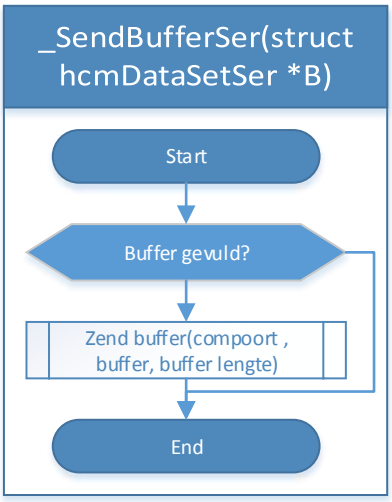

_MeassureCounterPerSec()
While(1) /*1x per seconde*/
Controller == V4L
Interrupt uitgeschakeld
Y = 4+1;
For(x=0; x < y; x++)
usiCounterRisingEdgePerSec[x] = usiCounterRisingEdge[x]; usiCounterRisingEdge[x] = 0;
Interrupt ingeschakeld
Geef waarschuwing nieuwe puls/ sec waarde
_V4LIntIO()
start
Rising edge op DI 1?
usiCounterRisingEdge[1]++
Rising edge op DI 2?
usiCounterRisingEdge[2]++
Rising edge op DI 3?
usiCounterRisingEdge[3]++
Rising edge op DI 4?
usiCounterRisingEdge[4]++
End

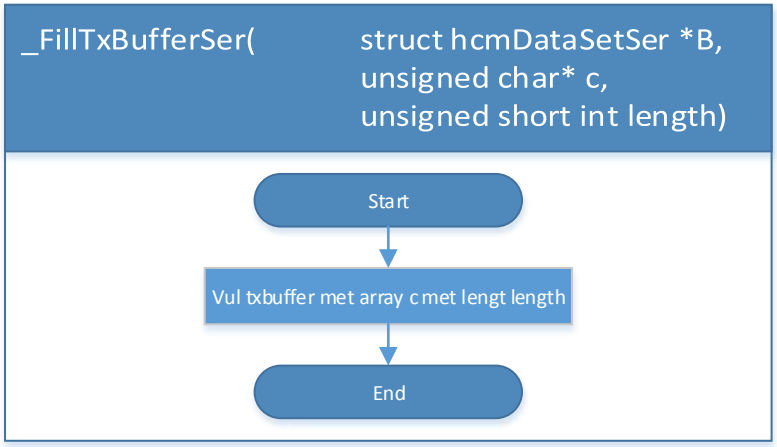

_ReadAndWriteIO(
struct hcmIOData *B)
start
Lees DI en plaats in DI array
Lees AI en plaats in AI array
Lees AO array en plaats in AO
Lees DO array en plaats in DO
Einde van functie

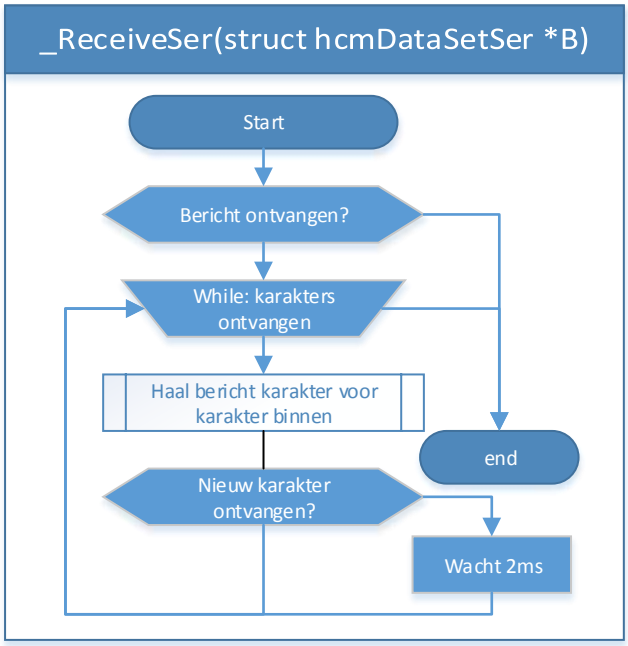

_SignalNewVal(
struct hcmIOData *B)
start
waarschuwing nieuwe puls/sec waarde?
Geef puls op bChangePerSecInd zodat ander CM en EM weten dat er een nieuwe puls/sec waarde is
Einde van functie

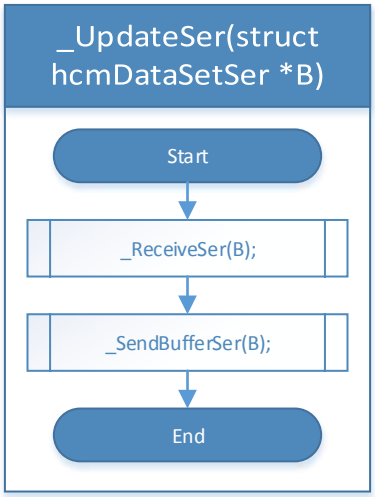
### 1.1.2 HCM serieel

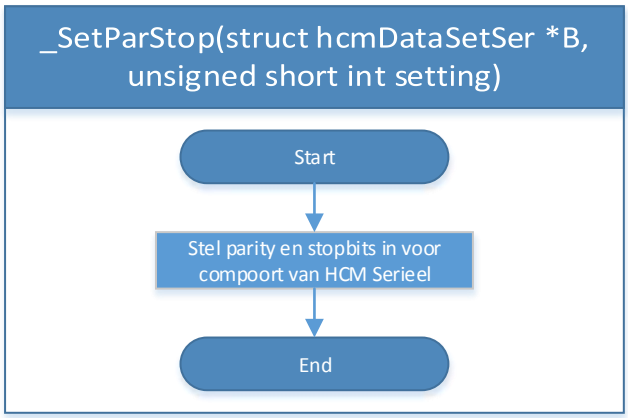


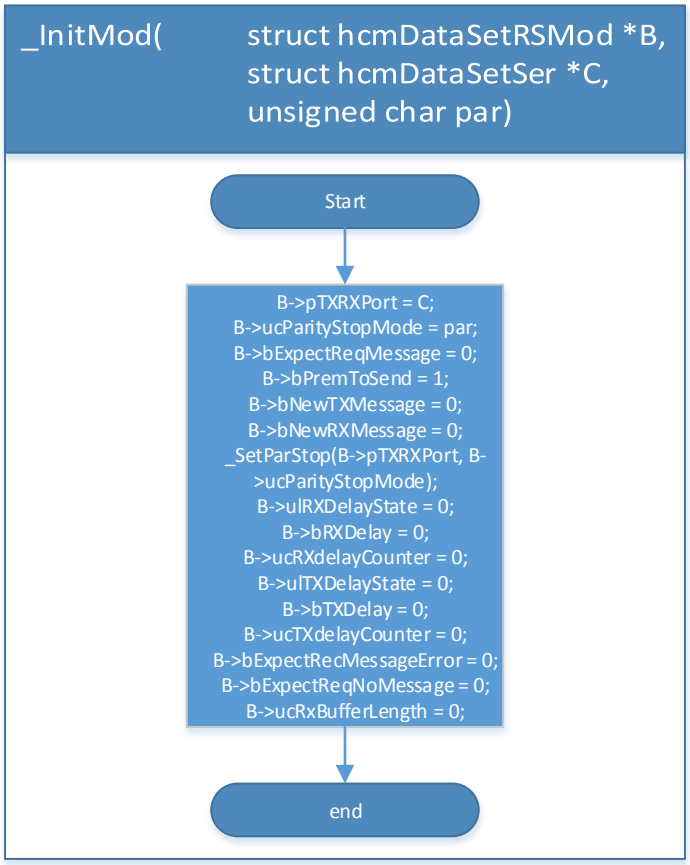




_ReceiveSer(struct hcmDataSetSer *B)
Start
Bericht ontvangen?
While: karakters ontvangen
Haal bericht karakter voor karakter binnen
end
Nieuw karakter ontvangen?
Wacht 2ms

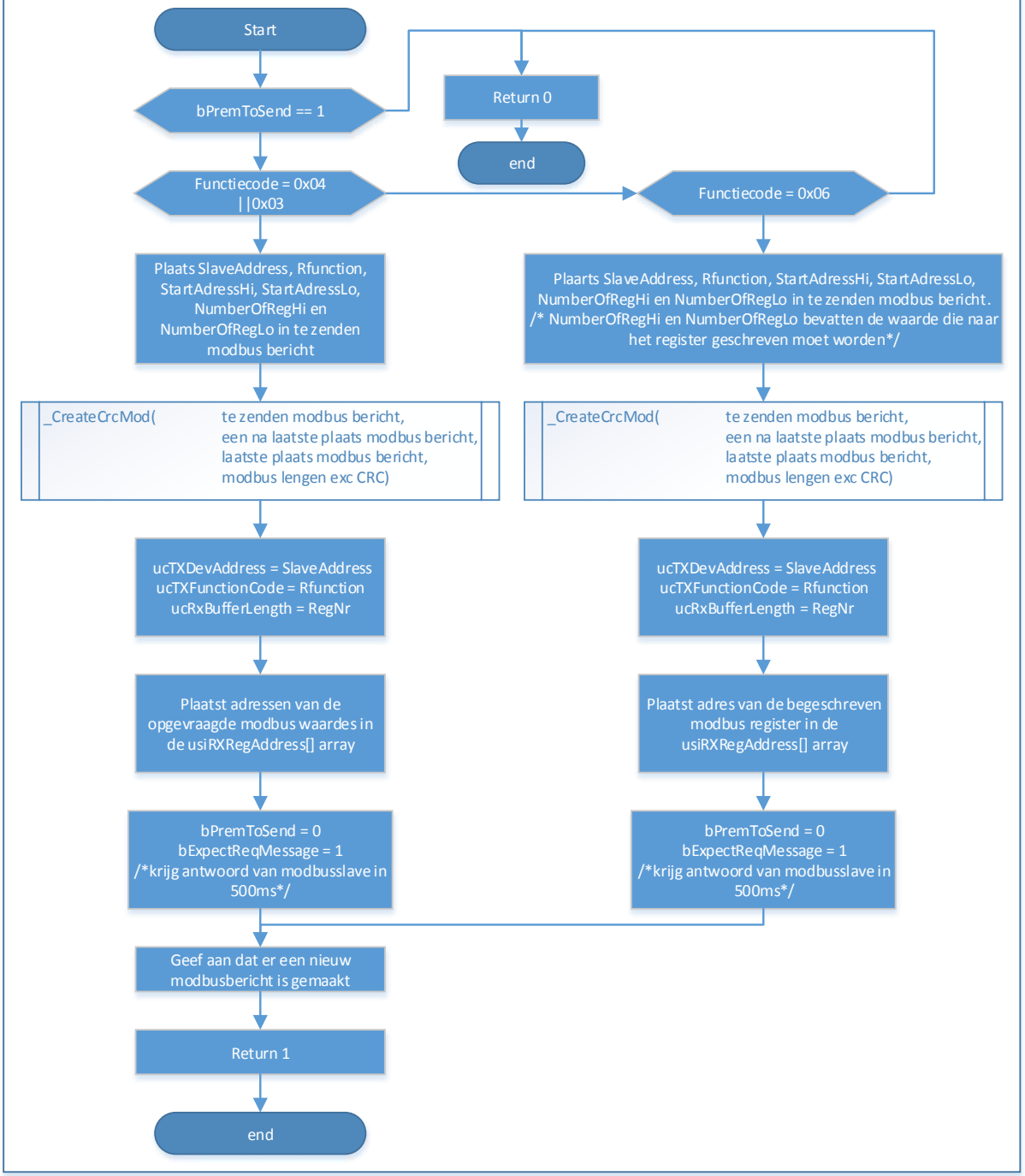

_UpdateSer(struct hcmDataSetSer *B)
Start
_ReceiveSer(B);
_SendBufferSer(B);
End

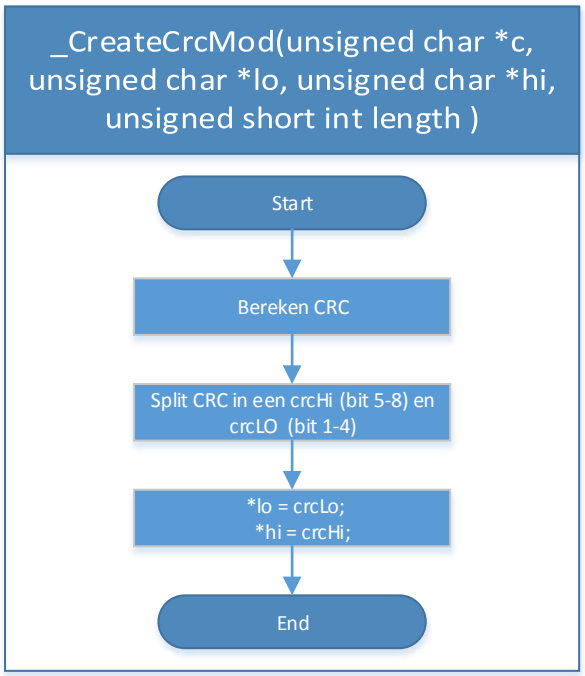

_SetParStop(struct hcmDataSetSer *B, unsigned short int setting)
Start
Stel parity en stopbits in voor compoort van HCM Serieel
End

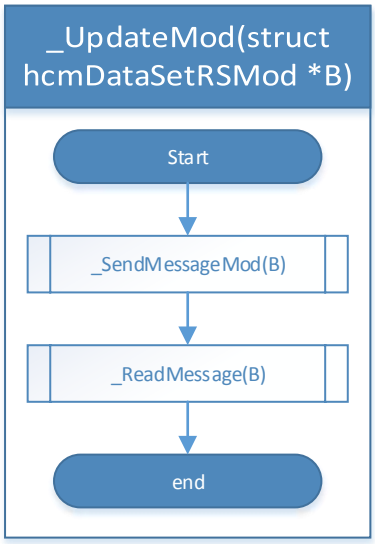
### 1.1.3 HCM RSModbus

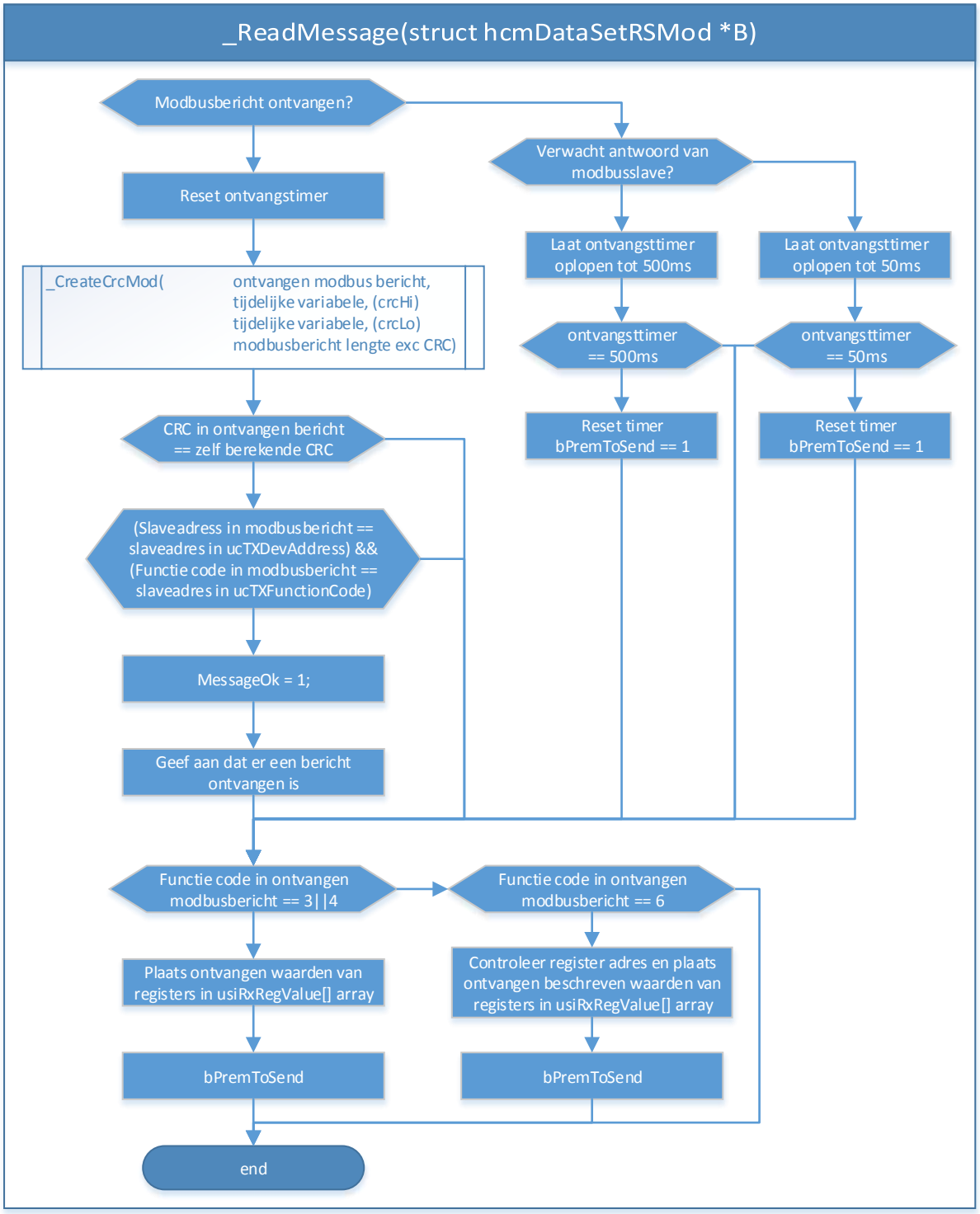


_CreateMessageMod( struct hcmDataSetRSMod *B, unsigned char SlaveAddress, unsigned char RFunction, unsigned char StartAdressHi, unsigned char StartAdressLo, unsigned char NumberOfRegHi, unsigned char NumberOfRegLo)

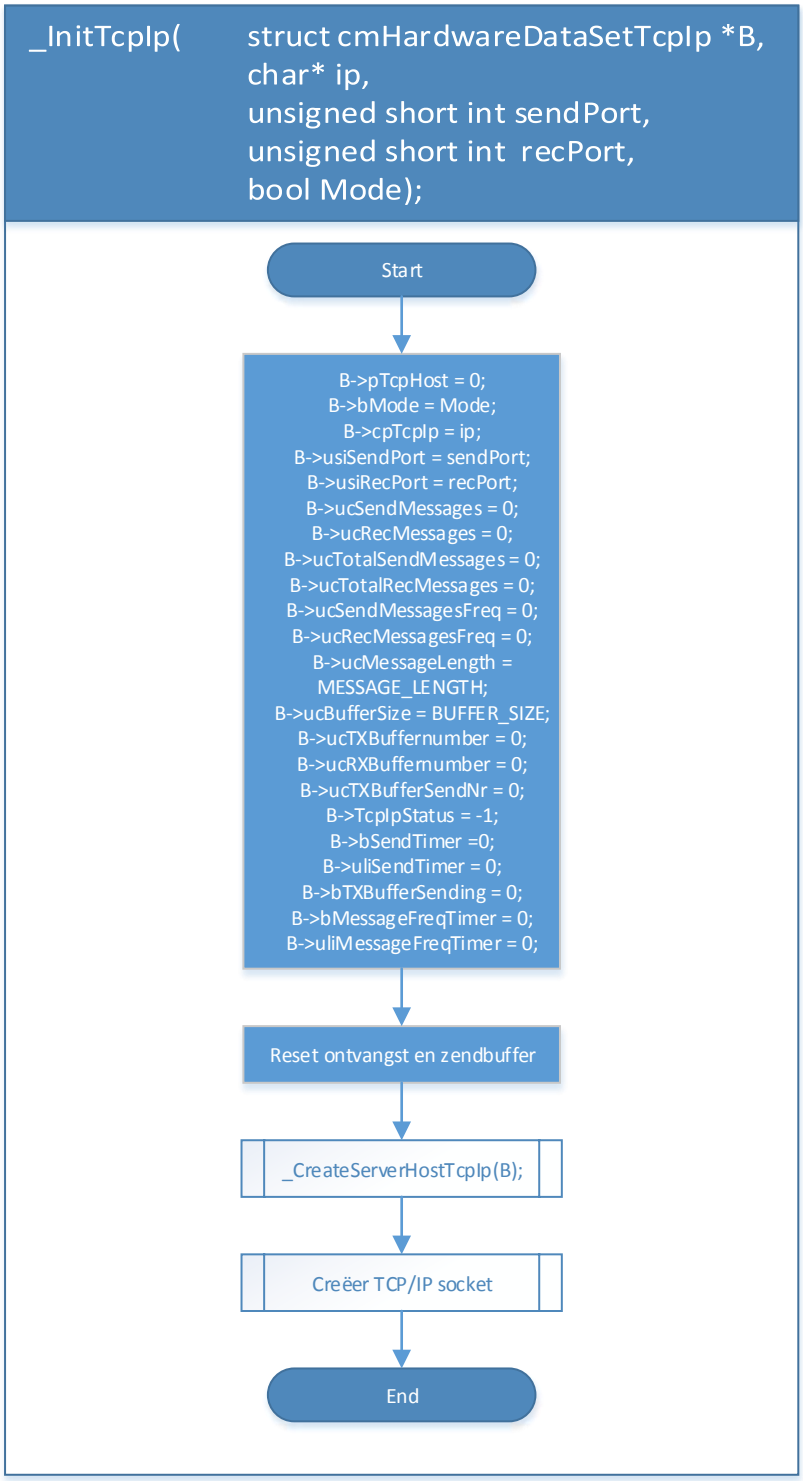


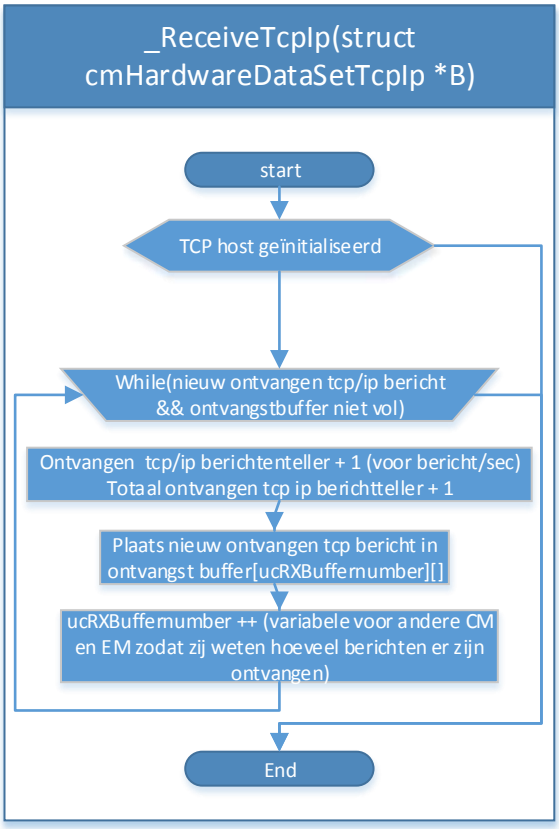

_CreateCrcMod(unsigned char *c, unsigned char *lo, unsigned char *hi, unsigned short int length )
Start
Bereken CRC
Split CRC in een crcHi (bit 5-8) en crcLO (bit 1-4)
*lo = crcLo;
*hi = crcHi;
End

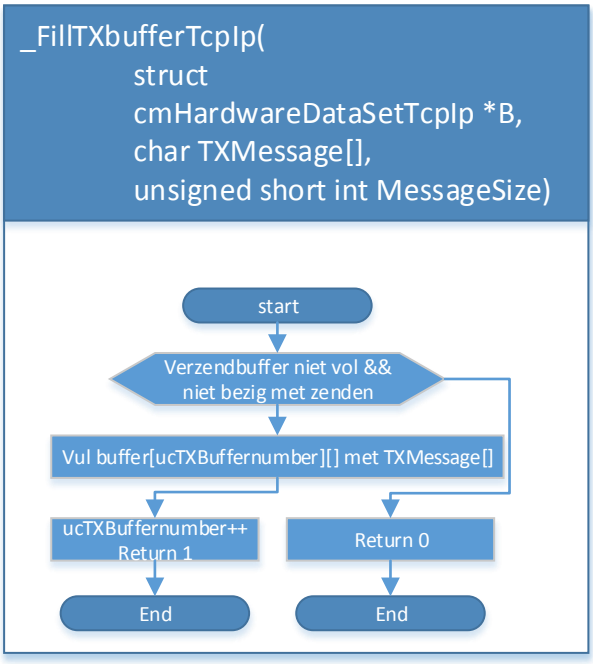

_UpdateMod(struct hcmDataSetRSMod *B)
Start
_SendMessageMod(B)
_ReadMessage(B)
end

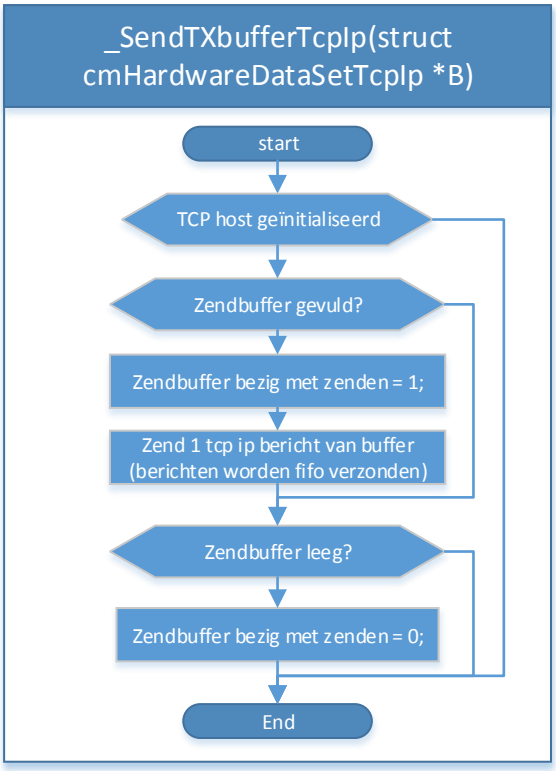

_ReadMessage(struct hcmDataSetRSMod *B)
Modbusbericht ontvangen?
Reset ontvangstimer
_CreateCrcMod(
ontvangen modbus bericht,
tijdelijke variabele, (crcHi)
tijdelijke variabele, (crcLo)
modbusbericht lengte exc CRC)
CRC in ontvangen bericht == zelf berekende CRC
(Slaveadress in modbusbericht == slaveadres in ucTXDevAddress) && (Functie code in modbusbericht == slaveadres in ucTXFunctionCode)
MessageOk = 1;
Geef aan dat er een bericht ontvangen is
Functie code in ontvangen modbusbericht == 3||4
Plaats ontvangen waarden van registers in usiRxRegValue[] array
bPremToSend
Functie code in ontvangen modbusbericht == 6
Controleer register adres en plaats ontvangen beschreven waarden van registers in usiRxRegValue[] array
bPremToSend
end
Verwacht antwoord van modbusslave?
Laat ontvangsttimer oplopen tot 500ms
ontvangsttimer == 500ms
Reset timer bPremToSend == 1
Laat ontvangsttimer oplopen tot 50ms
ontvangsttimer == 50ms
Reset timer bPremToSend == 1

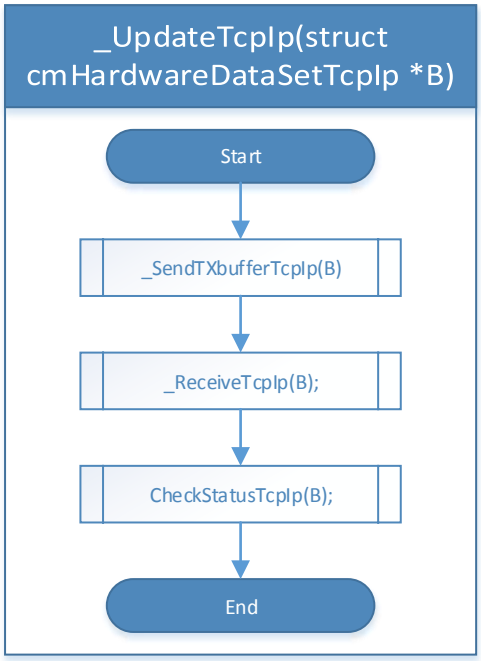
### 1.1.4 HCM TCP/IP



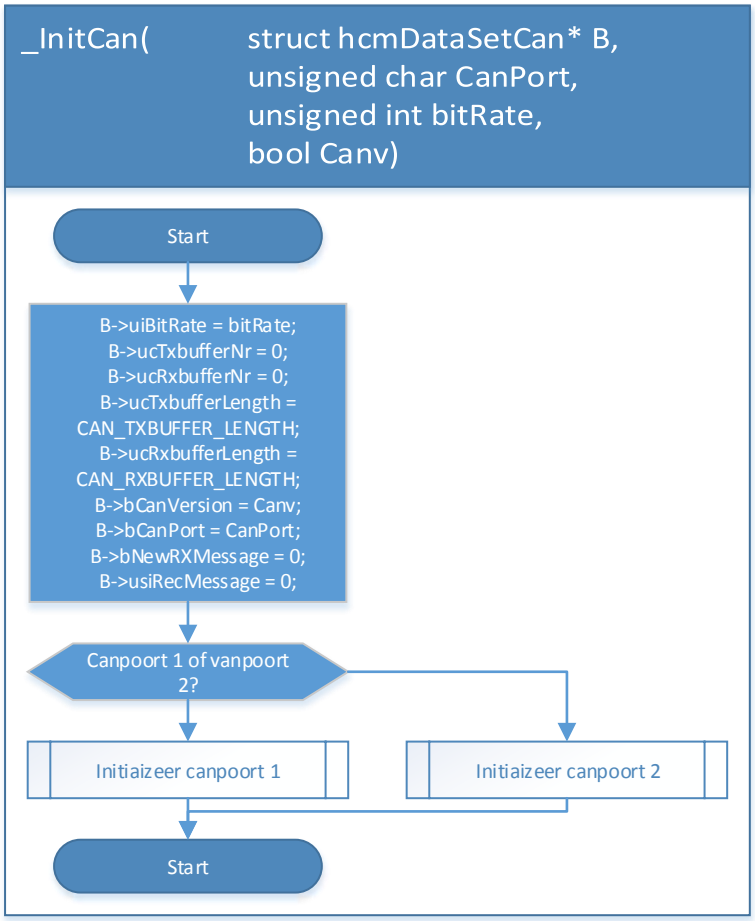

_ReceiveTcpIp(struct cmHardwareDataSetTcpIp *B)
start
TCP host geïnitialiseerd
While(nieuw ontvangen tcp/ip bericht && ontvangstbuffer niet vol)
Ontvangen tcp/ip berichtenteller + 1 (voor bericht/sec)
Totaal ontvangen tcp ip berichtteller + 1
Plaats nieuw ontvangen tcp bericht in ontvangst buffer[ucRXBuffernumber][]
ucRXBuffernumber ++ (variabele voor andere CM en EM zodat zij weten hoeveel berichten er zijn ontvangen)
End

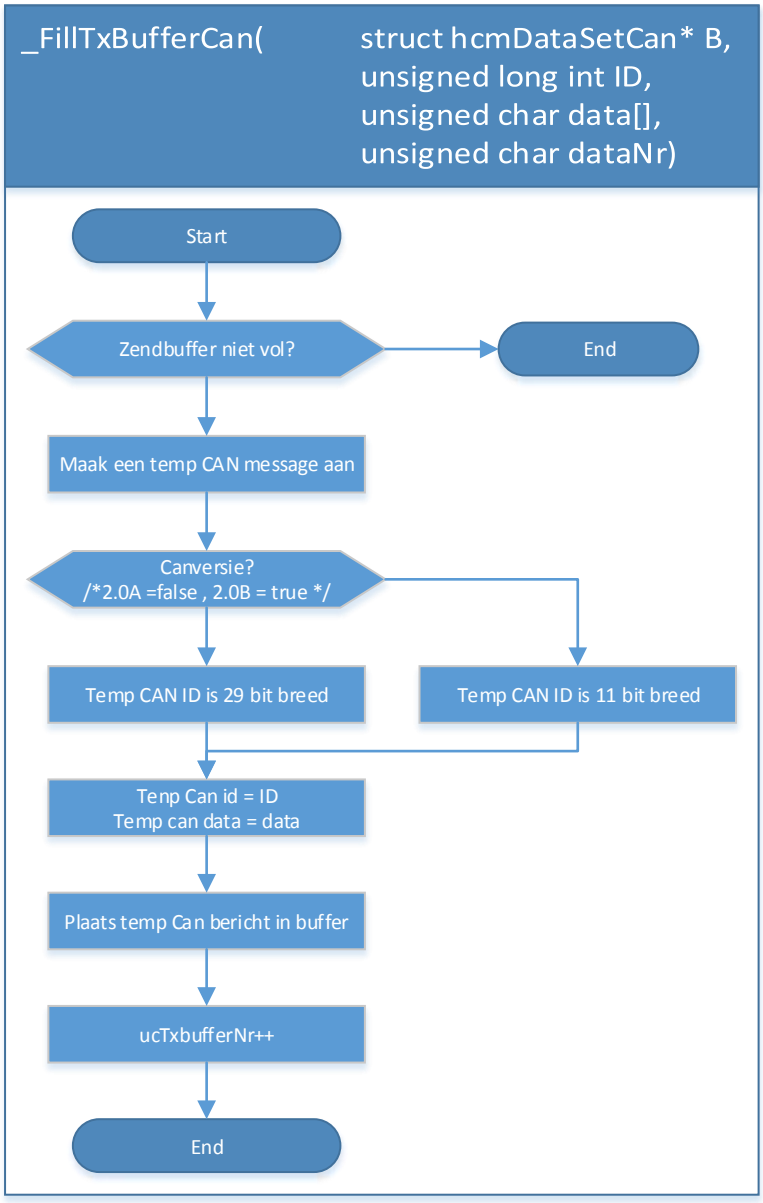

_FillTXbufferTcpIp(
struct
cmHardwareDataSetTcpIp *B,
char TXMessage[],
unsigned short int MessageSize)
start
Verzendbuffer niet vol && niet bezig met zenden
Vul buffer[ucTXBuffernumber][] met TXMessage[]
ucTXBuffernumber++
Return 1
Return 0
End
End

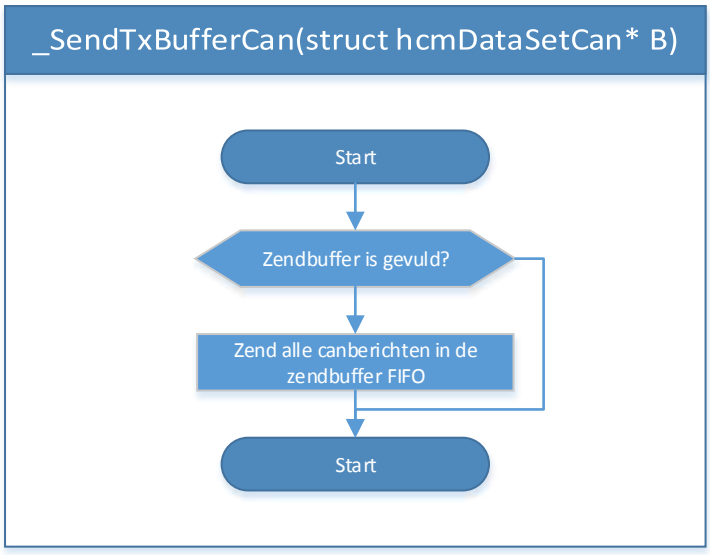

_SendTXbufferTcpIp(struct cmHardwareDataSetTcpIp *B)
start
TCP host geïnitialiseerd
Zendbuffer gevuld?
Zendbuffer bezig met zenden = 1;
Zend 1 tcp ip bericht van buffer (berichten worden fifo verzonden)
Zendbuffer leeg?
Zendbuffer bezig met zenden = 0;
End

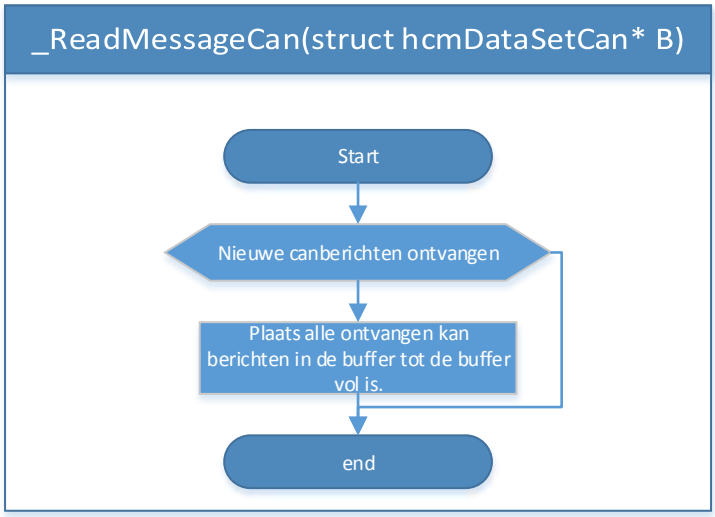

_UpdateTcpIp(struct cmHardwareDataSetTcpIp *B)
Start
_SendTXbufferTcpIp(B)
_ReceiveTcpIp(B);
CheckStatusTcpIp(B);
End

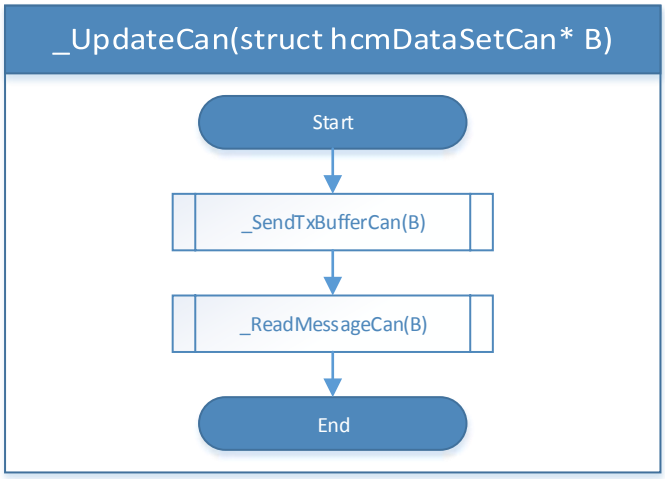
### 1.1.5 HCM CAN


_InitCan( struct hcmDataSetCan* B,
unsigned char CanPort,
unsigned int bitRate,
bool Canv)
Start
B->uiBitRate = bitRate;
B->ucTxbufferNr = 0;
B->ucRxbufferNr = 0;
B->ucTxbufferLength =
CAN_TXBUFFER_LENGTH;
B->ucRxbufferLength =
CAN_RXBUFFER_LENGTH;
B->bCanVersion = Canv;
B->bCanPort = CanPort;
B->bNewRXMessage = 0;
B->usiRecMessage = 0;
Canpoort 1 of vanpoort 2?
Initiaizeer canpoort 1
Initiaizeer canpoort 2
Start

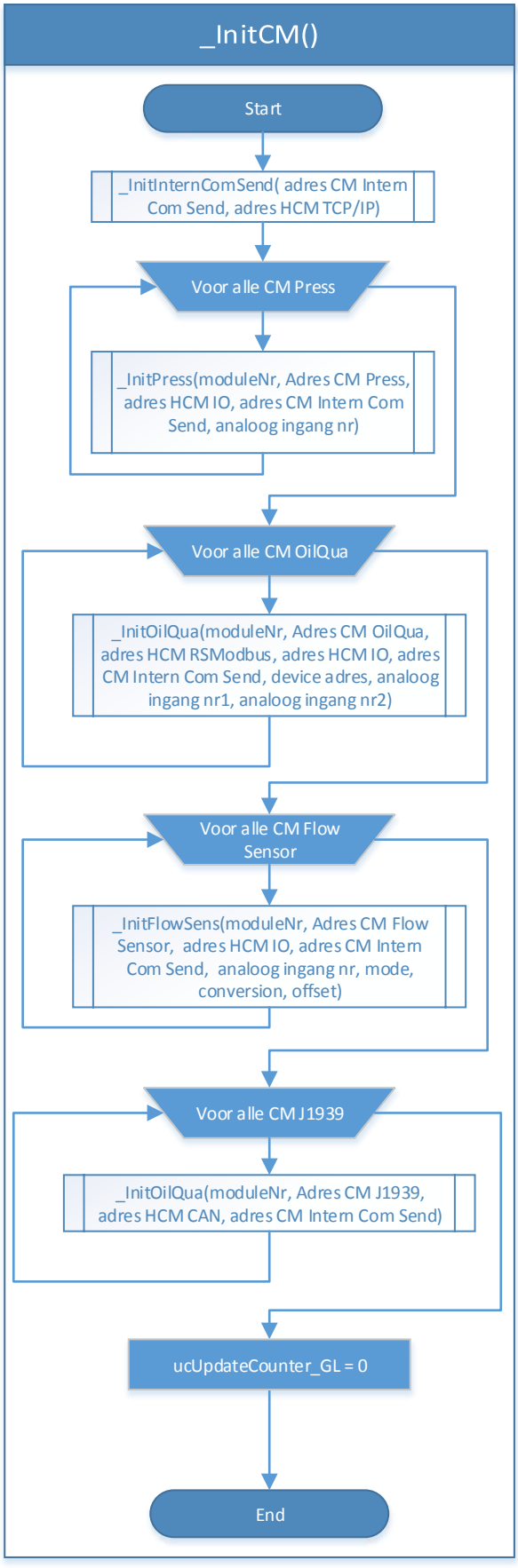

_FillTxBufferCan( struct hcmDataSetCan* B,
unsigned long int ID,
unsigned char data[],
unsigned char dataNr)
Start
Zendbuffer niet vol?
End
Maak een temp CAN message aan
Canversie?
/*2.0A =false , 2.0B = true */
Temp CAN ID is 29 bit breed
Temp CAN ID is 11 bit breed
Tenp Can id = ID
Temp can data = data
Plaats temp Can bericht in buffer
ucTxbufferNr++
End

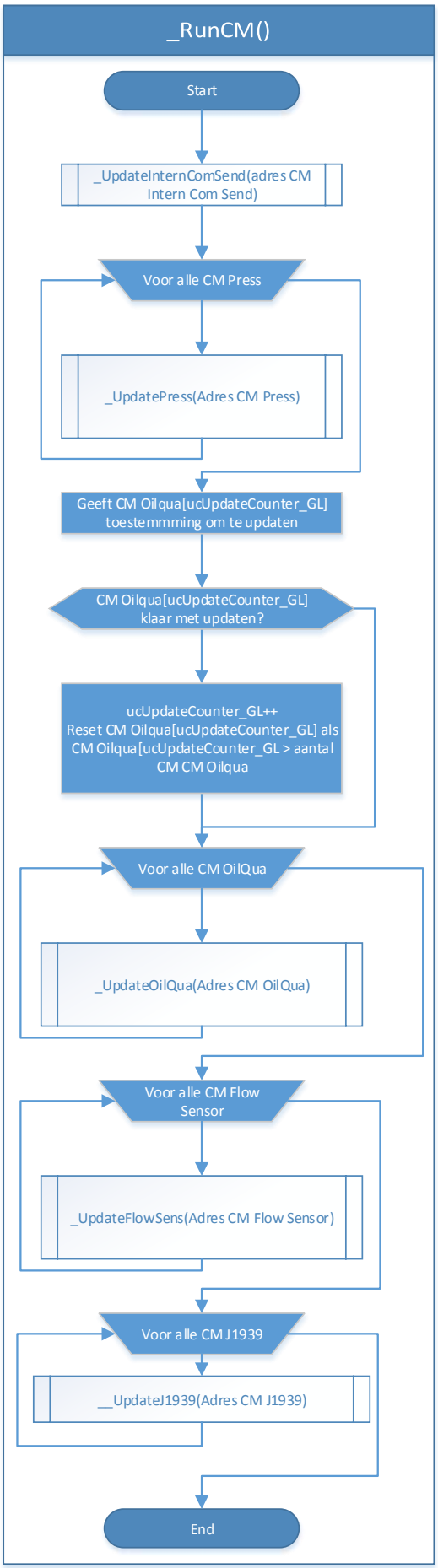

_SendTxBufferCan(struct hcmDataSetCan* B)
Start
Zendbuffer is gevuld?
Zend alle canberichten in de zendbuffer FIFO
Start


_ReadMessageCan(struct hcmDataSetCan* B)
Start
Nieuwe canberichten ontvangen
Plaats alle ontvangen kan berichten in de buffer tot de buffer vol is.
end

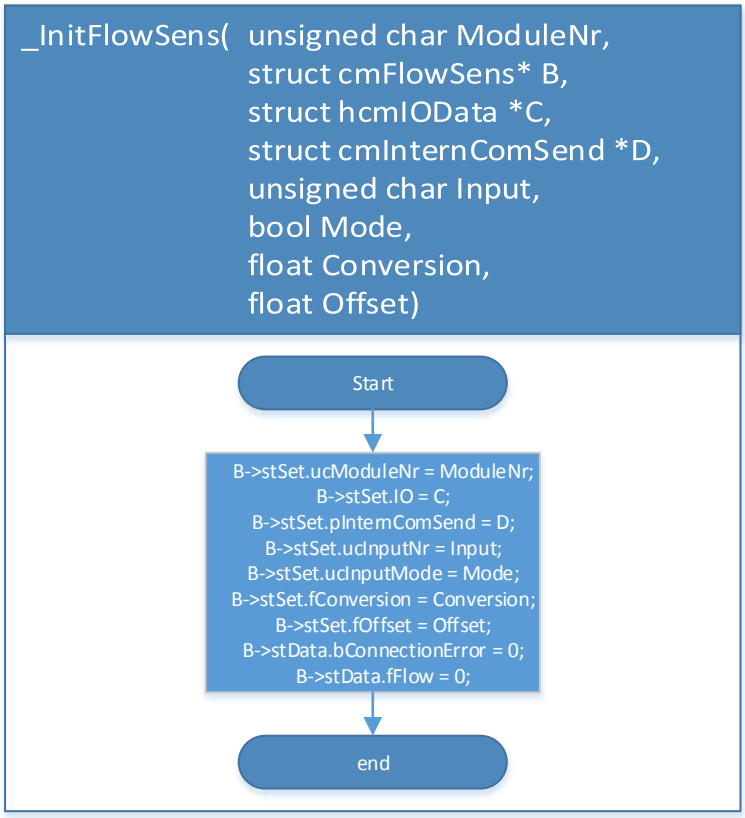

_UpdateCan(struct hcmDataSetCan* B)
Start
_SendTxBufferCan(B)
_ReadMessageCan(B)
End

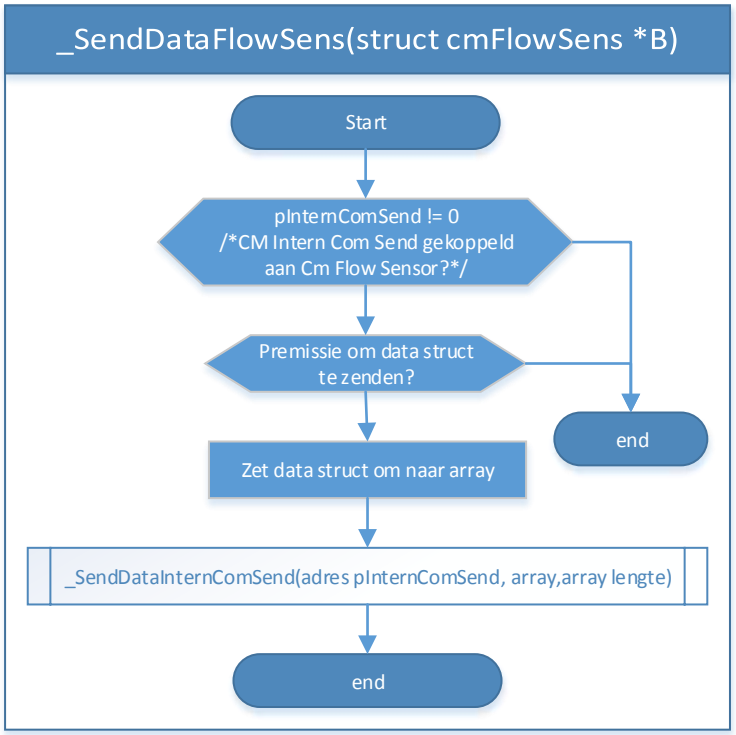
## 1.2 Control modules

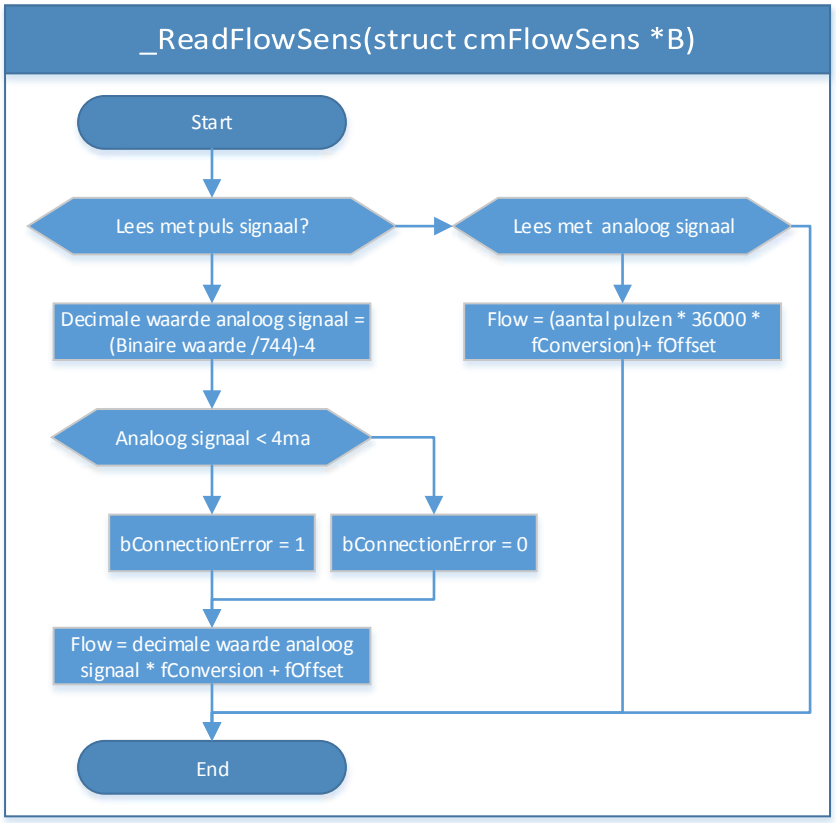


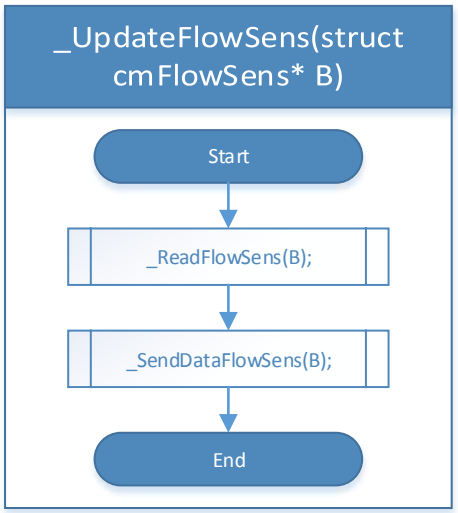

_RunCM()
Start
_UpdateInternComSend(adres CM Intern Com Send)
Voor alle CM Press
_UpdatePress(Adres CM Press)
Geeft CM Oilqua[ucUpdateCounter_GL] toestemmming om te updaten
CM Oilqua[ucUpdateCounter_GL] klaar met updaten?
ucUpdateCounter_GL++
Reset CM Oilqua[ucUpdateCounter_GL] als CM Oilqua[ucUpdateCounter_GL > aantal CM CM Oilqua
Voor alle CM OilQua
_UpdateOilQua(Adres CM OilQua)
Voor alle CM Flow Sensor
_UpdateFlowSens(Adres CM Flow Sensor)
Voor alle CM J1939
__UpdateJ1939(Adres CM J1939)
End

### 1.2.1 CM Flow sensor

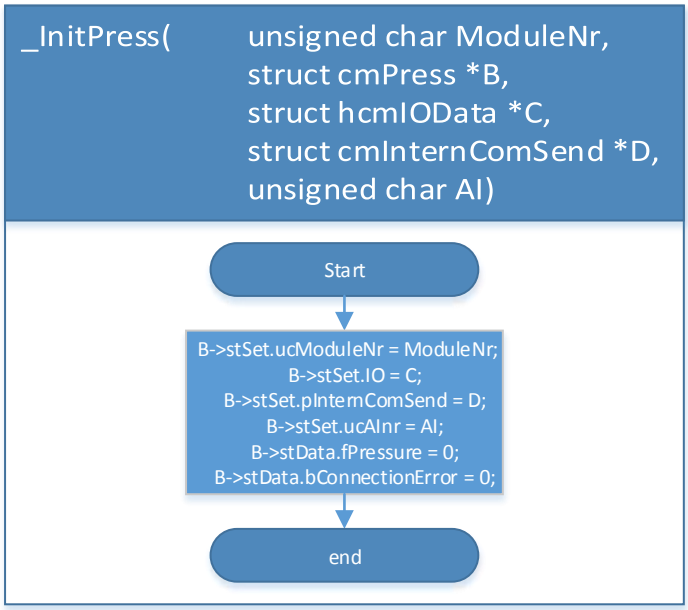


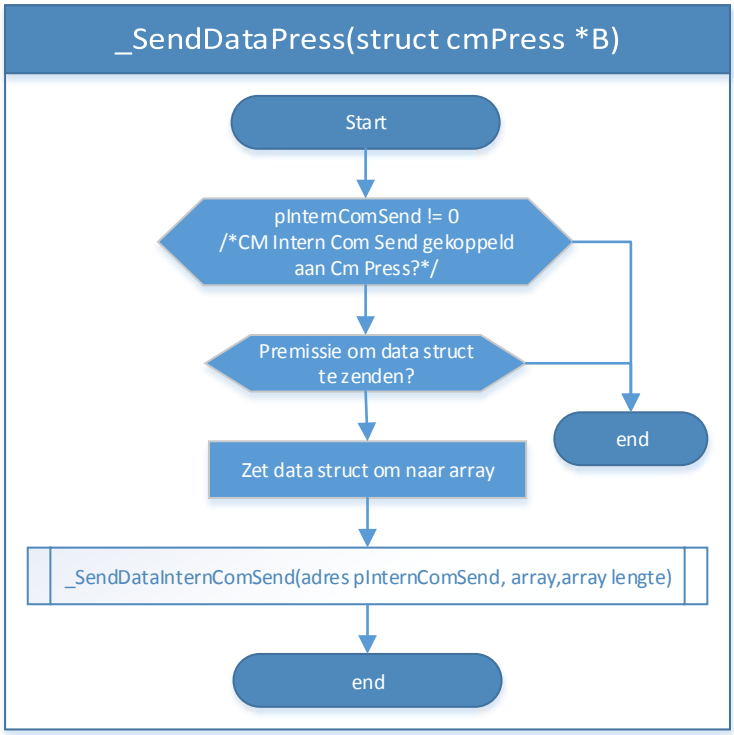



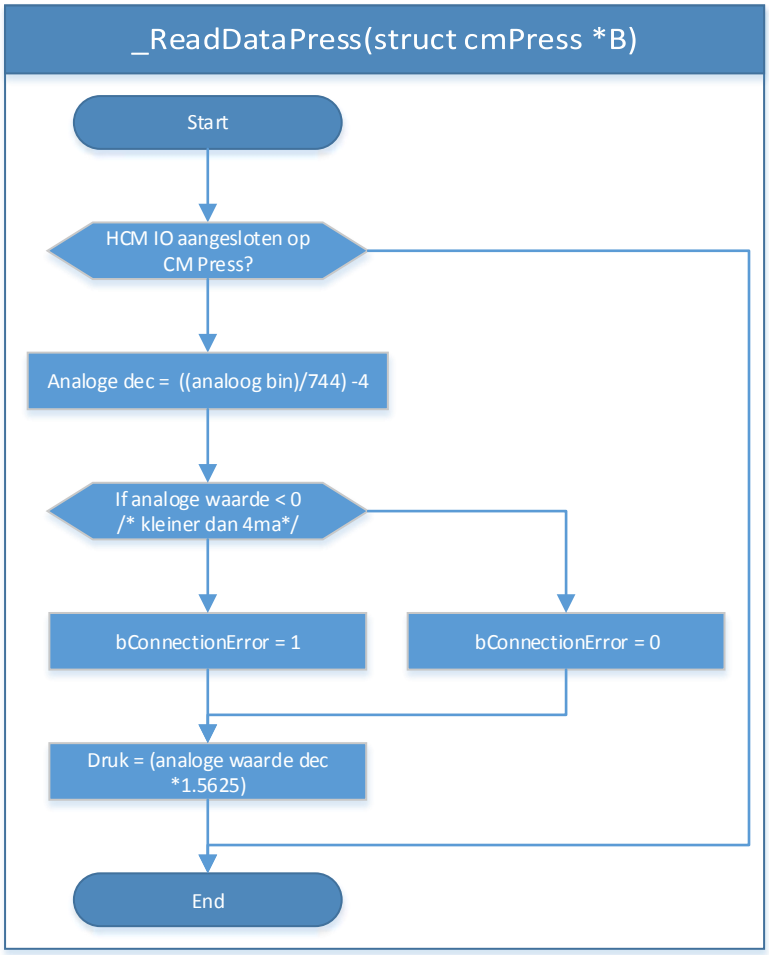

_ReadFlowSens(struct cmFlowSens *B)
Start
Lees met puls signaal?
Lees met analoog signaal
Decimale waarde analoog signaal = (Binaire waarde /744)-4
Flow = (aantal pulzen * 36000 * fConversion)+ fOffset
Analoog signaal < 4ma
bConnectionError = 1
bConnectionError = 0
Flow = decimale waarde analoog signaal * fConversion + fOffset
End

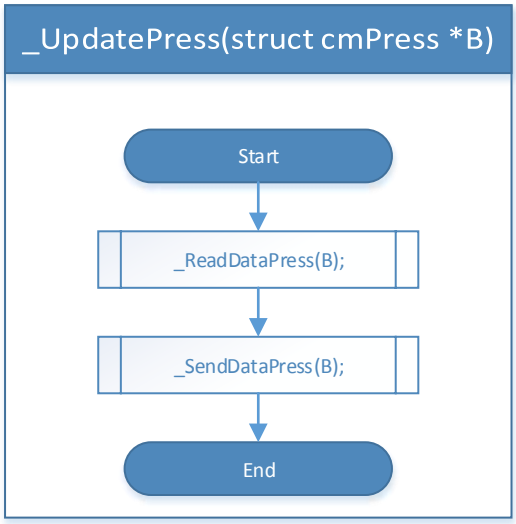

_UpdateFlowSens(struct cmFlowSens* B)
Start
_ReadFlowSens(B);
_SendDataFlowSens(B);
End

### 1.2.2 Pressure sensor control module

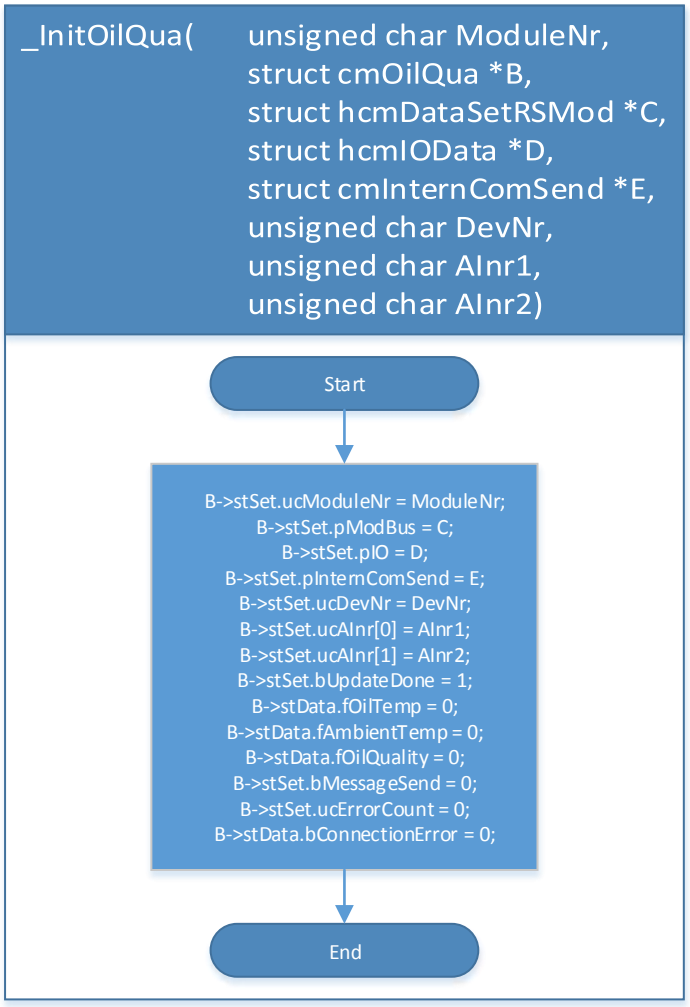


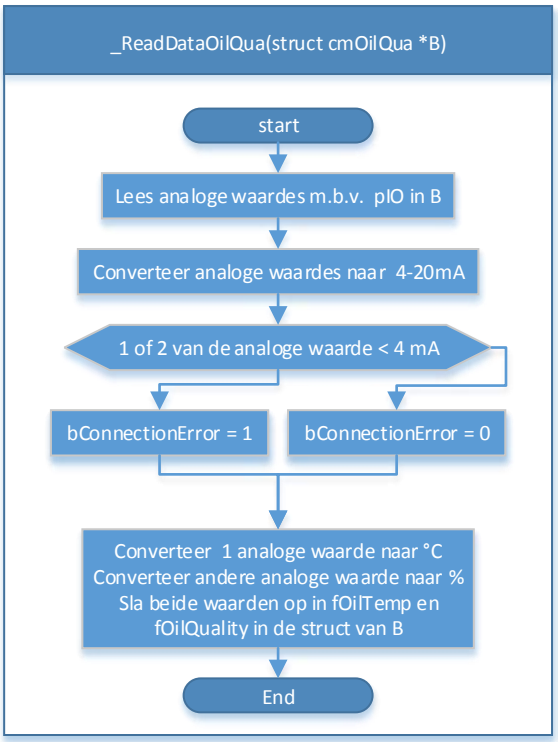



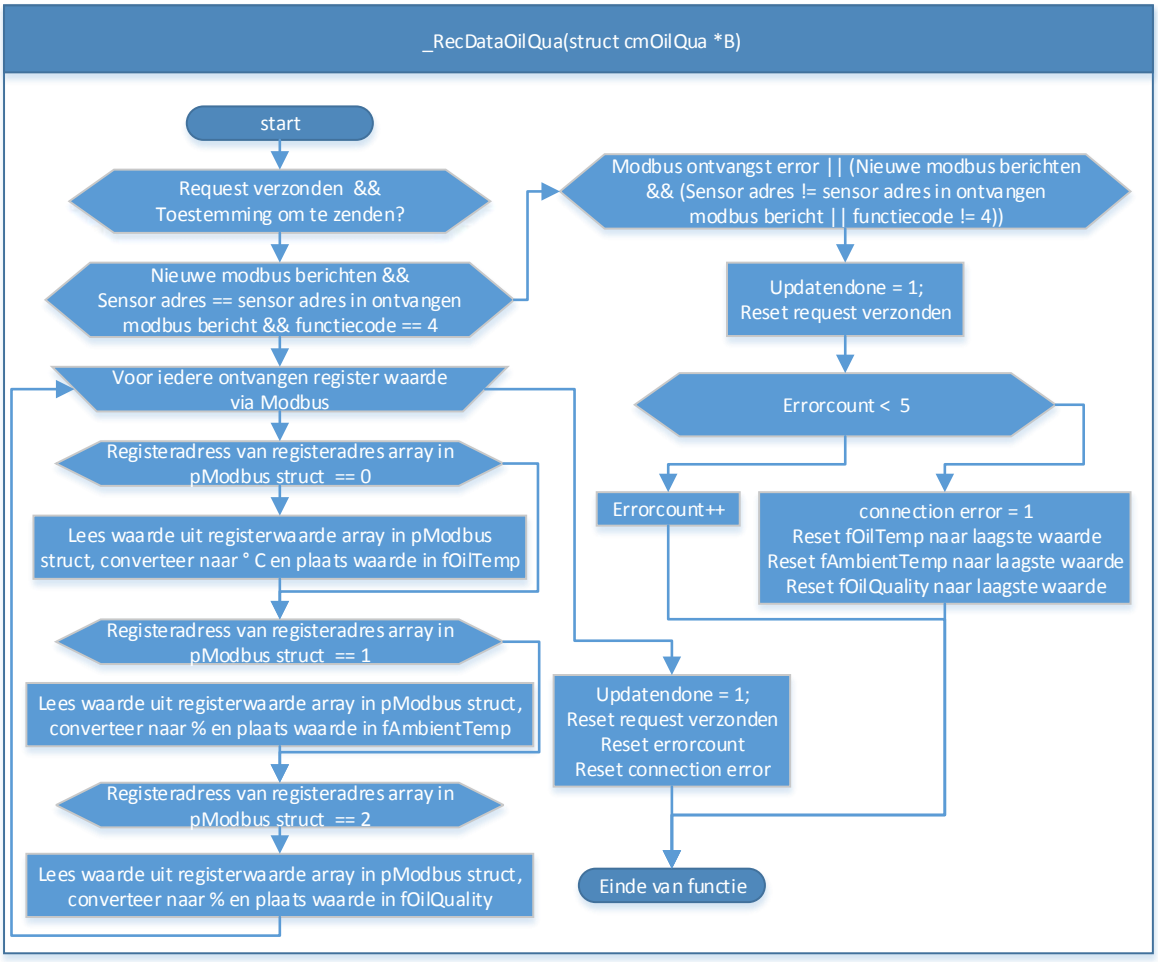

_ReadDataPress(struct cmPress *B)
Start
HCM IO aangesloten op CM Press?
Analoge dec = ((analoog bin)/744) -4
If analoge waarde < 0 /* kleiner dan 4ma*/
bConnectionError = 1
bConnectionError = 0
Druk = (analoge waarde dec *1.5625)
End

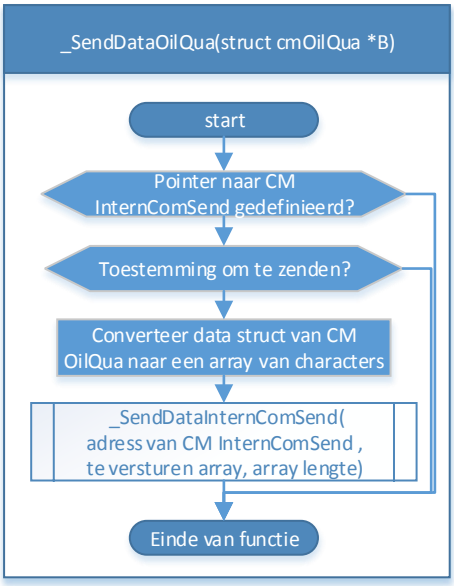

_UpdatePress(struct cmPress *B)
Start
_ReadDataPress(B);
_SendDataPress(B);
End

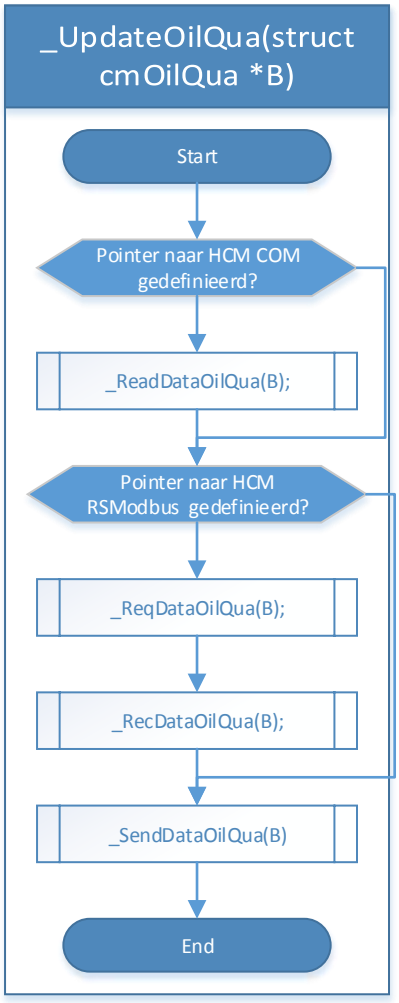
### 1.2.3 Oil quality sensor control module



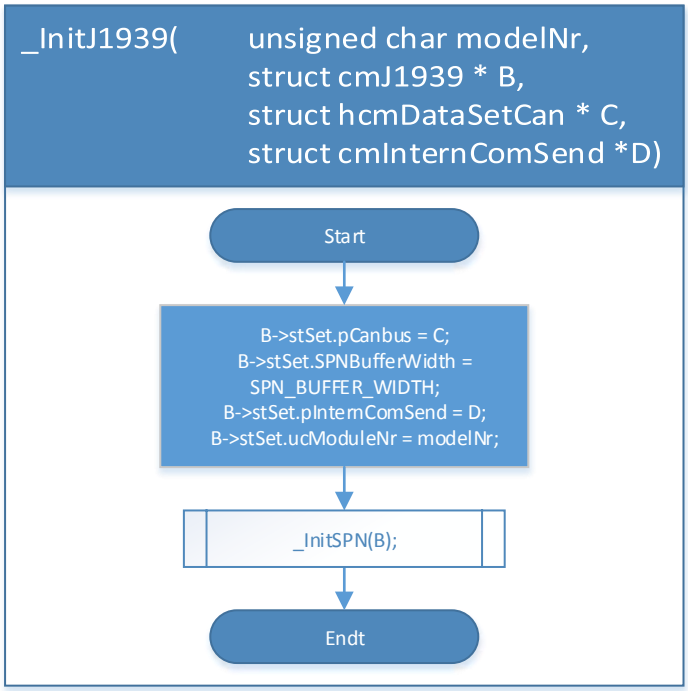



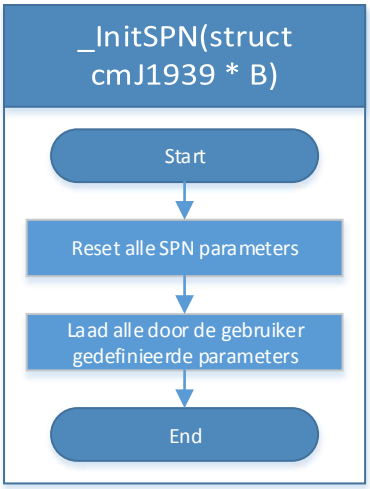

_RecDataOilQua(struct cmOilQua *B)
start
Request verzonden && Toestemming om te zenden?
Nieuwe modbus berichten && Sensor adres == sensor adres in ontvangen modbus bericht && functiecode == 4
Modbus ontvangst error || (Nieuwe modbus berichten && (Sensor adres != sensor adres in ontvangen modbus bericht || functiecode != 4))
Updatendone = 1; Reset request verzonden
Errorcount < 5
Errorcount++
connection error = 1 Reset fOilTemp naar laagste waarde Reset fAmbientTemp naar laagste waarde Reset fOilQuality naar laagste waarde
Voor iedere ontvangen register waarde via Modbus
Registeradress van registeradres array in pModbus struct == 0
Lees waarde uit registerwaarde array in pModbus struct, converteer naar ° C en plaats waarde in fOilTemp
Registeradress van registeradres array in pModbus struct == 1
Lees waarde uit registerwaarde array in pModbus struct, converteer naar % en plaats waarde in fAmbientTemp
Registeradress van registeradres array in pModbus struct == 2
Lees waarde uit registerwaarde array in pModbus struct, converteer naar % en plaats waarde in fOilQuality
Updatendone = 1; Reset request verzonden Reset errorcount Reset connection error
Einde van functie

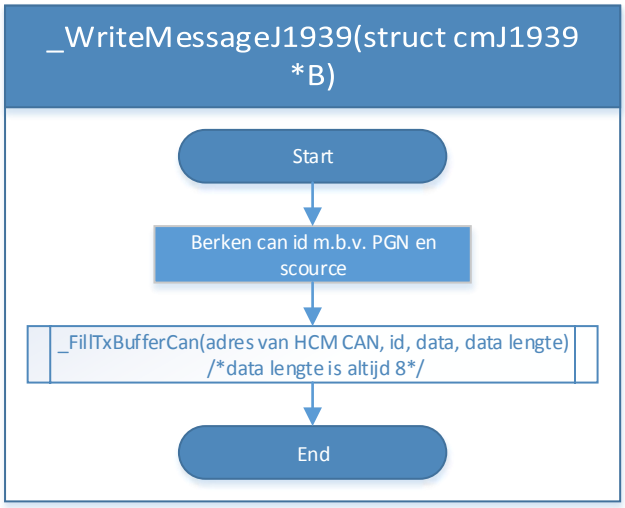

_SendDataOilQua(struct cmOilQua *B)
start
Pointer naar CM InternComSend gedefinieerd?
Toestemming om te zenden?
Converteer data struct van CM OilQua naar een array van characters
_SendDataInternComSend( adress van CM InternComSend , te versturen array, array lengte)
Einde van functie

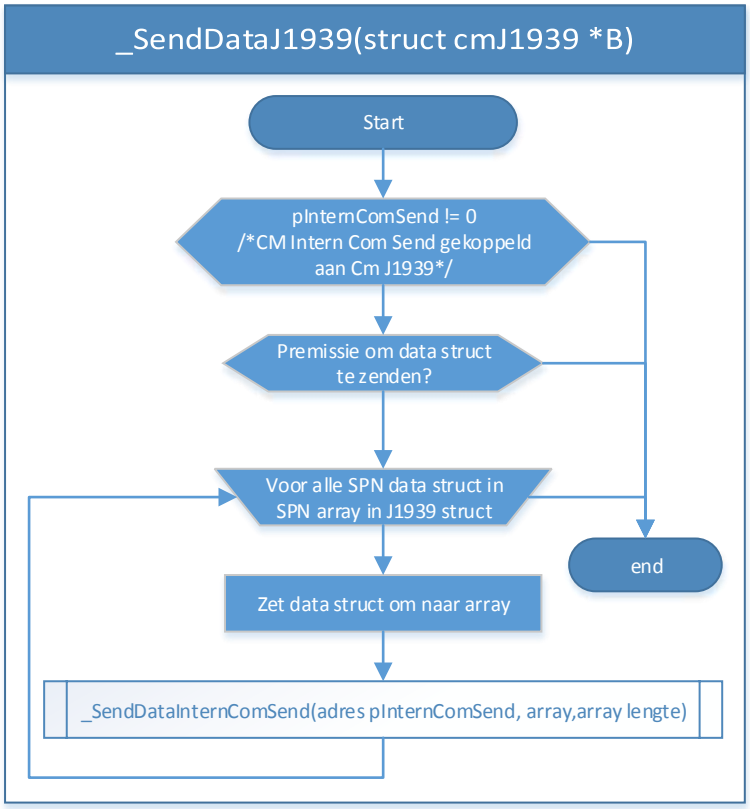

_UpdateOilQua(struct cmOilQua *B)
Start
Pointer naar HCM COM gedefinieerd?
_ReadDataOilQua(B);
Pointer naar HCM RSModbus gedefinieerd?
_ReqDataOilQua(B);
_RecDataOilQua(B);
_SendDataOilQua(B)
End

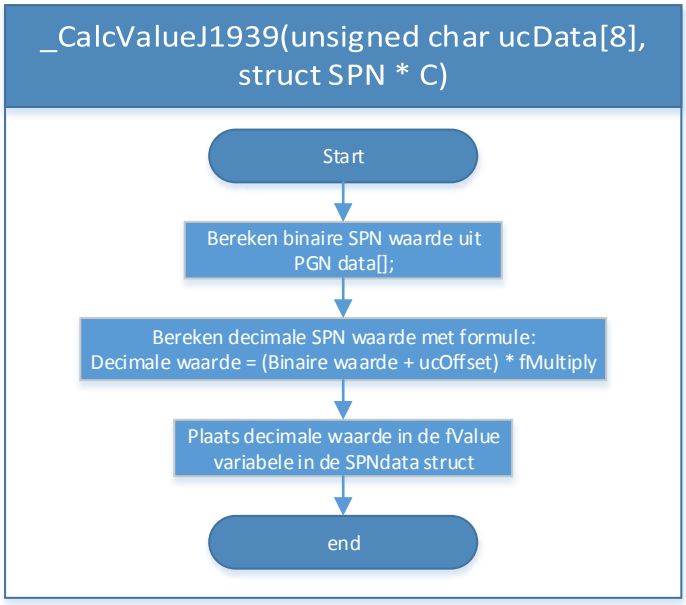
### 1.2.4 J1939 control module

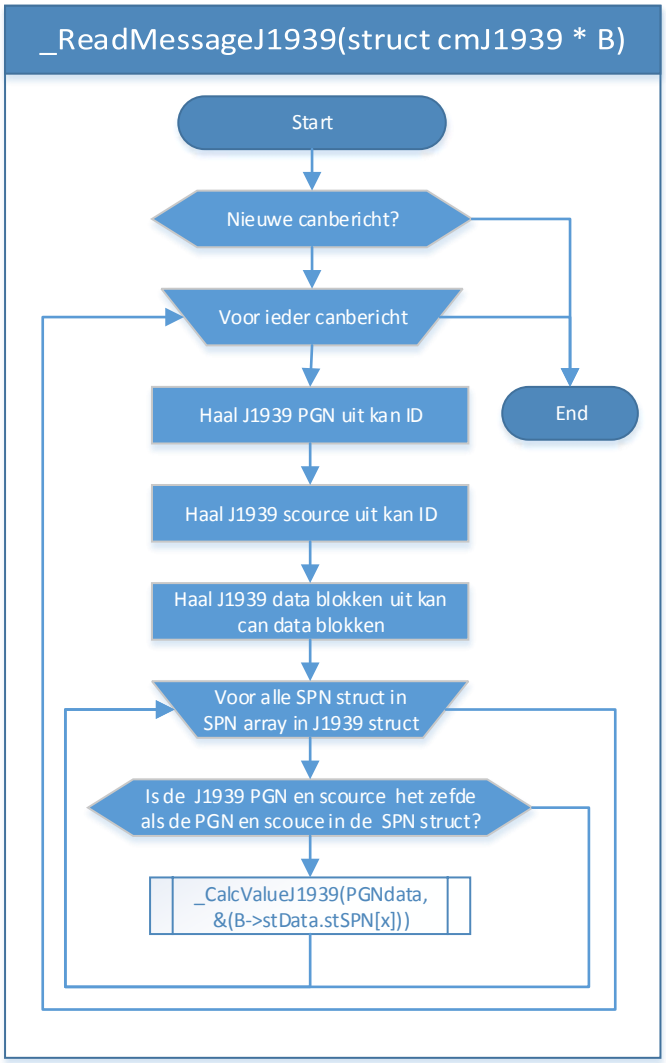


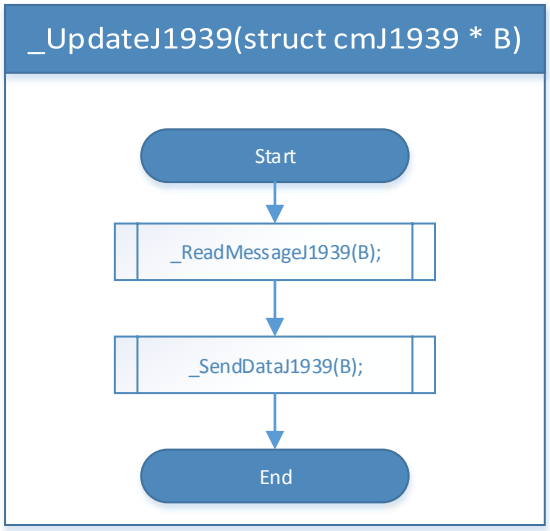



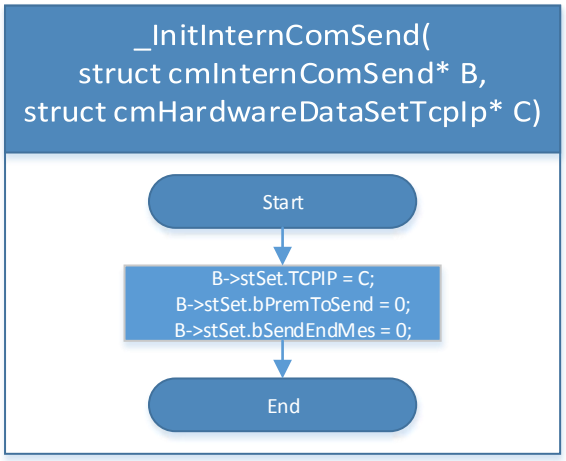

_SendDataJ1939(struct cmJ1939 *B)
Start
pInternComSend != 0
/*CM Intern Com Send gekoppeld
aan Cm J1939*/
Premissie om data struct
te zenden?
Voor alle SPN data struct in
SPN array in J1939 struct
end
Zet data struct om naar array
_SendDataInternComSend(adres pInternComSend, array,array lengte)

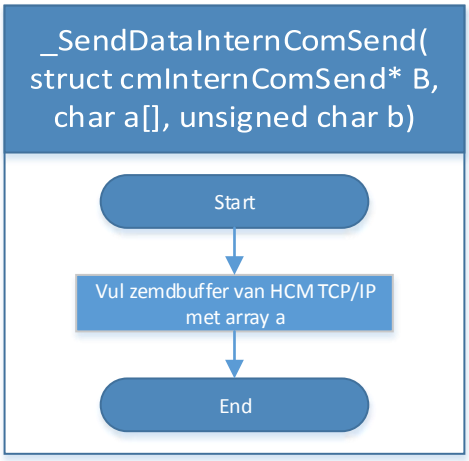

_CalcValueJ1939(unsigned char ucData[8],
struct SPN * C)
Start
Bereken binaire SPN waarde uit
PGN data[];
Bereken decimale SPN waarde met formule:
Decimale waarde = (Binaire waarde + ucOffset) * fMultiply
Plaats decimale waarde in de fValue
variabele in de SPNdata struct
end

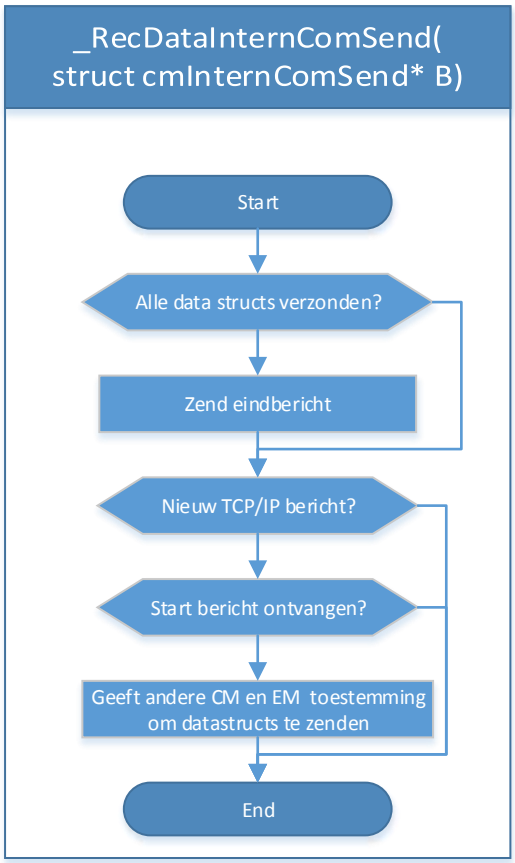

_ReadMessageJ1939(struct cmJ1939 * B)
Start
Nieuwe canbericht?
Voor ieder canbericht
End
Haal J1939 PGN uit kan ID
Haal J1939 scource uit kan ID
Haal J1939 data blokken uit kan can data blokken
Voor alle SPN struct in SPN array in J1939 struct
Is de J1939 PGN en scource het zefde als de PGN en scouce in de SPN struct?
_CalcValueJ1939(PGNdata, &(B->stData.stSPN[x]))

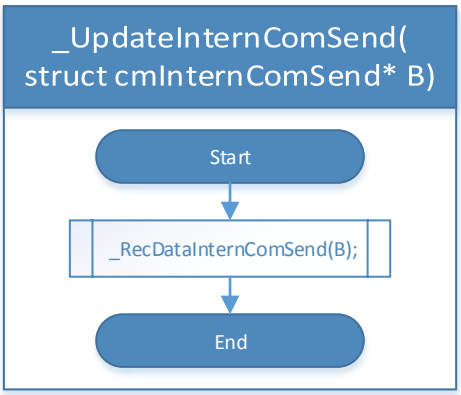

_UpdateJ1939(struct cmJ1939 * B)
Start
_ReadMessageJ1939(B);
_SendDataJ1939(B);
End

### 1.2.5 CM Intern Com Send

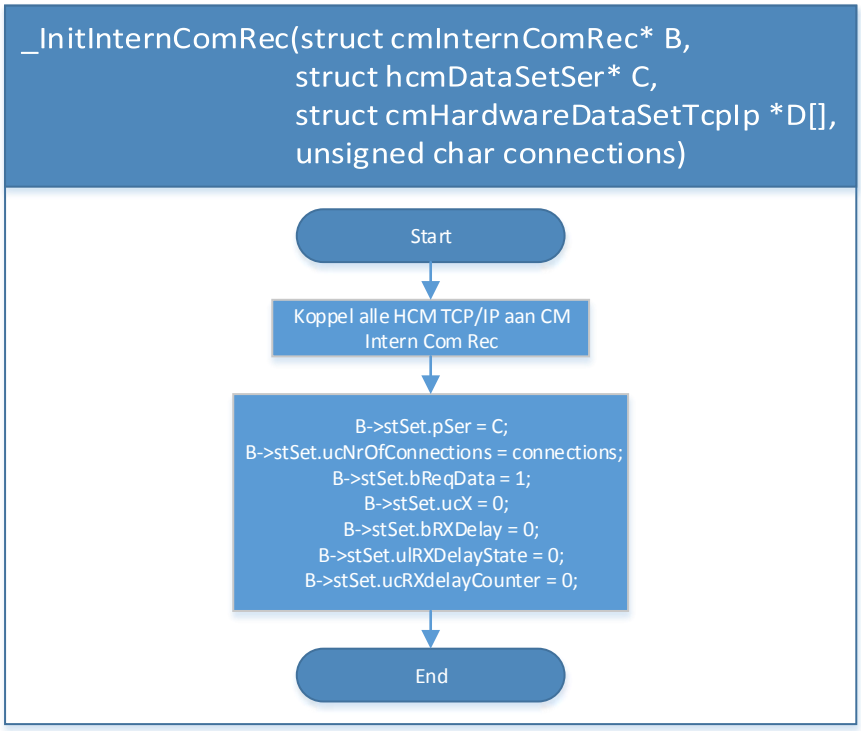


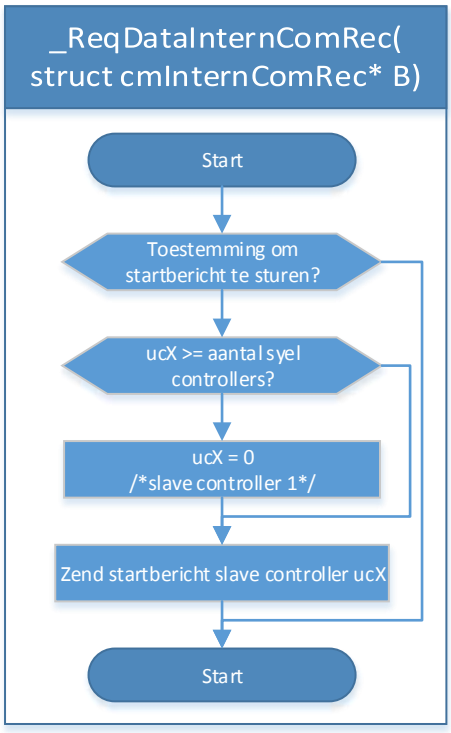



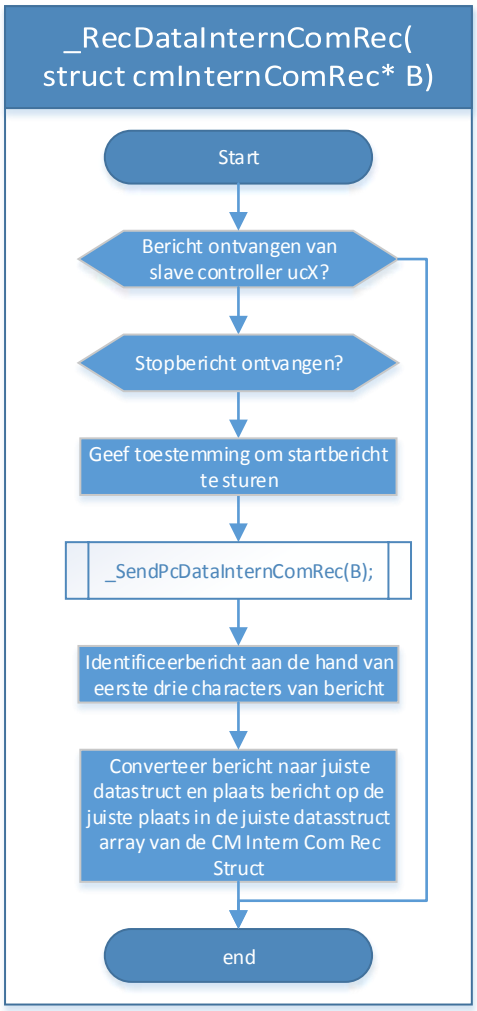



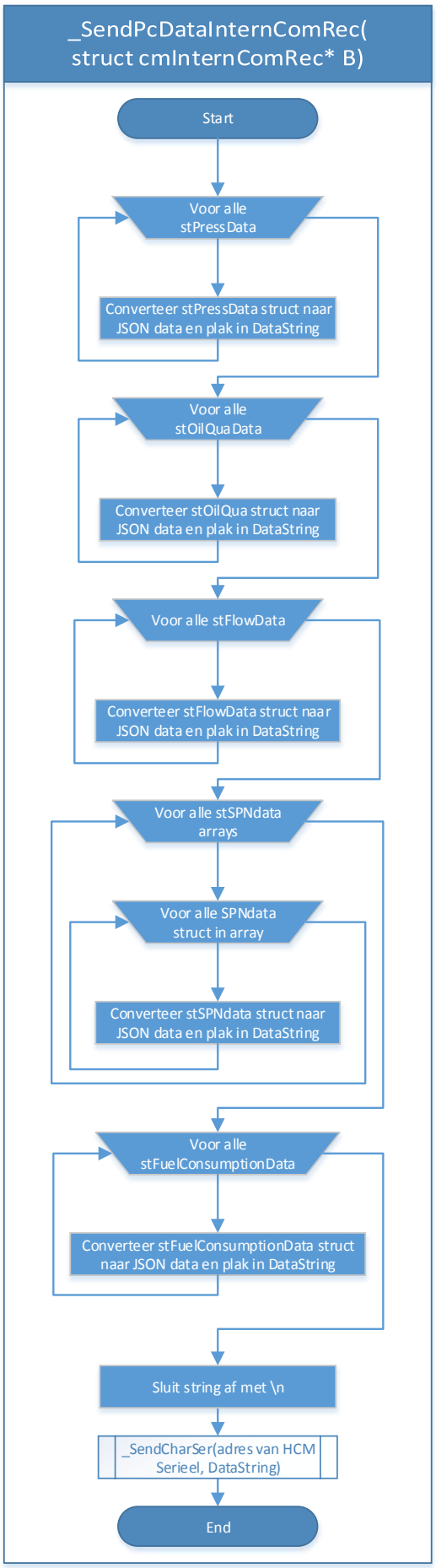



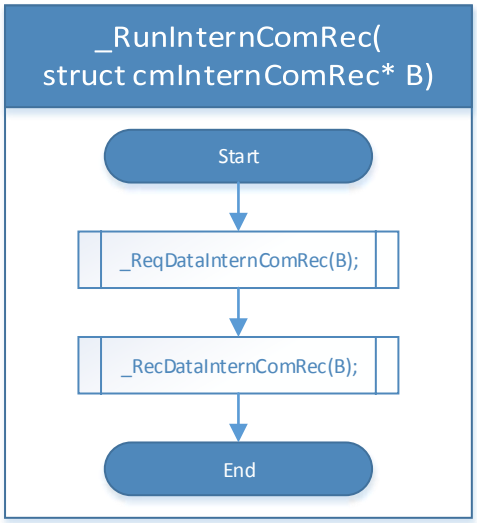
### 1.2.6 CM Intern Com Rec



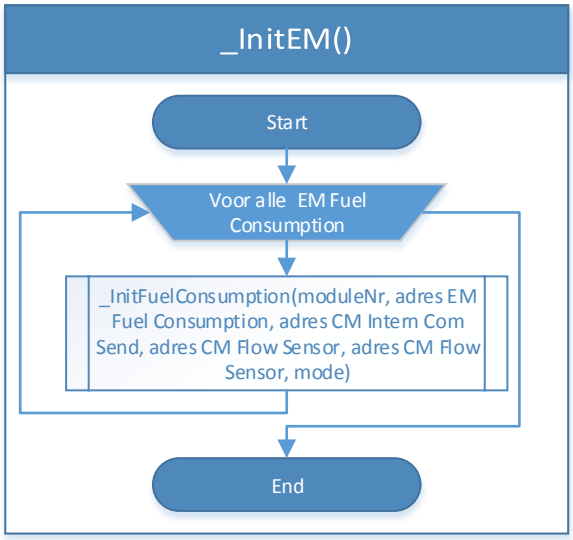



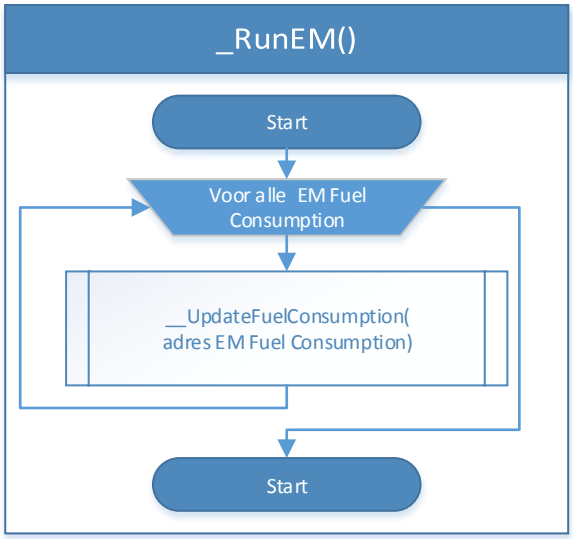

_RecDataInternComRec(
struct cmInternComRec* B)
Start
Bericht ontvangen van slave controller ucX?
Stopbericht ontvangen?
Geef toestemming om startbericht te sturen
_SendPcDataInternComRec(B);
Identificeerbericht aan de hand van eerste drie characters van bericht
Converteer bericht naar juiste datastruct en plaats bericht op de juiste plaats in de juiste datasstruct array van de CM Intern Com Rec Struct
end


_SendPcDataInternComRec(
struct cmInternComRec* B)
Start
Voor alle stPressData
Converteer stPressData struct naar JSON data en plak in DataString
Voor alle stOilQuaData
Converteer stOilQua struct naar JSON data en plak in DataString
Voor alle stFlowData
Converteer stFlowData struct naar JSON data en plak in DataString
Voor alle stSPNdata arrays
Voor alle SPNdata struct in array
Converteer stSPNdata struct naar JSON data en plak in DataString
Voor alle stFuelConsumptionData
Converteer stFuelConsumptionData struct naar JSON data en plak in DataString
Sluit string af met \n
_SendCharSer(adres van HCM Serieel, DataString)
End

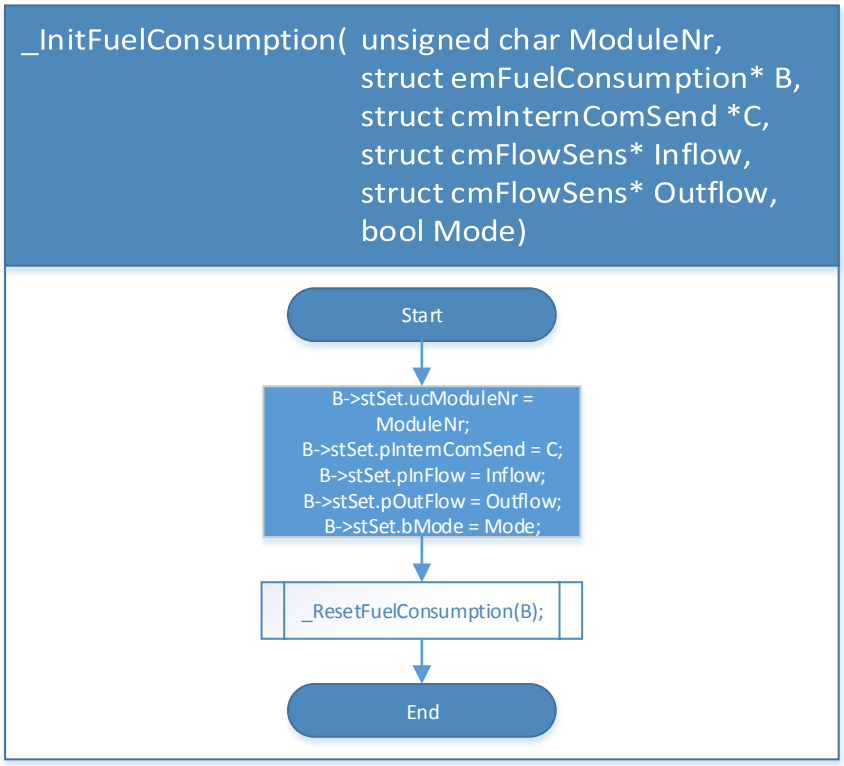

_RunInternComRec(
struct cmInternComRec* B)
Start
_ReqDataInternComRec(B);
_RecDataInternComRec(B);
End

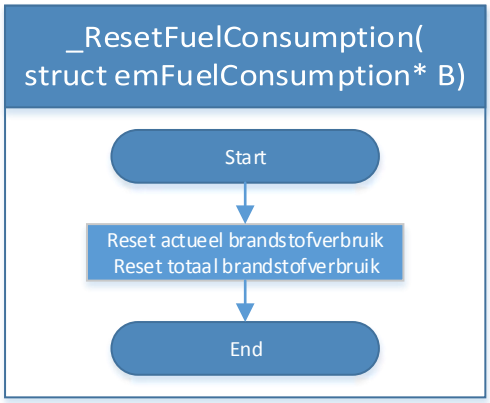
## 1.3 Equipment modules

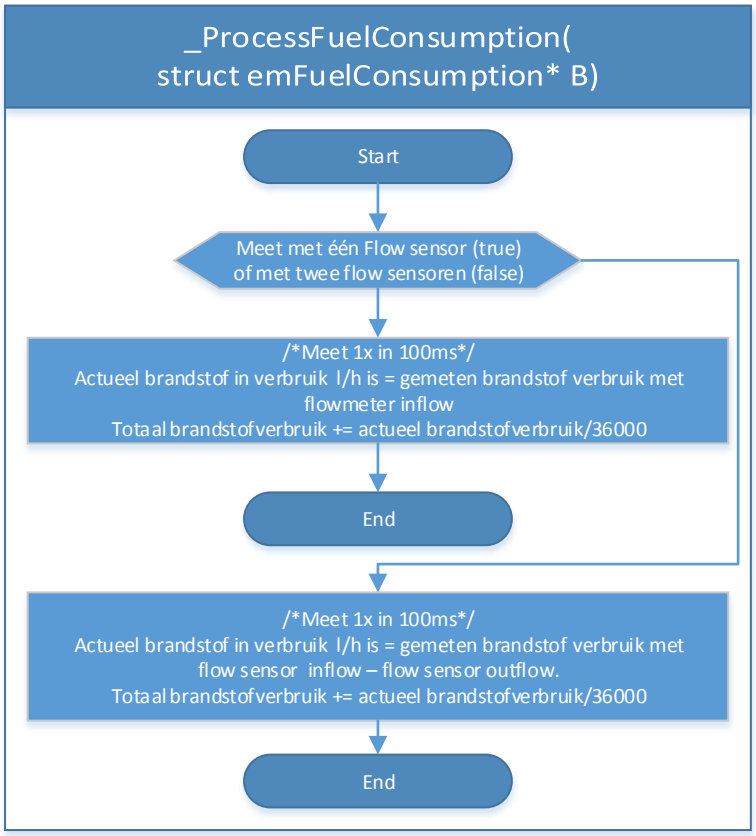


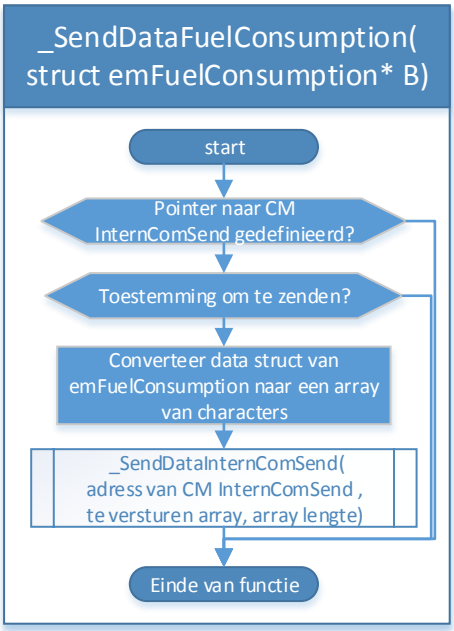



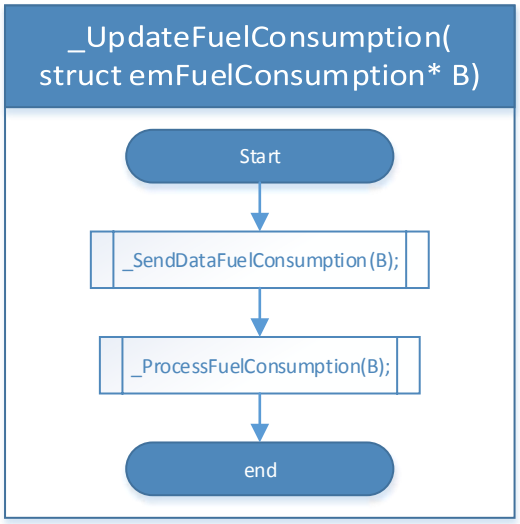
### 1.3.1 EM Fuel Consumption




_ProcessFuelConsumption(
struct emFuelConsumption* B)
Start
Meet met één Flow sensor (true)
of met twee flow sensoren (false)
/*Meet 1x in 100ms*/
Actueel brandstof in verbruik l/h is = gemeten brandstof verbruik met flowmeter inflow
Totaal brandstofverbruik += actueel brandstofverbruik/36000
End
/*Meet 1x in 100ms*/
Actueel brandstof in verbruik l/h is = gemeten brandstof verbruik met flow sensor inflow – flow sensor outflow.
Totaal brandstofverbruik += actueel brandstofverbruik/36000
End


_SendDataFuelConsumption(
struct emFuelConsumption* B)
start
Pointer naar CM InternComSend gedefinieerd?
Toestemming om te zenden?
Converteer data struct van emFuelConsumption naar een array van characters
_SendDataInternComSend(
adress van CM InternComSend ,
te versturen array, array lengte)
Einde van functie


_UpdateFuelConsumption(
struct emFuelConsumption* B)
Start
_SendDataFuelConsumption(B);
_ProcessFuelConsumption(B);
end

## 14.4 Bijlage vooranalyse

2016

# Vooranalyse

**Afstudeerstage VMB Automation**

**10059490**
**JORIS GROEN**
**25 AUGUSTUS 2016**

**HAAGSE HOGENSCHOOL | VMB besturingstechniek**

# Inhoud

1 Inleiding ..... 1
1.1 J1939 onderzoek (motor management) ..... 2
1.1.1 Beschikbare systemen ..... 3
1.1.2 Analoge uitgangen ..... 3
1.1.3 J1939 ..... 4
1.1.3.1 CAN log opstelling ..... 5
1.1.3.2 De CAN log ..... 6
1.1.3.3 Conclusie J1939 ..... 9
1.1.4 Conclusie motor management module ..... 10
1.2 Fuel management systeem Flowmeter onderzoek ..... 11
1.2.1 Opbouw en eisen ..... 11
1.2.2 Het onderzoek ..... 12
1.2.2.1 Soorten flowmeters ..... 12
1.2.2.2 Informatie terugkoppeling ..... 12
1.2.2.3 Voeding ..... 12

# 1 Inleiding

VMB gaat i.s.m. een ander bedrijf, dat gespecialiseerd is in dieselmotoren, een Ship Monitoring en Management System vanaf hier SMMS te noemen ontwikkelen. Het SMMS systeem zal voorlig bestaan een:

- Brandstof module
- Motormanagement module

Hier kunnen later naar behoefte modules aan toegevoegd worden

De brandstofmodule moet het actuele brandstofverbruik meten en doorgeven aan het de standaard module. De emissie module houdt bij wat de uitstoot is van de diesel motor. Dit kan de $CO_2$ uitstoot zijn maar het kan ook de $NO_3$ of $SO_2$ uitstoot zijn. De motormanagement module leest de gegevens van de motor uit. Dit gaat met behulp van een databus. Op deze databus zit de ECU[1] die de gegevens van de motor uit leest en op de databus zet.

Om te voorkomen dat er pas tijdens de ontwikkeling van een of meerdere modules blijkt dat een of meerdere modules technische gezien niet praktische haalbaar zijn wordt vooraf een vooronderzoek gedaan. Tijdens dit onderzoek zal bekeken worden welke onderdelen van het deel systeem essentieel zijn voor de werking en of deze onderdelen ontworpen en gerealiseerd kunnen worden. Het implementeren zelf gebeurd niet tijdens het onderzoek, er wordt alleen gekeken of en welke mogelijkheden er zijn om de data te implementeren. Op basis van de gevonden mogelijkheden wordt vervolgens geconcludeerd of het voor VMB mogelijk is om tijdens de ontwerpfase de data te implementeren. Mocht dit niet het geval zijn dan kan er geconcludeerd worden dat het ontwerpen en realiseren van de navigatiemodule (gedeeltelijk) niet mogelijk is. Alle onderzoeken en resultaten worden opgenomen in het onderzoeksrapport.

[1] De ECU is de engine control unit. De ECU maakt onderdeel uit van de moderne besturingssystemen in auto's, motoren of boten. De ECU verzorgt de besturing en uitlezen van gegevens van de motor en houd bij wat bijvoorbeeld de temperatuur van het koelwater is maar ook de snelheid waarmee de motor draait. Voor meer info: https://en.wikipedia.org/wiki/Engine_control_unit

## 1.1 J1939 onderzoek (motor management)

In dit onderzoek zal de technische haalbaarheid van de motormanagement module onderzocht worden. De motormanagement module toont de actuele motorgegevens en bewaard deze in een logbestand. Op deze manier is terug te kijken hoe de motor zich gedroeg tijdens een dag varen. Voor dit systeem is de informatie voorziening vanuit de motor erg belangrijk. Zonder informatie van de motor zijn er geen gegevens om te loggen en is de motormanagement module waardeloos. Een controller verzorgt het verzamelen van alle motor informatie en stuurt deze informatie naar een beeldscherm van het SMMS in de stuurhut waar de informatie getoond wordt.

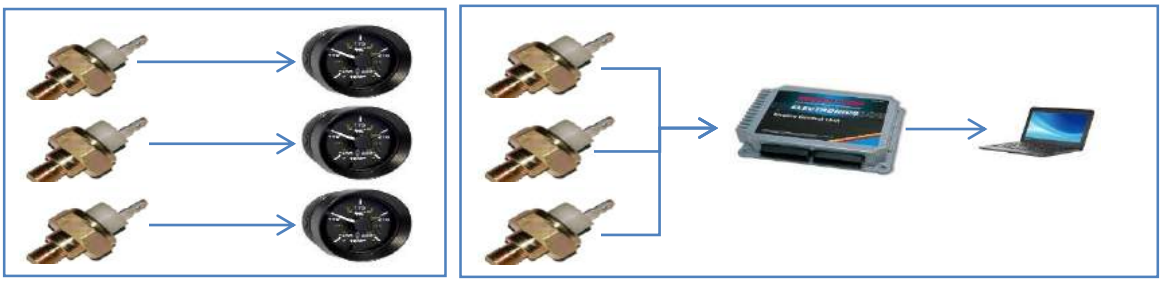
Als eerste zal de stagiair onderzoeken welke systemen er zijn die de motorkarakteristieken meten en verzenden. Vervolgens wordt er gekeken welke informatie van de motor wordt verzonden door deze systemen en welke informatie daarvan nuttig is voor de motormanagement module. Als laatst wordt op basis van de beschikbare nuttige motorgegevens bekeken of de motormanagement module een nuttige toevoeging kan zijn voor het SMMS.

Volgens de partner waarmee VMB het SMMS ontwerpt moet altijd de temperatuur van de koelvloeistof, de oliedruk en het toerental gemeten worden. Dit zijn de basisgegevens van de motor die interessant zijn voor de kapitein en het onderhoudspersoneel aan boord van het schip.

### 1.1.1 Beschikbare systemen

Voor het verkrijgen van informatie over de motor zitten er sensoren op de motor die bijvoorbeeld de koelwatertemperatuur of het toerental meten. Voor het verzamelen en verzenden van deze informatie zijn er twee systemen.

Bij oudere motoren worden de uitgangswaarden van de sensoren direct doorgegeven naar een metertje op de motor zelf. De waardes van de sensoren worden direct op de meters getoond. Dit systeem wordt veel toegepast bij motoren geproduceerd voor 1995 toegepast.



**Figuur 1.1: Opzet meetsystemen**

Bij modernere motoren wordt er gebruik gemaakt van een ECU (Engine Controll Unit). De ECU is samen met andere elektronische componenten verantwoordelijk voor het verwerken van besturingssignalen van gebruiker voor de auto, motor of het schip. De ECU verzend over een data bus berichten naar bijvoorbeeld besturingscomponenten voor de remmen of de automaat. Deze berichten worden verzonden over een CAN netwerk. Bij motorbesturingen voor auto, motoren en schepen wordt tegenwoordig veel gebruik gemaakt van het J1939 protocol[1].

Door met een controller bij oude motoren de waardes van de sensoren af te tappen en bij moderne motoren de j1939 berichten te bekijken kan het motormanagement system informatie vergaren over de motors op een schip.

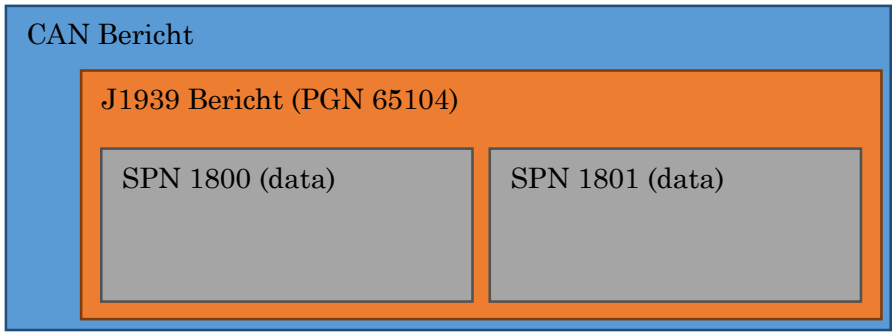
### 1.1.2 Analoge uitgangen

Voor meten van de temperatuur van de koelvloeistof, de oliedruk en het toerental zitten er altijd al sensoren op de motor. Mochten de sensoren niet geschikt kunnen ze altijd vervangen worden door sensoren van andere bedrijven. Een bedrijf genaamd Murphy heeft een module op de markt gebracht waarmee de analoge meetwaardes vertaald worden door en verzonden kunnen worden door de MECAN module van Murphy en d.m.v. een J1939 protocol[2]. Op deze manier kan het uitlezen van deze waardes op de zelfde manier gebeuren als bij het normale J1939 protocol. Later kan altijd nog per motor bekeken worden of er nog extra sensoren op de motor zitten. Die motorgegevens meten waarvan het handig is als de waardes van de deze meters worden doorgegeven aan het SMMS systeem.

[1] https://en.wikipedia.org/wiki/SAE_J1939 (application)
[2] http://www.fwmurphy.com/products/can-io-accessories/mecan

### 1.1.3 J1939

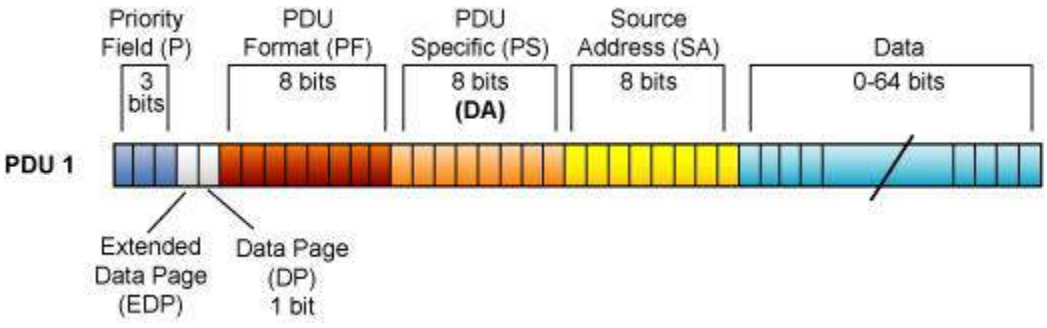
Het J1939 protocol is het berichtformaat waarin de gegevens van de motor worden verstuurd. Het J1939 protocol wordt verpakt in een CAN bericht[1] en wordt verstuurd over een seriële verbinding. Ieder J1939 bericht heeft een PGN (Parameter Groep Numbers) de PGN geeft aan welk bericht er verstuurd/ontvangen is bijvoorbeeld PGN 65104 staat voor de batterij (accu) temperatuur. Iedere J1939 bericht bestaat uit een aantal SPN (Suspect Parameter Number) de PGN 65104 bestaat uit SPN 1800 en 1801 deze SPN staan voor de batterij temperatuur 1 en voor batterij temperatuur 2.




**Figuur 1.2: Data inhoud CAN met J1939**

**Frame opbouw CAN(J1939)**

Zoals te zien is in de onderstaande afbeelding wordt de CAN identifier in het CAN frame gebruikt om het PGN nummer (PF en PS) van het J1939 bericht vast te leggen. In het 8 byte (64 bit) brede dataveld van het CAN frame. In het source adress wordt aangegeven van welke module het bericht verstuurd is. De priority geeft aan hoe belangrijk het bericht is. Berichten met een hogere prioriteit hebben voorrang op berichten die verzonden worden met een lagere prioriteit. Je hebt twee manieren soorten J1939 berichten die naar een specifiek adres gestuurd worden en berichten die naar een groep modules verstuurd wordt. De manier is afhankelijk van het PGN nummer van het bericht  Bij Broadcast berichten is de PF van het PGN nummer tussen de 240 en 255. Het bericht wordt verstuurd naar een groep apparaten. De groep wordt bepaald door de PS van het PGN nummer. Als de PF lager is dan 240 dan wordt het bericht verstuur naar een specifieke module. De PS is het adres waarnaar het J1939 bericht verzonden.

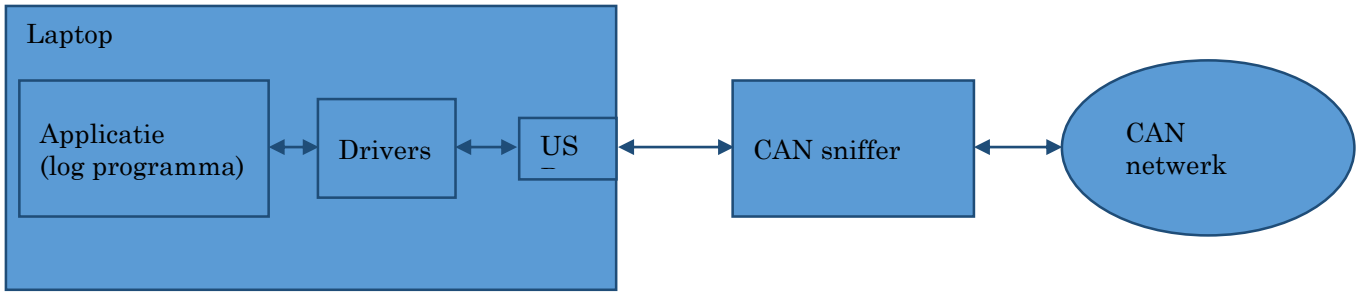



**Figuur 1.3:Frameopbouw CAN met J1939**

[1] Voor meer informatie over CAN zie: https://en.wikipedia.org/wiki/CAN_bus

**Toepassing**
Het J1939 protocol wordt voor veel meer dingen gebruikt dan alleen de motorbesturing. Er zijn berichten voor het bijhouden van de brandstofuitstoot tot aan het aanzetten van een bezet lichtje op een taxi. Doordat er veel soorten berichten zijn is het niet duidelijk welke berichten er door de ECU van de motor verzonden worden. Daarom is er een CAN log gedaan op een dieselmotor van een generator set. Op deze manier kan er bekeken worden welke PGN er verzonden worden en welke SPN's van de PGN's data bevatten. Dit kan per motor verschillen. Aangezien er maar van een motor een CAN log gemaakt kan worden, is het is niet zeker dat alle gegevens die bij die motor gemeten worden ook bij andere motoren gemeten worden. Er zal later bij het implementeren van het SMMS per motor gecontroleerd moeten worden welke gegevens uitgestuurd worden. Sommige PGN nummers zijn niet opgenomen in de J1939 standaard. Dit geeft bedrijven de ruimte om eigen J1939 berichten te maken. Nadeel hiervan is dat als je geen beschrijving van die eigen gemaakte berichten hebt mogelijk sommige data van motoren onbekend blijft.

#### *1.1.3.1 CAN log opstelling*
Voor het J1939 onderzoek moest er een CAN log gedaan worden. Bij een CAN log kijk je met een CAN sniffer welke berichten er over een CAN bus verstuurd worden. Met behulp van een software programma kun je al deze berichten in een bestandje zetten. De opstelling die de stagiair moest gebruiken zag er als volgt uit




**Figuur 1.4: Meetopstelling**

De seriële connector van de CAN logger wordt aangesloten op een canbus. De USB connector wordt aangesloten op een laptop. Nadat de drivers geïnstalleerd zijn op de laptop kan een programma gebruikt worden waar je in kan stellen wat de datasnelheid op de CAN bus is. Dit is meestal 125kb, 250kb, 500kb of 1Mb per seconde. Hoe sneller de data snelheid hoe korter de fysieke lengte van de CAN bus mag zijn. Als er op de CAN bus een bericht verstuurd word met 250kb per seconde dan kan een goed ingestelde CAN logger dit bericht opvangen en loggen in een bestandje. De CAN logger veranderd niks aan het bericht dat gewoon aankomt bij het juiste ontvangst adres van het bericht. Na eerst de CAN logger getest te hebben kon de stagiair met deze opstelling een CAN log maken van de J1939 dataoutput van een ECU van een dieselmotor. Met behulp van deze CAN log kan voor het vooronderzoek bepaald worden welke J1939 berichten er verstuurd worden door de ECU. Als dit bekend is kan bepaald worden welke gegevens ingepakt in J1939 berichten interessant zijn voor het motor management systeem en of er nog meer berichten zijn die interessant zijn voor het SMMS.

#### 1.1.3.2 De CAN log

Bij de CAN log heeft de motor eerst stationair gedraaid. Daarna heeft de motor even met belasting (generator) gedraaid.

Uit de CAN log kwamen de volgende berichten met bekende PGN nummers.

| PGN | Betekenis |
|---|---|
| 61444 | Electronic Engine Controller 1 |
| 65247 | Electronic Engine Controller 3 |
| 65262 | Engine Temperature 1 |
| 65263 | Engine Fluid Level/Pressure 1 |
| 65266 | Fuel Economy (Liquid) |
| 65269 | Ambient Conditions |
| 65270 | Inlet/Exhaust Conditions 1 |
| 65271 | Vehicle Electrical Power 1 |

**Tabel 1.1: Gelogde J1939 berichten**

Zoals eerder beschreven zitten in de PGN berichten data opgeslagen waarnaar gerefereerd wordt m.b.v. SPN. Als sommige SPN in een PGN bericht continue op hun hoogste waarde staan betekent dit dat er geen bruikbare data aanwezig was voor die SPN.

**Electronic Engine Controller 1**
Deze PGN bevat de volgende SPN

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 899 | Engine Torque Mode | nee |
| 4154 | Actual Engine - Percent Torque High Resolution | nee |
| 512 | Driver's Demand Engine - Percent Torque | nee |
| 513 | Actual Engine - Percent Torque | ja |
| 190 | Engine Speed | ja |
| 1483 | Source Address of Controlling Device for Engine Control | nee |
| 1675 | Engine Starter Mode | nee |
| 2432 | Engine Demand – Percent Torque | nee |

**Tabel 1.2: Data collectie PGN bericht**

De meeste SPN zijn niet bruikbaar alleen de motorsnelheid en het koppel van de motor worden gemeten. In de bijlage staan twee grafieken met daarin het snelheid verloop in rpm en het actuele motor koppel ten opzichte van het nominaal koppel van de motor.

**Electronic Engine Controller 3**
Deze PGN bevat de volgende SPN

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 514 | Nominal Friction - Percent Torque | nee |
| 515 | Engine's Desired Operating Speed | nee |
| 519 | Engine's Desired Operating Speed Asymmetry Adjustment | nee |
| 2978 | Estimated Engine Parasitic Losses - Percent Torque | ja |
| 3236 | Aftertreatment 1 Exhaust Gas Mass Flow | ja |
| 3237 | Aftertreatment 1 Intake Dew Point | nee |
| 3238 | Aftertreatment 1 Exhaust Dew Point | nee |
| 3239 | Aftertreatment 2 Intake Dew Point | nee |
| 3240 | Aftertreatment 2 Exhaust Dew Point | nee |

**Tabel 1.3: Data collectie PGN bericht**

In dit PGN bericht zijn twee SPN bruikbaar de 2978 en de 3236. SPN 2978 bevat data over het vermogen (van de motor) dat gebruikt word voor het aandrijven van koelventilator of airconditioning. De data wordt uitgedrukt in procenten berekend door: $koppelverlies\ (\%) = \frac{koppel\ nodig\ voor\ koelventilator\ e.d.}{nominaal\ koppel\ van\ de\ motor}$

Daarnaast geeft SPN 3236 weer hoeveel uitlaatgassen de uitlaat uitstromen. In de bijlage staan de gemeten waarden tijdens de CAN log.

**Engine Temperature 1**
Deze PGN bevat de volgende SPN

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 110 | Engine Coolant Temperature | ja |
| 174 | Engine Fuel Temperature 1 | ja |
| 175 | Engine Oil Temperature 1 | nee |
| 176 | Engine Turbocharger Oil Temperature | nee |
| 52 | Engine Intercooler Temperature | nee |
| 1134 | Engine Intercooler Thermostat Opening | nee |

**Tabel 1.4: Data collectie PGN bericht**

Deze PGN heeft data van SPN 110 en 174, zij geven informatie over de temperatuur van de koelvloeistof en de diesel. De temperatuur van de koelvloeistof is een van de parameters die in ieder geval gemeten moet worden. In de bijlage staan de gemeten temperaturen van de koelvloeistof en diesel.

**Engine Fluid Level Pressure 1**
Deze PGN bevat de volgende SPN

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 94 | Engine Fuel Delivery Pressure | ja |
| 22 | Engine Extended Crankcase Blow-by Pressure | nee |
| 98 | Engine Oil Level | nee |
| 100 | Engine Oil Pressure | ja |
| 101 | Engine Crankcase Pressure | nee |
| 109 | Engine Coolant Pressure | nee |
| 111 | Engine Coolant Level | nee |

**Tabel 1.5: Data collectie PGN bericht**

Deze PGN heeft data van SPN 94 en 100. De spn bevatten informatie over de druk van de diesel op het moment dat het de motor in stroomt en bevat de oliedruk in de motor. De oliedruk in de motor is een van de parameters die gemeten moet worden. In de bijlagen staan twee grafieken met de gemeten olie en diesel druk.

**Fuel Economy (Liquid)**
Deze PGN bevat de volgende informatie.

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 183 | Engine Fuel Rate | ja |
| 184 | Engine Instantaneous Fuel Economy | nee |
| 185 | Engine Average Fuel Economy | nee |
| 51 | Engine Throttle Position | nee |
| 3673 | Engine Throttle 2 Position | nee |

**Tabel 1.6: Data collectie PGN bericht**

Deze PGN heeft data van SPN  183. Deze SPN bevat dat over het dieselverbruik van de motor. Het dieselgebruik word gemeten in l/h. In de bijlagen staat het dieselverbruik gemeten tijdens de CAN log.

**Ambient Conditions**
Deze PGN bevat de volgende informatie

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 108 | Barometric Pressure | ja |
| 170 | Cab Interior Temperature | nee |
| 171 | Ambient Air Temperature | nee |
| 172 | Engine Air Inlet Temperature | ja |
| 79 | Road Surface Temperature | nee |

Tabel 1.7: Data collectie PGN bericht

Deze PGN heeft data van SPN 108 en 172. Deze SPN bevat informatie over de luchtdruk buiten de motor en de luchttemperatuur buiten de motor. De gemeten temperatuur en druk tijdens de CAN log staat in de bijlagen.

**Inlet Exhaust Conditions 1**
Deze PGN bevat de volgende informatie

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 81 | Engine Diesel Particulate Filter Inlet Pressure | nee |
| 102 | Engine Intake Manifold #1 Pressure | ja |
| 105 | Engine Intake Manifold 1 Temperature | ja |
| 106 | Engine Air Inlet Pressure | nee |
| 107 | Engine Air Filter 1 Differential Pressure | nee |
| 173 | Engine Exhaust Gas Temperature | nee |
| 112 | Engine Coolant Filter Differential Pressure | nee |

Tabel 1.8: Data collectie PGN bericht

Deze PGN heeft data van SPN 102 en 105. Deze SPN bevatten informatie over de lucht die de motor naar binnen zuigt voor de lucht/dieselmix die nodig is voor het aandrijven van de motor. De gemeten waarden staan in de bijlagen.

**Vehicle Electrical Power 1**
Deze PGN bevat de volgende informatie.

| SPN | Betekenis | Bruikbare data |
|---|---|---|
| 114 | Net Battery Current | nee |
| 115 | Alternator Current | nee |
| 167 | Charging System Potential (Voltage) | nee |
| 168 | Battery Potential / Power Input 1 | nee |
| 158 | Keyswitch Battery Potential | ja |

Tabel 1.9: Data collectie PGN bericht

Deze PGN heeft data van SPN 158. SPN 158 bevat data over de voedingsspanning van de ACCU. De gemeten spanning staat hier in de bijlagen.

#### *1.1.3.3 Conclusie J1939*

In Tabel 1.10 op pagina 10 staat een overzicht van alle SPN die gemeten konden worden tijdens de CAN log. De meeste SPN data is bruikbaar voor de motormanagement module. Met deze informatie kan informatie getoond worden over de status van de motor. Met SPN 513 kan gezien worden of de motor niet te veel moeite moet doen om de motor te laten draaien. Een plotselinge piek in de koppel kan betekend dat er een grotere belasting aan de motor hangt (rommel aan de schroef). SPN 190, SPN 110, en

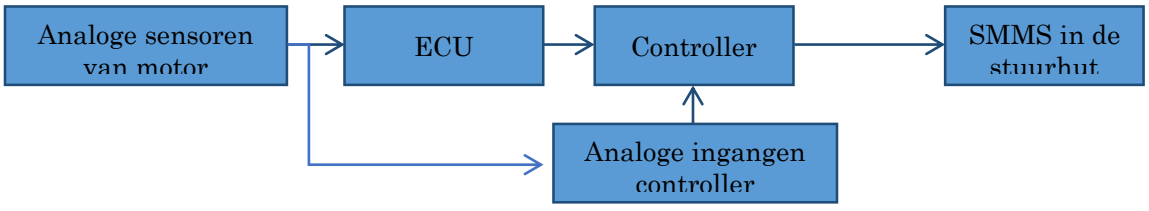
SPN 100 bevatten de informatie die volgens de partner van VMB in ieder geval weergegeven moet worden. De data SPN 3236 bevat informatie over de hoeveel uitlaatgassen die de motor uitstromen. Dit kan een aanvulling zijn op een emissie meetsysteem. SPN 174 en 94 geven informatie over de diesel temperatuur en druk. Als de druk wegvalt krijgt de motor geen diesel meer. Dit kan betekenen dat de brandstofpomp stuk is of dat er een aanvoerbuis geblokkeerd is. SPN 183 kan een grote aanvulling zijn op het Fuel management Systeem. Als blijkt dat de meter die de brandstofverbruik van de motor meet precies genoeg is kan die zelfs de flowmeters voor het meten van het brandstofverbruik vervangen.

| SPN | Betekenis | Bruikbare data | Bruikbaar voor SMMS |
|---|---|---|---|
| 513 | Actual Engine - Percent Torque | ja | Motor management |
| 190 | Engine Speed | ja | Motor management |
| 2978 | Estimated Engine Parasitic Losses - Percent Torque | ja | - |
| 3236 | Aftertreatment 1 Exhaust Gas Mass Flow | ja | Emissie |
| 110 | Engine Coolant Temperature | ja | Motor management |
| 174 | Engine Fuel Temperature 1 | ja | Fuel management |
| 94 | Engine Fuel Delivery Pressure | ja | Fuel management |
| 100 | Engine Oil Pressure | ja | Motor management |
| 183 | Engine Fuel Rate | ja | Fuel management |
| 108 | Barometric Pressure | ja | - |
| 172 | Engine Air Inlet Temperature | ja | - |
| 102 | Engine Intake Manifold #1 Pressure | ja | - |
| 105 | Engine Intake Manifold 1 Temperature | ja | - |
| 158 | Keyswitch Battery Potential | ja | - |

**Tabel 1.10: Bruikbare data**

### 1.1.4 Conclusie motor management module

De motormanagement module kan genoeg informatie verkrijgen om nuttige informatie aan de schipper te kunnen weergeven. Met behulp van analoge sensoren kan de motormanagement module direct van oude motoren de minimale informatie aflezen en weergeven. Als de motor bestuurd wordt door een ECU zijn de mogelijkheden velen malen groter en zal het motormanagement systeem zeker een toevoeging zijn voor het SMMS.




**Figuur 1.5: Opzet motormanagement module**

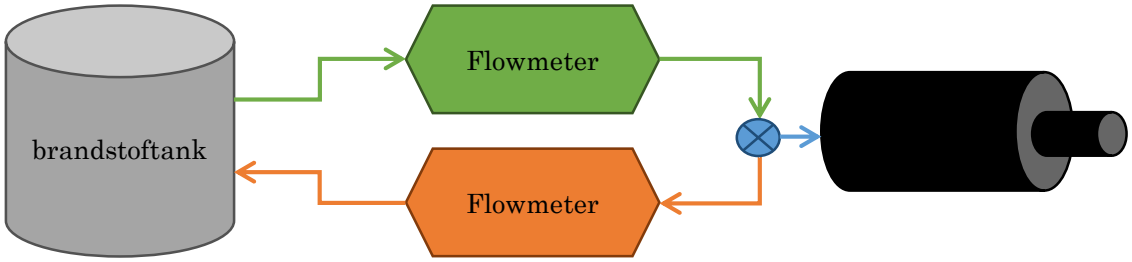
## 1.2 Fuel management systeem Flowmeter onderzoek

Op een binnenvaartschip zit meestal al een systeem die het totale brandstofverbruik bijhoudt. Dit brandstofsysteem wordt ontwikkeld zodat het actueel brandstofverbruik per motor bekeken kan worden. Op deze manier kan i.c.m. de motormanagement module de meest efficiënte vaarsnelheid (toerental van de motor) van het schipbepaald worden. dit word gedaan door bij elke motorsnelheid het brandstofverbruik te meten. Na de meting weet het SMMS het efficiëntste toerental.

Maar dit is voor de meeste brandstofsystemen (auto, motor e.d.) niet nodig. Hun motoren zijn voorzien van moderne technologie en dus erg zuinig. Daarnaast komt in de autowereld het hybride motorsysteem opzetten. Hierdoor bespaard de bestuurder veel brandstof. In de scheepvaart ligt dit anders. Schepen gaan erg lang mee de meeste ongeveer 30 jaar. Daarnaast is de scheepvaarsector erg conservatief ingesteld nieuwe systemen worden maar langzaam ingevoerd. De rede hiervoor is dat in het ergste geval voor 30 jaar vast zit aan een niet of slecht werkend systeem. Hierdoor varen de meeste schepen met oude maar betrouwbare motorsystemen. Dit brandstofsysteem zorgt er voor dat met de oude motoren zo zuinigmogelijk gevaren kan worden. Door de prijs van dit systeem laag te houden is het voor de schipper mogelijk om zuiniger en uiteindelijk ook goedkoper te varen.

### 1.2.1 Opbouw en eisen

Voor het meten van het brandstofverbruik van motoren worden meestal flowsensoren gebruikt. In het geval van een schip zal dit om grote flowsensoren gaan aangezien een dieselmotor van een schip veel diesel gebruikt. Een speciaal technische aspect aan de dieselmotor t.o.v. een benzine motor is dat een dieselmotor ook diesel terug voert naar de brandstoftank. Voor iedere dieselmotor in het schip moeten twee flowsensoren geïnstalleerd worden. Een voor het meten van de aanvoer van diesel en een voor het meten van de afvoer van diesel terug naar de brandstoftank. Het verschil tussen de gemeten waarden bepaald het dieselverbruik.




**Figuur 1.6: Opzet brandstof meting per motor**

De flowsensoren moeten bij voorkeur op een 230V wisselspanning of 24V gelijkspanning werken. De flowsensoren hoeven alleen de flowsnelheid van de diesel te meten andere gegevens als temperatuur is een leuke extra, maar niet nodig. De flowsensor moet zijn gegevens via een analoge uitgang (4mA tot 20mA) of een RS485 databusverbinding kunnen verzenden naar het SMMS. De accuraatheid van de meters is niet de belangrijkste parameter een afwijking onder de 1 a 2 procent is genoeg. De flowmeters moeten ten minste tot 3000l/h kunnen meten. Belangrijker zijn de prijs van de flowmeter en de installatie/onderhoudsvriendelijkheid. De flowmeter moet geschikt zijn om de flowsnelheid van het medium diesel te kunnen meten en moet geschikt zijn om geïnstalleerd te kunnen worden op een binnenvaartschip.

### 1.2.2 Het onderzoek

Als eersten zal wat algemene informatie opgezocht moeten worden over flowsensoren. Welke soorten flowsensoren zijn er en hoe werken ze. Er zal gezocht moeten worden naar een aantal leveranciers van flowsensoren voor het meten van het diesel verbruik van een motor. Daarna zal bepaald moeten worden welke flowsensor productlijnen van deze leveranciers geschikt zijn om te plaatsen op het schip. Er is steekproefgewijs bij verschillende leveranciers een offerte opvraag gedaan. De resultaten zie je in Tabel 1.11, Tabel 1.12 en Tabel 1.13.

#### *1.2.2.1 Soorten flowmeters*

Er zijn veel verschillende flowmeters. Hieronder worden de soorten goedkopere flowsensorsoorten besproken die de leveranciers leveren.

**Schoepenmeter**

Dit is de goedkoopste sensor soort die er te vinden is. Deze sensor werkt met het zelfde principe als het oude schoepenrad. Het rad gaat draaien doordat een stromende vloeistof tegen de schoepen duwt. Hoe sneller de vloeistof stroomt hoe sneller het schoepenrad gaat draaien. De sensor heef een klein rad ingebouwd in de behuizing en werkt daardoor met het zelfde principe

#### *1.2.2.2 Informatie terugkoppeling*

Belangrijk voor uitlezen van de flowmeter is op welke manier de gemeten flowrate teruggekoppeld wordt naar het systeem. Duurdere flowmeters willen hiervoor nog wel een modbus gebruiken. Bij het uizoeken van de goedkopere flowmeters werden vooral analoge signalen gebruikt en puls signalen. De controller zal dus de flowmeters moeten kunnen uitlezen m.b.v. een analoog of een puls signaal.

#### *1.2.2.3 Voeding*

De meeste flow sensoren kunnen werken op 24VDC

| | Markt technical | | Nexon Elektronic | | |
|---|---|---|---|---|---|
| serie | ZHA | ZHA | FTB200 | FTB200 | FTB200 |
| prijs(€) (meter) | 792 | 792 | 580 | 619 | 619 |
| prijs(€) (output) | 266 | 413 | 0 | 0 | 0 |
| prijs(€) (extra)(connectors ect.) | 26 | 26 | ? | ? | ? |
| systeemprijs (2x flowmeter + evt. converter) | 2168 | 2462 | 1160 | 1238 | 1238 |
| | | | | | |
| Pulse per liter | 3480 | - | ? | - | - |
| Max freq (Hz) | 1450 | - | ? | - | - |
| Min flow (l/h) | 30 | 30 | 96 | 96 | 96 |
| Max flow (l/h) | 1500 | 1500 | 600 | 600 | 600 |
| max druk (bar) | 315 | 315 | 400 | 400 | 400 |
| Line size | G1/4" | G1/4" | DN6 | DN6 | DN6 |
| Line connection | draad | draad | draad(NPT) | draad | draad |
| min. medium temp (°c) | -20 | -20 | -40 | -40 | -40 |
| max. medium temp (°c) | 150 | 150 | 180 | 180 | 180 |
| min. Omgevings temp (°c) | -10 | -10 | -40 | -40 | -40 |
| max. Omgevings temp (°c) | 60 | 60 | 85 | 85 | 85 |
| Output | | | | | |
| Pulse | ja | nee | ja | nee | nee |
| Analoog | nee | ja | nee | ja (4-20mA) | ja (0-10V) |
| temperatuur compensatie | nee | nee | nee | nee | nee |

Tabel 1.11

| | GHM meettechbiek | Piusi | EDGE(WPA) |
|---|---|---|---|
| serie | VHZ | K400 | |
| prijs(€) (meter) | 975,6 | 256 | 380 |
| prijs(€) (output) | 0 | 0 | 0 |
| prijs(€) (extra)(connectors ect.) | ? | 0 | 0 |
| systeemprijs (2x flowmeter + evt. converter) | 1951,2 | 512 | 380 |
| | | | |
| Pulse per liter | ? | 100 | ? |
| Max freq (Hz) | ? | 50 | ? |
| Min flow (l/h) | 30 | 60 | 2 |
| Max flow (l/h) | 3000 | 1800 | 330 |
| max druk (bar) | 200 | 70 | |
| Line size | G3/4" | 1/2" | |
| Line connection | draad | draad | |
| min. medium temp (°c) | -25 | -20 | 10 |
| max. medium temp (°c) | 85(120) | 70 | 120 |
| min. Omgevings temp (°c) | -20 | -10 | -20 |
| max. Omgevings temp (°c) | 70 | 60 | 60 |
| Output | | | |
| Pulse | ja | ja | ja |
| Analoog | nee | nee | ja |
| temperatuur compensatie | nee | nee | ? |

Tabel 1.12

| | maretron | | | Tuthill | | |
|---|---|---|---|---|---|---|
| serie | M2AR | M4AR | M8AR | TM03 | TM04a | TM06a |
| prijs(€) (meter) | 396 | 556 | 716 | 595 | 595 | 0 |
| prijs(€) (output) | 320 | 320 | 320 | | 0 | 0 |
| prijs(€) (extra)(connectors ect.) | 0 | 0 | 0 | | 0 | 0 |
| systeemprijs (2x flowmeter + evt. converter) | 1112 | 1432 | 1752 | 1190 | 1190 | 0 |
| | | | | | | |
| Pulse per liter | 400 | 112 | 52 | | 53 | |
| Max freq (Hz) | 55,56 | 46,67 | 60,67 | | 32,68 | |
| Min flow (l/h) | 25 | 180 | 480 | 60 | 60 | |
| Max flow (l/h) | 500 | 1500 | 4200 | 660 | 2220 | 4560 |
| max druk (bar) | 68,95 | 137 | 137 | | 80 | |
| Line size | 1/4" | 1/2" | 3/4" | 3/8" | 1/2" | 3/4" |
| Line connection | draad | draad | draad NPT | draad | draad | draad |
| min. medium temp (°c) | -20 | -20 | -20 | -40 | -40 | |
| max. medium temp (°c) | 80 | 80 | 80 | 150 | 150 | |
| min. Omgevings temp (°c) | -40 | -10 | -40 | - | - | |
| max. Omgevings temp (°c) | 80 | 80 | 80 | - | - | |
| Output | | | NMEA | | | |
| Pulse | ja | ja | encoder | ja | ja | |
| Analoog | nee | nee | nee | nee | nee | |
| temperatuur compensatie | nee | nee | nee | nee | nee | |

Tabel 1.13

## 14.5 Bijlage Wireless Communication onderzoek

2016

# Wireless Communication onderzoek

**Afstudeerstage VMB Automation**

**10059490**
**JORIS GROEN**
**25 AUGUSTUS 2016**

HAAGSE HOGENSCHOOL | VMB besturingstechniek

# 1 Inleiding

Het SMMS is verdeeld over meerdere ruimtes in een schip. De deelinstallaties moeten met elkaar kunnen communiceren. Doordat de schepen vaak met waterdichte compartimenten werken is het boren van gaten tussen deze compartimenten meestal lastig of niet toegestaan. Verder worden de volgende eisen gesteld aan de communicatie hardware:

- Makkelijk te instaleren en onderhouden
- Het boren van gaten voor het aanleggen van de installatie zo veel mogelijk vermijden
- Schaalbaar: de communicatie installatie moet makkelijk uitgebreid kunnen worden voor grotere installaties.
- Gegarandeerd: Het netwerk moet in de meeste omstandigheden kunnen blijven functioneren.
- Veilig: personen moeten geen verbinding kunnen maken met dit netwerk. Dit netwerk is alleen bedoeld voor de communicatie tussen de verschillende onderdelen van het SMMS
- Betrouwbaar: het mag niet mogelijk zijn om kustmatig gegenereerde berichten te verzenden op het netwerk. Bijvoorbeeld berichten versturen om opgeslagen statistieken te beïnvloeden (b.v. gemiddeld toerental van de motor e.d.).
- Moet in ieder geval tot 8 controller met elkaar verbinden en onderling laten communiceren.

# 2 Opzet netwerk

In dit hoofdstuk wordt de opzet van het netwerk beschreven. Er zullen keuzes gemaakt worden met betrekking tot de te gebruiken databus, bedraad  of onbedraad netwerk en veiligheidsprotocol.

## 2.1 Databus

Er zijn een aantal databussen beschikbaar op de plc's waarmee de plc's onderling kunnen communiceren

- RS232
- RS485
- CAN
- Ethernet

**RS232**

RS232 is een data bus voor communicatie tussen twee apparaten. Officieel kunnen er maximaal twee apparaten aangesloten worden op de databus. De afstand waarmee de RS232 nog kan blijven werken is bij gebruik van standaardkabels beperkt tot 15 meter. Met speciale kabels kunnen lengtes van 300 meter bereikt worden.

**RS485**
RS485 is een databus voor communicatie tussen twee of meer apparaten. Afhankelijk van de gewenste snelheid kan de RS485 gebuikt worden tot een afstand van maximaal 1200m bij een snelheid van 100 kbit/s (10 meter bij 35Mbit/s). RS485 wordt mogelijk gebruikt om sensoren m.b.v. een modbus protocol uit te lezen.

**CAN**
CAN staat voor Controlled Area Network. CAN werd oorspronkelijk alleen in voertuigen gebruikt maar wordt nu ook toegepast in de industrie. CAN kan afstanden van 40 tot 1000 meter overbruggen bij data snelheden van respectievelijk 1Mbit/s tot 50kbit/s. De CAN bus wordt al gebruikt om J1939 data van de motoren uit te lezen.

**Ethernet**
De ethernet bus op de plc wordt gebruikt om de plc te programmeren. Daarnaast kan de ethernet bus gebruikt worden voor het verbinden van plc's voor onderlinge communicatie. De ethernet bus op de plc kan tot 16 verbindingen tegelijkertijd realiseren. Deze 16 tegelijkertijd gebruikte verbindingen kunnen i.t.t. de andere databussen voor verschillende doeleinde gebruikt worden bijvoorbeeld:

- 8 TCP verbindingen voor onderlinge communicatie
- 1 verbinding voor internet communicatie
- 2 UDP verbindingen om een overige toepassingen
- 5 nog niet gebruikte verbindingen

## 2.2 Draadloos of bedraad
Alle besproken databussen kunnen standaard in bedrade vorm gebruikt worden. Met behulp van extra apparatuur kunnen alle besproken databussen ook in onbedrade vorm gebruikt worden. De onbedrade vorm maakt de installatie makkelijker te instaleren maar brengt wel extra kosten met zich mee. De extra apparatuur die aangeschaft wordt moet tegen trillingen en temperaturen op het schip kunnen. De extrakosten worden enigszins gecompenseerd doordat er minder geld nodig is voor kabels. Hierdoor is deze apparatuur vrij prijzig. Voor de onbedrade vorm moeten de antennes waarschijnlijk op het dek van het schip geplaatst worden om een stabiele draadloze verbinding te garanderen. Toch zal de bedrade vorm meer werk en boorgaten kosten om te realiseren.

## 2.3 Veiligheid
Alle databussen in bedrade vorm hebben geen mogelijkheid om hun databussen te beveiligen. In draadloze vorm heeft ethernet de mogelijkheid om zijn draadloze verbinding te beveiligen en de data geencrypt te versturen. Je hebt een aantal beveiligingsprotocollen voor draadloze communicatie.

**WEP**
Staat voor Wired Equivalent Privacy. Dit is een verouderd en dus onveiliger beveiligings protocol dan het nieuwere WPA(2). Er wordt aangeraden dit protocol alleen te gebruiken als WPA(2) niet beschikbaar is.

**WPA(2)**
Is een nieuwer en veiliger beveiligingsprotocol en lost de grootste beveiligings problemen met het WEP protocol op. WPA(2) is er in personal en eterprise vorm. De personal vorm werkt met vooraf gedeelde sleutels waarmee gebruikers kunnen inlogen op het netwerk. De enterterprice vorm werk met certificaten en een authenticatie server. Iedere gebruiker die een certificaat heeft dat goedgekeurd wordt door de authenticatie server kan gebruikmaken van het netwerk.

## 2.4 Draadloze ethernet modes

Apparaten kunnen d.m.v. ethernet verbindingen draadloos met elkaar verbonden worden m.b.v. WLAN. Deze apparaten kunnen in verschillende modes werken. In Accespoint mode kunnen gebruikers die het WEP of WPA2 wachtwoord hebben inloggen op het wifinetwerk van het accespoint. In repeater modus kan de draadloze dekking van een WLAN uitgebreid worden. Een WLAN module repeat de data die hij binnenkrijgt van een andere WLAN module. In bridgemode is het mogelijk om met behulp van twee WLAN modules twee of meer apparaten draadloos met elkaar te verbinden. Bridge en AP mode kunnen naast elkaar gebruikt worden, maar het is ook mogelijk om alleen bridge mode te gebruiken. Hierdoor kunnen personen niet meer inloggen op het netwerk. Alleen apparaten die bedraad vast zitten aan de twee WLAN modules kunnen gebruik maken van het draadloze netwerk. WLAN modules in bridge mode kunnen meestal geen gebruik maken van het WPA(2) protocol maar wel van het WEP protocol.

# 3 Conclusie

Aangezien de communicatie veilig en betrouwbaar moet zijn is het logische om de ethernet bus te gebruiken i.c.m. de draadloze communicatie waarbij gebruikgemaakt wordt van WLAN modules in brigde mode(zonder AP mode) met het WEP of WPA2 beveiligingsprotocol. Hierdoor is het niet mogelijk om op afstand, bijvoorbeeld vanaf de wal in een haven, de communicatie te beïnvloeden. Het blijft mogelijk om de communicatie te beïnvloeden, maar dit is alleen mogelijk als de “hacker” fysiek op het schip aanwezig. Door de installatie in een afgesloten kasten te plaatsen is het bijna niet om in te breken op het netwerk. Door gebruik te maken van draadloze communicatie wordt het makkelijker om de communicatie installatie te instaleren en wordt de schaalbaarheid van de installatie verbeterd. Het nadeel van draadloze communicatie t.o.v. bedrade communicatie is de verslechterde garantie op een goed functioneren van de communicatie en mogelijk verslechterde beveiliging (WPA2 i.p.v. WEP). De verslechterde garantie op goed functioneren kan grotendeels voorkomen worden door de antennes van de draadloze accespoint te plaatsen op een plek waar beide antennes onder alle omstandigheden “zicht” op elkaar hebben.

Mocht blijken dat de draadloze oplossing te veel extra kosten met zich mee brengt kan altijd nog gekeken worden of de bedrade versie een oplossing is.

# 4 Bibliografie

*CAN Bus*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/CAN_bus

*CAN-AIR/2*. (sd). Opgehaald van ESD: https://esd.eu/sites/default/files/manuals/can-air2_datasheet_en.pdf

*Difference Between Wi-Fi Bridges & Wi-Fi Repeaters*. (sd). Opgehaald van Classroom Synonym: http://classroom.synonym.com/difference-between-wifi-bridges-wifi-repeaters-16865.html

*RS232*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/RS-232

*RS-485*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/RS-485

*What is the Maximum Cable Length For a CAN Bus*. (sd). Opgehaald van National instruments: http://digital.ni.com/public.nsf/allkb/D5DD09186EBBFA128625795A000FC025

*Wi-Fi Protected Access*. (sd). Opgehaald van Wikipedia: https://nl.wikipedia.org/wiki/Wi-Fi_Protected_Access

*Wired Equivalent Privacy*. (sd). Opgehaald van Wikipedia: https://nl.wikipedia.org/wiki/Wired_Equivalent_Privacy

*Wireless distribution system*. (sd). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Wireless_distribution_system

## 14.6 Bijlage VPN onderzoek

2016

# Test rapport VPN

**Afstudeerstage VMB Automation**


**10059490**
**JORIS GROEN**

**25 AUGUSTUS 2016**


HAAGSE HOGENSCHOOL | VMB besturingstechniek

# 1 Inleiding

Voor het versturen van data van het schip naar de vaste wal is er een systeemnodig die contact maken met een server op de wal en de data naar de server kan zenden. Daarnaast moeten klanten verbinding kunnen maken met de server op de wal om de data van hun schepen in te zien. De volgende eisen worden gesteld aan het data dataverbinding tussen de pc op het schip en de server op de wal:

- De pc op het schip moet via het internet verbinding kunnen maken met de server op de wal
- De dataverbinding moet gegarandeerd worden
- De dataoverdracht moet beveiligd  zijn tegen lezen door derden
- Ongewenste personen mogen geen verbinding kunnen maken met de server of client
- VMB moet de controle houden over de server. VMB moet kunnen bepalen wie er toegang heeft tot welke data.
- De data van verschillende gebruikers moeten gebruikers moeten gescheiden blijven van elkaar

De dat data kan beveiligd worden tegen lezen door derden door middel van encryptie. Door bij het verbinding maken met de server een vorm van authenticatie toe te passen kan gegarandeerd worden dat alleen mensen met toestemming verbinding kunnen maken met de server. Voor verbinden van de pc op het schip en de server op de wal zal een VPN verbinding opgezet worden tussen de router op het schip en de server op de wal. Voor de router op het schip wordt een 3G router van weidmuller gebruikt die mogelijkheden heeft om een VPN verbinding op te zetten.

## 2 VPN

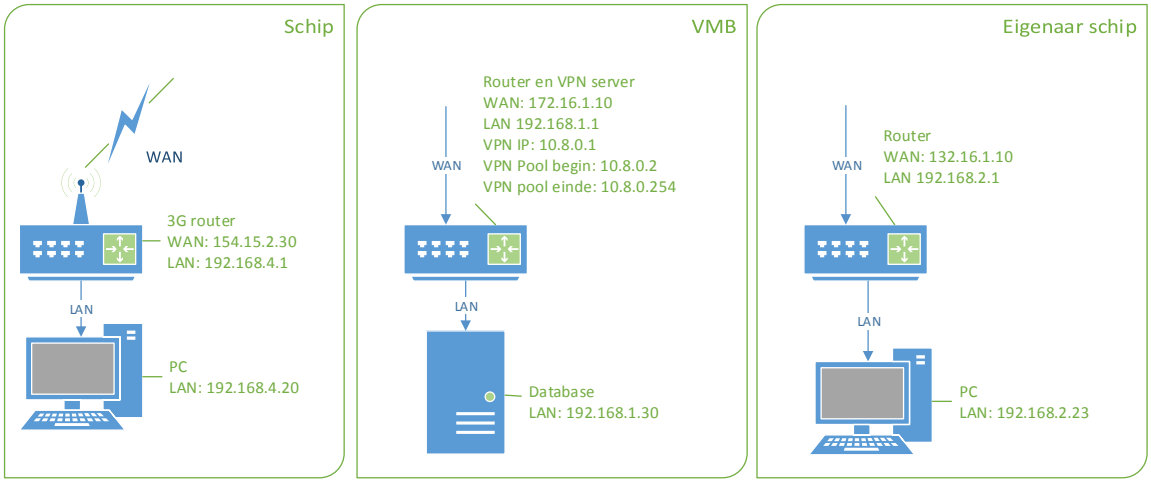
VPN staat voor virtual private network. VPN wordt gebruikt om een verbinding van een pc op een particulier (private) netwerk te tunnelen over bijvoorbeeld het internet naar een andere pc op een ander particulier netwerk. Als een apparaat zich aanmeld bij de VPN server wordt na goedkeuring van diverse controles het apparaat toegevoegd aan het VPN. Het apparaat krijgt een eigen ip adres in het VPN. Het is mogelijk om communicatie tussen cliënt's onmogelijk te maken. Hierdoor kunnen de cliënt's alleen communiceren met de VPN server.

VPN servers kunnen gehuurd worden bij externe partijen of kan zelf opgezet worden. Simpele VPN servers kunnen draaien op een pc, maar professionele VPN's worden meestal gedraaid op een aparte server. Het is ook mogelijk om een router die VPN ondersteund op te zetten als VPN server.

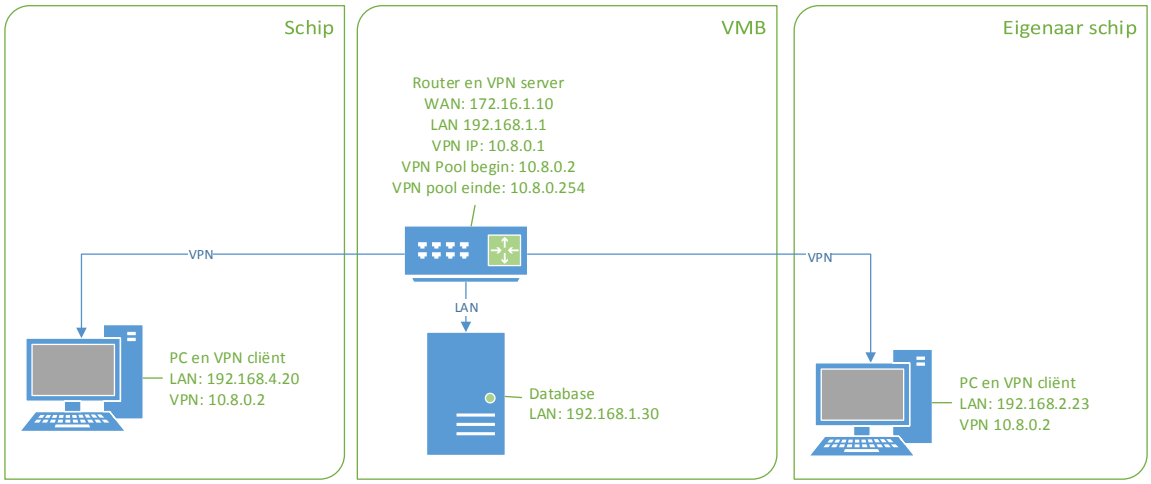
Er zijn twee mogelijkheden voor het opzetten van een VPN netwerk. Je kan een Site verbinding toevoegen dan worden routers als cliënt toegevoegd aan een VPN, als het LAN netwerk achter de router geroute wordt naar andere VPN cliënt's dan kunnen anderen cliënt's het LAN netwerk benaderen. Dit is handig als je meerdere apparaten in het zelfde LAN netwerk wilt toevoegen aan een VPN. Er moet wel gelet worden op de netwerkadressen van de LAN netwerken achter de routers die als cliënt aan het VPN worden toegevoegd. De netwerkadressen mogen niet het zelfde zijn anders krijg je ip conflicten. Het is ook mogelijk om Routers te laten werken als VPN servers.

Je kan ook een enkele pc toevoegen aan een VPN. Alleen de pc kan andere cliënts benaderen en andersom. Hierdoor wordt de rest van het LAN netwerk waar de pc op aangesloten is afgeschermd van het VPN.

Hieronder volgt een voorbeeld van een VPN netwerk. Twee pc's proberen verbinding te leggen met de database. Alleen de pc's en de database zitten in verschillende netwerken. Door een VPN verbinding op te zetten kunnen de pc verbinding maken met de database. Door alleen server naar client communicatie toe te staan kan de eigenaar van het schip niet direct het schip benaderen. Hierdoor houd de eigenaar van de server controle over wie welke data kan inzien. Na het opzetten van de VPN verbinding kunnen de pc de VPN server benaderen en via gepushte routing informatie ook de database.




**Figuur 2: Netwerk voor opzetten VPN**




**Figuur 1: Netwerk na opzetten VPN**

# 3 Beveiliging

De beveiliging van een VPN verbinding heeft 3 onderdelen:

- Encryptie
- Authenticatie

Deze beveiligingen zorgen er voor dat alleen de gewenste personen verbinding kunnen maken met het VPN. Dat de informatie die d.m.v. de VPN verbinding over het internet verzonden wordt, niet te lezen is door andere mensen op het internet. En zorgt er voor dat informatie die alleen voor bepaalde mensen op het VPN bedoelt is niet door andere mensen op het VPN te bekijken is.

## 3.1 Encryptie

Encryptie is een manier om informatie veilig te verzenden van een zender naar een ontvanger. Encriptie is niks anders dan informatie omzetten in geheimtaal, verzenden, ontvangen en daarna weer vertalen naar begrijpelijke taal. Dit wordt gedaan met sleutels. Als beide partijen de juiste sleutels hebben kunnen ze ontvangen berichten ontcijferen en verzonden berichten vercijferen. Je hebt twee manieren van encryptie: asynchrone encryptie en synchrone encryptie.

**Synchrone encryptie**

Iedere partij heeft 1 identieke sleutel waarmee zij berichten vercijferen en ontcijferen. Met synchrone encryptie kunnen bestanden sneller vercijferd en ontcijferd dan met asynchrone encryptie. Dit type encryptie wordt vooral gebruikt voor het vercijferen en ontcijferen van grote bestanden. Verder is het lastig om de sleutel aan andere partijen door te geven. Zonder dat ongewenste personen de sleutel in handen krijgen. Als de sleutel in verkeerde handen valt kan iedereen de geëncrypte data lezen.

**Asynchrone encryptie**

De basis van asynchrone encryptie werd gelegd door Whitfield Diffie en Martin Hellman. Bij asynchrone encryptie heeft iedere gebruiker twee sleutels een sleutel waarmee alleen vercijferd kan worden en een sleutel waarmee ontcijferd kan worden. De vercijfersleutel wordt de public key genoemd, omdat deze sleutel aan iedereen die data naar de partij wil sturen verzonden wordt. Op deze manier kan iedereen vercijferde data naar de eigenaar van de public key stuurt. De ontcijfer key word de private key genoemd, omdat deze sleutel nooit wordt gedeeld met een andere partij. Hierdoor kan de eigenaar van de private key berichten die vercijferd zijn met zijn public key ontcijfer. Doordat met de public key geen berichten ontcijferd kunnen worden is het veilig deze met alle nodige partijen te delen. Asynchrone encryptie kost meer tijd om te vercijferen en ontcijferen, daarom word asynchrone encryptie vooral toegepast bij kleine bestande formaten.

Door beide methode te combineren kan je de beste eigenschappen van asynchrone en synchrone encryptie samen voegen. Door m.b.v. asynchrone encryptie een symmetrische sleutel veilig te versturen kunnen grote datapakketten m.b.v. de synchrone sleutel ontcijferd worden.

## 3.2 Authenticatie

Als cliënt A zich aanmeld bij VPN server dan moet geverifieerd worden dat cliënt A daadwerkelijk cliënt A is en niet een vermomde cliënt B die zich voordoet als cliënt A. Dit proces wordt Authenticatie genoemd. Dit kan met behulp van gebruikersnamen en wachtwoorden, certificaten of een combinatie van de twee.

Bij certificaten worden server certificaten aangemaakt en cliënt certificaten aangemaakt. De certificaten worden allemaal ondertekent met het zelfde hoofd certificaat. Als cliënt A zich aanmeld bij VPN server dan zullen de server en de cliënt hun certificaten uitwisselen. De VPN server controleert het certificaat van cliënt A. De VPN server controleert of het certificaat ondertekent is met het juiste hoofdcertificaat of het een cliënt certificaat en of de commonname in het certificaat overeenkomt met een configuratie bestand in zijn configuratiemap. Als dit klopt word de cliënt verbonden aan het VPN netwerk.

# 4 Beveiligingsprotocollen

Bij een VPN heb je een aantal beveiligingsprotocollen die op verschillende manieren de VPN verbinding beveiligen d.m.v. encryptie en authenticatie. De vier bekendste vormen zijn:

| Protocol | OSI laag |
|---|---|
| L2TP/IPsec | Datalink |
| PPTP | Datalink |
| SSL/TSL | Sessie |
| Openvpn | Sessie |

**L2TP/IPsec**

L2TP/IPsec werkt op de 3e laag van het OSI model. L2TP/IPsec wordt vooral gebruikt voor site to site VPN verbindingen (als twee verschillende netwerken m.b.v. een VPN verbinding aan elkaar gekoppeld worden). L2TP/IPsec gebruikt de beste eigenschappen van het L2TP en IPsec protocol. L2TP zorgt voor het “tunnelen” van de VPN verbinding. IPsec verzorgt de encryptie en authenticatie van de VPN verbinding.

**PPTP**

Een beveiligingsprotocol dat ontwikkeld is in de jaren 90. PPTP maakt geen gebruik van encryptie. Hierdoor wordt protocol alleen gebruikt als de veiligheid van de VPN verbinding niet belangrijk is.

**SSL/TSL**

Het secure socket layer protocol wordt veel gebruikt door overheden en grote bedrijven. Het SSL/TSL protocol wordt gebruikt voor het beveiligen van verbindingen bij bijvoorbeeld online betalen. Voor authenticatie maakt SSL/TSL gebruik van digitale certificaten die extra beveiligd kunnen worden met wachtwoorden. Voor encryptie voor het authenticeren van de verbinding gebruik gemaakt van asynchrone encryptie met een

public en een private sleutel. Als de verbinding beveiligd is word op basis van een public sleutel van de client of de server een synchrone sleutel gemaakt waarmee de verbinding beveiligd wordt. Dit maakt de verbinding sneller. Voor de veiligheid wordt om het uur een nieuwe sleutel aangemaakt nadat opnieuw geverifieerd is dat de verbinding veilig is.

### OpenVPN

OpenVPN is een gratis openscource beveiligingsprotocol dat gebaseerd is op het SSL/TSL beveiligingsprotocol. De werking van het openVPN protocol is het zelfde als het SSL/TSL protocol. Het enige verschil is dat er client en server software geïnstalleerd moet worden op de pc en de VPN server om een VPN op te kunnen zetten. In de configuratie bestanden van de openVPN Server kan worden aangegeven met welke apparaten aangesloten op het VPN de client's verbinding kunnen maken. Bij het opzetten van de verbinding wordt als ondersteuning het Diffie–Hellman protocol gebruikt.

## 5 Keuze

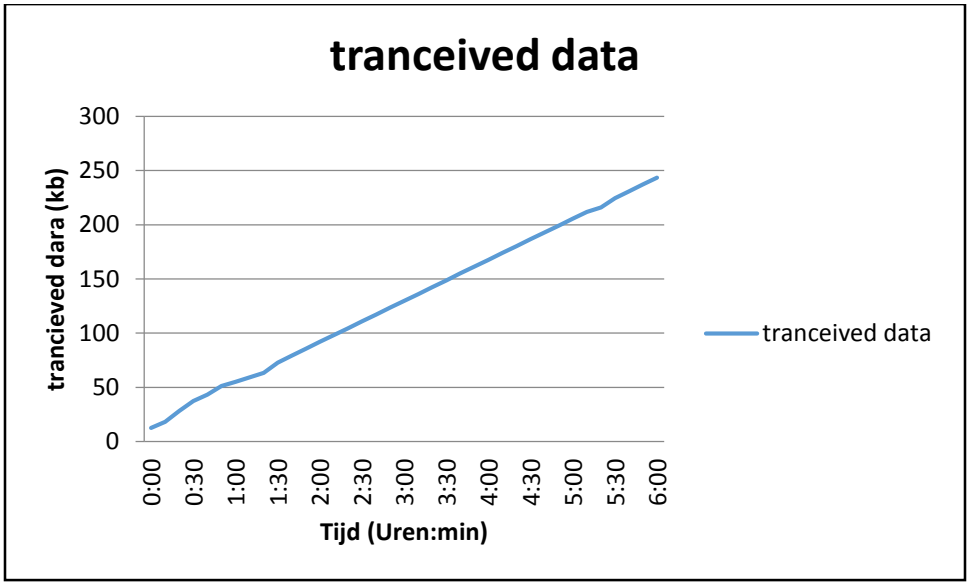
Als VPN server bij VMB zal een router gebruikt worden die de mogelijkheid heeft om als VPN server te werken. Op het schip zal een PC d.m.v. een VPN verbinding maken met de database bij VMB die in het LAN netwerk licht achter de router die ook als VPN server werkt. Door dit door een PC i.p.v. van de 3G router op het schip te laten doen wordt voorkomen dat alle andere apparaten (besturing e.d.) onnodig benaderd kunnen worden vanuit de VPN server. De klant kan m.b.v. de openVPN software verbinding maken met de server. (zie Figuur 2 en Figuur 1)

Voor het beveiligen van de VPN zal gebruik gemaakt worden van het beveiligingsprotocol openVPN. Hier is voor gekozen omdat de openVPN software gratis te downloaden is gebruik maakt van een van de veiligste beveiligingsprotocollen en zowel verbindingen voor routers als voor pc's ondersteund. Door alleen server naar cliënt communicatie toe te staan wordt voorkomen dat klanten zelf verbinding kunnen maken met de PC op hun schip. Hierdoor houd VMB controle over de data. Later kan de database zo ingesteld worden dat klanten alleen bij data kunnen van hun eigen schepen.

De router die werkt als VPN server zal geplaatst worden bij VMB achter de router die de verbinding naar het internet verzorgt. Hierdoor krijgt de VPN server de zelfde beveiliging als alle andere apparaten die op het netwerk zijn aangesloten. Bij VMB is een noodstroomvoorziening met noodstroomaggregaat aanwezig. Hierdoor kan ook bij stroomuitval de VPN verbinding gegarandeerd worden.

# 6 Dataverbruik

Het is met het oog financiële kosten voor de schippers belangrijk dat het data verbruik zo laag mogelijk blijft. Er is onderzocht hoeveel MB het kost om een VPN verbinding een maand lang open te houden zonder er data overheen te sturen. De gebruikte data is overhead voor het instant houden van de verbinding. Het dataverbruik van de VPN verbinding is over een tijdsduur van 6 uur gelogd. Het dataverbruik voor het behouden van de VPN verbinding was 250kB per 6 uur. Deze berekeningen zijn onder voorbehoud. Bij het idle dataverbruik hangt veel af van de instellingen van en omstandigheden waaronder een VPN verbinding wordt opgezet.



# 7 Bibliografie


Crawford, J. T. (sd). *How VPNs Work*. Opgehaald van howstuffworks Tech: http://computer.howstuffworks.com/vpn1.htm

*Diffie–Hellman key exchange*. (sd). Opgehaald van wikipedia: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

geier, e. (sd). *How and why to setup a vpn today*. Opgehaald van pcworld: http://www.pcworld.com/article/2030763/how-and-why-to-set-up-a-vpn-today.html

*IPsec*. (sd). Opgehaald van wikipedia: https://nl.wikipedia.org/wiki/IPsec

*openvpn*. (sd). Opgehaald van how to vpn: https://openvpn.net/index.php/open-source/documentation/howto.html#auth

Phifer, L. (sd). *Tunnel vision: Choosing a VPN -- SSL VPN vs. IPSec VPN*. Opgehaald van searchsecurity.techtarget: http://searchsecurity.techtarget.com/feature/Tunnel-vision-Choosing-a-VPN-SSL-VPN-vs-IPSec-VPN

*Public-key cryptography*. (sd). Opgehaald van wikipedia: https://en.wikipedia.org/wiki/Public-key_cryptography

*RSA (cryptografie)*. (sd). Opgehaald van wikipedia: https://nl.wikipedia.org/wiki/RSA_(cryptografie)#Sleutels

## 14.7 Bijlage broncode

# HARDWARE_CONTROL_MODULES.c

```
//designed original scource: Joris Groen
//and Enrico Molinari from syel Italy for parity and stop bit implemntation.
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 20-07-2016

//**********defines for HCM's*********************************************//
//to set stop, start and parity bits for serial port (uart 0-3)

#ifndef DEF_ALREADY_DEFINED;

    #define UART0_BASE_ADDR  0xE000C000
    #define U0LCR (*(volatile unsigned long *)(UART0_BASE_ADDR + 0x0C))
    #define UART1_BASE_ADDR  0xE0010000
    #define U1LCR (*(volatile unsigned long *)(UART1_BASE_ADDR + 0x0C))
    #define UART2 BASE ADDR  0xE0078000
    #define U2LCR (*(volatile unsigned long *)(UART2_BASE_ADDR + 0x0C))
    #define UART3_BASE_ADDR  0xE007C000
    #define U3LCR (*(volatile unsigned long *)(UART3_BASE_ADDR + 0x0C))

    #define DEF_DEVICE 1  //V4L =1 V7=2 V10=3

    #if DEF_DEVICE == 1
        #define DEF_DI 4
        #define DEF_DO 4
        #define DEF AI 4
        #define DEF_AO 2
    #endif

    #if DEF_DEVICE == 2
        #define DEF_DI 16
        #define DEF DO 16
        #define DEF AI 2
        #define DEF_AO 3
    #endif //DEF_V7

    #if DEF_DEVICE == 3
        #define DEF DI 32
        #define DEF DO 32
        #define DEF_AI 8
        #define DEF_AO 4
    #endif //DEF_V10



    #ifndef DEF_REMOTEIO
        #define DEF_REM_DI 0 //define the number of remote DI io
        #define DEF REM AI 0 //define the number of remote AI io
        #define DEF REM AO 0 //define the number of remote AO io
        #define DEF REM DO 0 //define the number of remote DO io
    #endif // DEF_REMOTE_IO

    #ifdef DEF REMOTEIO
        #define DEF REM DI ? //define the number of remote DI io
        #define DEF REM AI ? //define the number of remote AI io
        #define DEF_REM_AO ? //define the number of remote AO io
        #define DEF_REM_DO ? //define the number of remote DO io
    #endif // DEF_REMOTE_IO

#endif //DEF_ALREADY_DEFINED

//include files.
#include "HCM_IO_LIB.c";
#include "HCM_SERIAL_LIB.c"
#include "HCM MODBUS LIB.c"
#include "HCM TCPIP LIB.c"
#include "HCM_CAN_LIB.c";




//-----------------------------------------------------------------------------//

//Global programm variables
```

```
//-------------------------------------------------------------------------------//
struct hcmIOData sthcmIOData;      //IO stryuct
struct hcmDataSetSer sthcmDataSetSer[3];
struct hcmDataSetRSMod sthcmDataSetRSMod;
struct cmHardwareDataSetTcpIp stcmHardwareDataSetTcpIp[4];
struct hcmDataSetCan sthcmDataSetCan;

//-------------------------------------------------------------------------------//
//Initialize hardware function. initializes all the hardware on the PLC
//-------------------------------------------------------------------------------//
void _InitHCM(){
    int x = 0;
    //initialize IO
    _InitIO(&sthcmIOData, 0);

    //initialize Serial port
    _InitSer(&sthcmDataSetSer[0], COM1, 9600);
    _InitSer(&sthcmDataSetSer[1], COM2, 115200);
    _InitSer(&sthcmDataSetSer[2], COM3, 9600);

    //initialize the modbus
     InitMod(&sthcmDataSetRSMod, &sthcmDataSetSer[0], 3);
    _SetParStop(&sthcmDataSetSer[2], 0x03);
    _SetParStop(&sthcmDataSetSer[0], 0x03);

     InitTcpIp(&stcmHardwareDataSetTcpIp[0], "192.168.1.12", 2600, 4600, 1);
     InitTcpIp(&stcmHardwareDataSetTcpIp[1], "192.168.1.100", 2500, 4500, 0);
    /*_InitTcpIp(&stcmHardwareDataSetTcpIp[2], "192.168.1.103", 2501, 4503, 0);
    _InitTcpIp(&stcmHardwareDataSetTcpIp[3], "192.168.1.104", 2502, 4504, 0); */

    _InitCan(&sthcmDataSetCan, 0, CANBitrate1M, 1);

}

//For hardware function thet need to be called every programmsycle
void  RunHCM(){
    int x = 0;

    _UpdateIO(&sthcmIOData);

    _UpdateSer(&sthcmDataSetSer[2]);
     UpdateSer(&sthcmDataSetSer[1]);
     UpdateSer(&sthcmDataSetSer[0]);
    _UpdateMod(&sthcmDataSetRSMod);

    for(x = 0; x<1; x++){
        _UpdateTcpIp(&stcmHardwareDataSetTcpIp[x]);
    }
    _UpdateCan(&sthcmDataSetCan);

};
```

# HCM_IO_LIB

```
#include "HCM IO LIB.h";
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

void _MeassureCounterPerSec(); // for task exc DONT MOVE OR MODIFY

//initiaize the input and output array's and initialize all the outputs to 0
void _InitIO(struct hcmIOData *B, unsigned char RemoteIO){
    B->ucModuleNr = 1;
    B->NrOfDI = DEF_DI+1+DEF_REM_DI;
    B->NrOfDO = DEF_DO+1+DEF_REM_DO;
    B->NrOfAI = DEF_DO+1+DEF_REM_AI;
    B->NrOfAO = DEF_AO+1+DEF_REM_AO;
    B->Device = DEF_DEVICE;

    int x = 0;
    for(x = 0; x < B->NrOfDO; x++){
        B->bDO[x] = 0;
        out((x), 0);
    }
    __o();
    for(x = 1; x < B->NrOfAO; x++){
        B->usiAO[x] = 0;
        anal[x] = 0;
    }
    __i();
    B->bDI[0] = 0;
    for(x = 1; x < B->NrOfDI; x++){
        B->bDI[x] = I(x);
    }
    B->usiAI[0] = 0;
    for(x = 1; x < B->NrOfAI; x++){
        B->usiAI[x] = pot[x-1];
    }
    if(B->Device == 1) x= 4;
    if(B->Device == 2) x= 16;
    if(B->Device == 3) x= 32;

    B->bChangePerSec = 0 ;
    B->ucChangePerSecOld = 0;
    B->bChangePerSecInd = 0;
    memset(B->usiCounterRisingEdge, 0, (x+1));
    memset(B->usiCounterRisingEdgePerSec, 0, (x+1));
    memset(B->bMark, 0, (x+1));
    phcmIOData = B;
    init gpio EXT IRQ(B);
    if(RemoteIO > 0){
        can_perif_ext((int)RemoteIO, 1, 1000, 1);
    }
}
```

```
void _V4LIntIO()
{
    VICIntEnClr|=(1<<17);
    if(IO0 INT STAT R&(1<<8)){
        phcmIOData->usiCounterRisingEdge[1]++;
        phcmIOData->bMark[1] = 1;
        IO0_INT_CLR|=(1<<8); // set this flag in order to clear pending IRQ
    }
    if(IO0 INT STAT R&(1<<9)){
        phcmIOData->usiCounterRisingEdge[2]++;
        phcmIOData->bMark[2] = 1;
        IO0_INT_CLR|=(1<<9); // set this flag in order to clear pending IRQ
    }
    if(IO2 INT STAT R&(1<<14)){
        phcmIOData->usiCounterRisingEdge[3]++;
        phcmIOData->bMark[3] = 1;
        IO2_INT_CLR|=(1<<14); // set this flag in order to clear pending IRQ
    }
    if(IO2 INT STAT R&(1<<15)){
        phcmIOData->usiCounterRisingEdge[4]++;
        IO2_INT_CLR|=(1<<15); // set this flag in order to clear pending IRQ
    }
    if((IO2_INT_STAT_R&(1<<14)) && phcmIOData->bMark[3] == 0){
        phcmIOData->usiCounterRisingEdge[3]++;
        IO2_INT_CLR|=(1<<14); // set this flag in order to clear pending IRQ
    }
    if((IO0 INT STAT R&(1<<9)) && phcmIOData->bMark[2] == 0){
        phcmIOData->usiCounterRisingEdge[2]++;
        IO0_INT_CLR|=(1<<9); // set this flag in order to clear pending IRQ
    }
    if((IO0 INT STAT R&(1<<8)) && phcmIOData->bMark[1] == 0){
        phcmIOData->usiCounterRisingEdge[1]++;
        IO0_INT_CLR|=(1<<8); // set this flag in order to clear pending IRQ
    }

    memset(phcmIOData->bMark, 0, 5);
    VICVectAddr=0;
    VICIntEnable|=(1<<17);
}

void init_gpio_EXT_IRQ(struct hcmIOData *B)
{
    SCS|=1;
    if(B->Device == 1){
        IO0_INT_CLR|=(1<<8); // set this flag in order to clear pending IRQ
        IO0_INT_CLR|=(1<<9); // set this flag in order to clear pending IRQ
        IO2_INT_CLR|=(1<<14); // set this flag in order to clear pending IRQ
        IO2_INT_CLR|=(1<<15); // set this flag in order to clear pending IRQ
    }

    EXTINT=0xf; // 1111 (PORT3,PORT2,PORT1,PORT0)
    EXTMODE=0xf; // 0xf = binary 1111 = xxxx (PORT3,PORT2,PORT1,PORT0)

    if(B->Device == 1){
        install_irq(EINT3_INT,(void*)_V4LIntIO,NORMAL_PRIORITY);
        IO0_INT_EN_R|=(1<<8);
        IO0_INT_EN_R|=(1<<9);
        IO2_INT_EN_R|=(1<<14);
        IO2_INT_EN_R|=(1<<15);
        IO0_INT_CLR=(1<<8);
        IO0_INT_CLR=(1<<9);
        IO2_INT_CLR=(1<<14);
        IO2_INT_CLR=(1<<15);
    }
    exec_task(_MeassureCounterPerSec, 4, 1);
}
```

```
void _MeassureCounterPerSec(/*struct hcmIOData *B*/){
    while(1){
    every(100);
    unsigned char x = 0;

    unsigned char y = 0;
    if(phcmIOData->Device == 1) y = 4+1;
    if(phcmIOData->Device == 2) y = 16+1;
    if(phcmIOData->Device == 3) y = 32+1;
    VICIntEnClr|=(1<<17);
    for(x=1; x<y; x++){
        phcmIOData->usiCounterRisingEdgePerSec[x] = phcmIOData->usiCounterRisingEdge[x];
        phcmIOData->usiCounterRisingEdge[x] = 0;
    }
    VICIntEnable|=(1<<17);
     p(5);
    phcmIOData->bChangePerSec = 1;
    _v(5);
    /*time[1] =  ftime(0) - time[0];
    time[0] = _ftime(0);     */
    }
}

//read inputs into inpur arrays. write outputs from output arrays.
//force update inputs and outputs

void  ReadAndWriteIO(struct hcmIOData *B){
    int x = 0;
    for(x = 1; x < B->NrOfDO; x++){
        out((x), B->bDO[x]);
    }
      o();
    for(x = 1; x < B->NrOfAO; x++){
        anal[x] = B->usiAO[x];
    }
      i();
    for(x = 1; x < B->NrOfDI; x++){
        B->bDI[x] = I(x);
    }
    for(x = 1; x < B->NrOfAI; x++){
        B->usiAI[x] = pot[x-1];
    }
}

void _SignalNewVal(struct hcmIOData *B){
    _p(5);
    B->bChangePerSecInd = PulseOn(&(B->ucChangePerSecOld), B->bChangePerSec);
    B->bChangePerSec = 0;
    _v(5);
}

void _UpdateIO(struct hcmIOData *B){
     ReadAndWriteIO(B);
    _SignalNewVal(B);
}
```

# HCM_SERIAL_LIB.c

```
#include "HCM SERIAL LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 24-08-2016



//fill TXbuffer with new message to be send. TXBufferNr will contain the nr of char to be send
//if buffer isn't filledd TXBufferNr will remain 0 and nothing will be send over the Serial
port
void _FillTxBufferSer(struct hcmDataSetSer *B, unsigned char* c, unsigned short int length){
    unsigned short int x =0;
    for (x = 0; x < length; x++){
        B->ucTXBuffer[x] = c[x];
    }
    B->ucTXBufferNr = length;
}

//initialize
void _InitSer(struct hcmDataSetSer *B, COM *C, long int Baud){
    B->pPort = C;
    B->liBaud = Baud;
    B->ucTXBufferNr = 0;
    B->ucRXBufferNr = 0;
    B->RXErrorFlag = 0;
    //B->RXDelay= 0;
    //B->RXNewMessage = 0;
    //B->TXSendOk = 1; //someone has to send te first message //not used anymore
    com_open(B->pPort, B->liBaud);
}



//force send single char doesnt take the TXBufferNr into account
void _SendCharSer(struct hcmDataSetSer *B, unsigned char* c){
    /*unsigned long x =0;
    for(x=0; x<length; x++){
        com_tx(B->pPort, c[x]);
    } */
    com_txs(B->pPort, c);
}

//send message if buffer is filled eg TXBufferNr > 0.
//and if there is premission to send a new message. eg. if a response on the
//last send command has bin received by the controller
void  SendBufferSer(struct hcmDataSetSer *B){
    unsigned short int x = 0;
    if(B->ucTXBufferNr > 0){
        com_txsl(B->pPort, &(B->ucTXBuffer), (B->ucTXBufferNr));
        B->ucTXBufferNr = 0;
    }
}
```

```
//read the serial port. If there is an new message put it in ucRXbuffer.
void _ReceiveSer(struct hcmDataSetSer *B){
    B->ucRXBufferNr = 0;
    B->RXNewMessage = 0;
    while(!com_rx_empty(B->pPort))   //while in the buffer of rx is not empty
    {
        B->ucRXBuffer[B->ucRXBufferNr] = com_rx(B->pPort);
        B->ucRXBufferNr++;
        B->RXNewMessage = 1;
        if(com_rx_empty(B->pPort))wait(2);

    }
}

//read new messages and try to sennd buffer.
void  UpdateSer(struct hcmDataSetSer *B){
    _ReceiveSer(B);
    _SendBufferSer(B);
}

//funtion that temporarly disables the comport and configures the settings
//for the stop, start and parity bits and number of databits
//restvalue 0x00
//Bit   Symbol                  Value Description
//1:0   Word Length Select      00 5 bit character length
//                              01 6 bit character length
//                              10 7 bit character length
//                              11 8 bit character length
//2     Stop Bit Select         0 1 stop bit.
//                              1 2 stop bits (1.5 if UnLCR[1:0]=00).
//3     Parity Enable           0 Disable parity generation and checking.
//                              1 Enable parity generation and checking.
//5:4   Parity Select           00 Odd parity. Number of 1s in the transmitted
//                              character and the attached parity bit will be odd.
//                              01 Even Parity. Number of 1s in the transmitted
//                              character and the attached parity bit will be even.
//                              10 Forced "1" stick parity.
//                              11 Forced "0" stick parity.
//6     Break Control           0 Disable break transmission.
//                              1 Enable break transmission. Output pin UART0 TXD
//                              is forced to logic 0 when UnLCR[6] is active high.
//7     Divisor Latch           0 Disable access to Divisor Latches.
//      Access Bit              1 Enable access to Divisor Latche
//      (DLAB)

void _SetParStop(struct hcmDataSetSer *B, unsigned short int setting){
    com_disable(B->pPort);
    if (B->pPort == COM1){
        U0LCR    = setting;
    }
    if (B->pPort == COM2){
        U1LCR    = setting;
    }
    if (B->pPort == COM3){
        U2LCR    = setting;
    }
    if (B->pPort == COM4){
        U3LCR    = setting;
    }
    com_enable(B->pPort);


}
```

# HCM_MODBUS_LIB.c

```
#include "HCM MODBUS LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 24-08-2016

//initialize
void _InitMod(struct hcmDataSetRSMod *B, struct hcmDataSetSer *C, unsigned char par){
    B->pTXRXPort = C;
    B->ucParityStopMode = par;
    B->bExpectReqMessage = 0;
    B->bPremToSend = 1;
    B->bNewTXMessage = 0;
    B->bNewRXMessage = 0;
    _SetParStop(B->pTXRXPort, B->ucParityStopMode);
    B->ulRXDelayState = 0;
    B->bRXDelay = 0;
    B->ucRXdelayCounter = 0;
    B->ulTXDelayState = 0;
    B->bTXDelay = 0;
    B->ucTXdelayCounter = 0;
    B->bExpectRecMessageError = 0;
    B->bExpectReqNoMessage = 0;
    B->ucRxBufferLength = 0;

}

//converts 2 Usigned char to Unsigned Short Int
void _UcToUsi(unsigned short int *temp, unsigned char RegHi, unsigned char RegLo){
    *temp = RegHi;
    *temp <<= 8;
    *temp = *temp + (unsigned short int)RegLo;
}

//converts Unsigned Short Int to 2 Usigned char
void _UsiToUc(unsigned short int temp, unsigned char *RegHi, unsigned char *RegLo){
    *RegLo = (temp & 0x00FF);
    temp  >>= 8;
    *RegHi = temp;
}

//creates the Modbus Message. returns 1 if creation was succesfull.
//can only create the message if there is premmission to send. E.G.
//the send buffer is empty and the repuly of the last send message has bin received
//after the creaton of the modbus message some flags will be set to hadnle the
//answer of the send message
bool _CreateMessageMod( struct hcmDataSetRSMod *B, unsigned char SlaveAddress,
                        unsigned char RFunction, unsigned char StartAdressHi,
                        unsigned char StartAdressLo, unsigned char NumberOfRegHi,
                        unsigned char NumberOfRegLo)
{
    if (B->bPremToSend == 1){
        if ((RFunction == 0x04) || (RFunction == 0x03)){
            unsigned short int x = 0;
            B->ucTXMessageLength = 8;
            unsigned short int RegNr = 0;
            unsigned short int Regstart = 0;

            B->ucTXMessage[0] = SlaveAddress;
            B->ucTXMessage[1] = RFunction;
            B->ucTXMessage[2] = StartAdressHi;
            B->ucTXMessage[3] = StartAdressLo;
            B->ucTXMessage[4] = NumberOfRegHi;
            B->ucTXMessage[5] = NumberOfRegLo;


            _CreateCrcMod(B->ucTXMessage, &(B->ucTXMessage[6]), &(B->ucTXMessage[7]), (B-
>ucTXMessageLength)-2);

            _UcToUsi(&RegNr, NumberOfRegHi, NumberOfRegLo);
            _UcToUsi(&Regstart, StartAdressHi, StartAdressLo);
            B->ucTXDevAddress = SlaveAddress;
            B->ucTXFunctionCode = RFunction;
            B->ucRxBufferLength = RegNr;
            for(x = 0; x < RegNr; x++){
                B->usiRXRegAddress[x] = Regstart+(unsigned short int)x;
```

```
                //B->usiRxRegValue[x]= 0;
            }

            B->bExpectReqMessage = 1;
            B->bPremToSend = 0;
        }
        else if(RFunction == 6){
            unsigned short int x = 0;
            B->ucTXMessageLength = 8;
            unsigned short int RegNr = 0;
            unsigned short int Regstart = 0;

            B->ucTXMessage[0] = SlaveAddress;
            B->ucTXMessage[1] = RFunction;
            B->ucTXMessage[2] = StartAdressHi;
            B->ucTXMessage[3] = StartAdressLo;
            B->ucTXMessage[4] = NumberOfRegHi;//value to be written to the register
            B->ucTXMessage[5] = NumberOfRegLo;//value to be written to the register


             CreateCrcMod(B->ucTXMessage, &(B->ucTXMessage[6]), &(B->ucTXMessage[7]), (B-
>ucTXMessageLength)-2);

            _UcToUsi(&RegNr, NumberOfRegHi, NumberOfRegLo);
             UcToUsi(&Regstart, StartAdressHi, StartAdressLo);
            B->ucTXDevAddress = SlaveAddress;
            B->ucTXFunctionCode = RFunction;
            B->usiRXRegAddress[0] = Regstart;
            //B->usiRxRegValue[0]= 0;

            B->bExpectReqMessage = 1;
            B->bPremToSend = 0;
        }

        B->bNewTXMessage = 1;
        return 1;
    }
    return 0;
}


//is used to create checksum for send messages and to check the checksum of
//received messages.
void  CreateCrcMod(unsigned char *c, unsigned char *lo, unsigned char *hi, unsigned short int
length ){
    int x = 0;
    int i =0;
    unsigned short int crc = 0xFFFF;
    unsigned char crcHi = 0;
    unsigned char crcLo = 0;
    unsigned short int message[length];
    for (x = 0; x < length; x++){
        crc ^= (unsigned short int)c[x];
        int i =0;
        for (i = 0; i < 8; i++){
            if((crc & 0x0001) == 0) {
                crc >>= 1;
            }
            else {
                crc >>= 1;
                crc ^= 0xA001;
            }
        }
    }

    crcLo = (crc & 0x00FF);
    crc >>= 8;
    crcHi = crc;
    *lo = crcLo;
    *hi = crcHi;
}

//if a new message is created. this function will send i once over the serial line.
void _SendMessageMod(struct hcmDataSetRSMod *B){
    if(B->bNewTXMessage){
        B->bTXDelay = DelayUp(&(B->ulTXDelayState), 10, !(B->bTXDelay));
        if(B->bTXDelay){
```

```
            B->ucTXdelayCounter = B->ucTXdelayCounter +1;
        }
        if (B->ucTXdelayCounter > 4){
            //B->bPremToSend = 0;
            _FillTxBufferSer(B->pTXRXPort, B->ucTXMessage, B->ucTXMessageLength);
            B->ucTXdelayCounter = 0;
            B->bNewTXMessage = 0;
            //test7++ ;
        }
    }
}

//if a new message has arrived in the Serial RX buffer this function will read that messeage.
//first it chack if the message is undammaged (crc check). then the fucntion checks if message
//is the answer on the last send messages. if message is ok the the content of the message
//will be put in the appropiate aarrays (e.g.value of red registers will be put in the value
array).
//if there is no reply on a request for 500ms the message is considered lost.
//a recerror falg will be raised for one cycle. and a new request can be send
//(as if a reply did arrive)
void  ReadMessage(struct hcmDataSetRSMod *B){
    unsigned char MessageOk = 0; //0 = wrong messaga
    B->bNewRXMessage = 0;
    B->bExpectRecMessageError = 0;
    MessageOk = 0;

    if (B->pTXRXPort->RXNewMessage == 1){
        unsigned char crcHi = 0;
        unsigned char crcLo = 0;
        //unsigned char crcHiH = 0;


        B->ucRXdelayCounter = 0;

        _CreateCrcMod((B->pTXRXPort->ucRXBuffer), &crcLo, &crcHi, (B->pTXRXPort-
>ucRXBufferNr)-2);
        if((B->pTXRXPort->ucRXBuffer[B->pTXRXPort->ucRXBufferNr-1] == crcHi) &&
           (B->pTXRXPort->ucRXBuffer[B->pTXRXPort->ucRXBufferNr-2] == crcLo)){

            if((B->pTXRXPort->ucRXBuffer[0] == B->ucTXMessage[0]) &&
               (B->pTXRXPort->ucRXBuffer[1] == B->ucTXMessage[1])){
                MessageOk = 1;
                B->bNewRXMessage = 1;
           }
        }
    }
    else{
        if(B->bExpectReqMessage == 1){
            B->bRXDelay = DelayUp(&(B->ulRXDelayState), 100, !(B->bRXDelay));
            if(B->bRXDelay){
                B->ucRXdelayCounter = B->ucRXdelayCounter +1;
            }
            if (B->ucRXdelayCounter > 5){
                B->ucRXdelayCounter = 0;
                B->bPremToSend = 1;
                B->bExpectReqMessage = 0;
                B->bExpectRecMessageError = 1;
            }
        }
        else if(B->bExpectReqNoMessage == 1){
            B->bRXDelay = DelayUp(&(B->ulRXDelayState), 10, !(B->bRXDelay));
            if(B->bRXDelay){
                B->ucRXdelayCounter = B->ucRXdelayCounter +1;
            }
            if (B->ucRXdelayCounter > 5){
                B->ucRXdelayCounter = 0;
                B->bPremToSend = 1;
                B->bExpectReqNoMessage = 0;
            }
        }
    }

    if(MessageOk &&((B->pTXRXPort->ucRXBuffer[1] == 4)||(B->pTXRXPort->ucRXBuffer[1] == 3))){
        unsigned long int x = 0;
        unsigned short int value = 0;
        for(x = 0; x < ((B->pTXRXPort->ucRXBuffer[2])/2); x++){
```

```
            _UcToUsi(&value, B->pTXRXPort->ucRXBuffer[x*2+3], B->pTXRXPort-
>ucRXBuffer[x*2+4]);
            B->bExpectReqMessage = 0;
            B->bPremToSend = 1;
            B->usiRxRegValue[x] = value;
        }
    }
    if(MessageOk &&(B->pTXRXPort->ucRXBuffer[1] == 6)){
        unsigned long int x = 0;
        unsigned short int value = 0;
        _UcToUsi(&value, B->pTXRXPort->ucRXBuffer[2], B->pTXRXPort->ucRXBuffer[3]);
        if(value == B->usiRXRegAddress[0]){
             UcToUsi(&value, B->pTXRXPort->ucRXBuffer[4], B->pTXRXPort->ucRXBuffer[5]);
            B->bExpectReqMessage = 0;
            B->bPremToSend = 1;
            B->usiRxRegValue[0] = value;
        }
    }

}

//function the bundles the send and receive function in one.
void _UpdateMod(struct hcmDataSetRSMod *B){
    _SendMessageMod(B);
     ReadMessage(B);
}
```

# HCM_TCPIP_LIB.c

```
    #define BUFFER SIZE 100
#define MESSAGE LENGTH 210
#include "HCM_TCPIP_LIB.h"
////////////////////////////////////////////////////////////////////////////
//Designed original scource: Joris Groen & Arjan Ippel (checkStatus() and AutoConnect() and
Buffer principle)
//Futher implemented for SMMS and expanded by: Joris Groen
//Date: 20-08-2016

//Creer een server ip waar de berichten naar worden verstuurd.
void _InitTcpIp(struct cmHardwareDataSetTcpIp *B, char* ip, unsigned short int sendPort,
unsigned short int  recPort, bool Mode){
    B->pTcpHost = 0;
    B->bMode = Mode;
    B->cpTcpIp = ip;
    B->usiSendPort = sendPort;
    B->usiRecPort = recPort;
    B->ucSendMessages = 0;
    B->ucRecMessages = 0;
    B->ucTotalSendMessages = 0;
    B->ucTotalRecMessages = 0;
    B->ucSendMessagesFreq = 0;
    B->ucRecMessagesFreq = 0;
    B->ucMessageLength = MESSAGE_LENGTH;
    B->ucBufferSize = BUFFER_SIZE;
    B->ucTXBuffernumber = 0;
    B->ucRXBuffernumber = 0;
    B->ucTXBufferSendNr = 0;
    B->TcpIpStatus = -1;
    B->bSendTimer =0;
    B->uliSendTimer = 0;
    B->bTXBufferSending = 0;
    B->bMessageFreqTimer = 0;
    B->uliMessageFreqTimer = 0;


    sprintf(B->cStatusTekst,"Niks");
    int x, y;
    for(x = 0; x < B->ucBufferSize; x++){
        for (y = 0; y < B->ucMessageLength; y++){
            B->ucTXBuffer[x][y] = 0;
            B->ucRXBuffer[x][y] = 0;
        }
    }
     CreateServerHostTcpIp(B);
    EMAC Tx(B->pTcpHost,0, TCP CREATE ,0);
}

////////////////////////////////////////////////////////////////////////////

void  CreateServerHostTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    // create host
   if (B->pTcpHost == 0)B->pTcpHost = CreateHost(B->cpTcpIp,B->usiSendPort,B->usiRecPort,0,0);
}

////////////////////////////////////////////////////////////////////////////
// Maak creeren een host en maak vervolgens verbinding
void  ConnectTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    if (B->pTcpHost == 0)
    {
        CreateServerHostTcpIp(B);
       EMAC_Tx(B->pTcpHost,0,_TCP_CREATE_,0);
    }
    if (B->pTcpHost)
    {
       EMAC Tx(B->pTcpHost,0, TCP CONNECT ,0);
    }
}

////////////////////////////////////////////////////////////////////////////
// Sluit de verbinding
void  DisconnectTcpIp(struct cmHardwareDataSetTcpIp *B)
{
```

```
    if (B->pTcpHost)
    {
       EMAC Tx(B->pTcpHost,0, TCP DISCONNECT ,0);
       free(B->pTcpHost);
       B->pTcpHost = 0;
       //_CreateServerHostTcpIp(B);
    }
}

////////////////////////////////////////////////////////////////////////////
// Sluit zonder ACK de verbinding
void _AbortTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    if (B->pTcpHost)
    {
       EMAC Tx(B->pTcpHost,0, TCP ABORT ,0);
    }
}

////////////////////////////////////////////////////////////////////////////
// vernietig de socket
void  DestroyTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    if (B->pTcpHost)
    {
       EMAC Tx(B->pTcpHost,0, TCP DESTROY  ,0);
       free(B->pTcpHost);
    }
    B->pTcpHost = 0;
}

////////////////////////////////////////////////////////////////////////////
// Zend een bericht
void _SendTcpIp(struct cmHardwareDataSetTcpIp *B, unsigned char TXMessage[],int
sizeofTXMessage)
{
    if (B->pTcpHost)
    {

       if (EMAC_Tx(B->pTcpHost,TXMessage,_TCP_SEND_,sizeofTXMessage)){
           B->ucTotalSendMessages++;
           B->ucSendMessages++;
       }
    }
}
////////////////////////////////////////////////////////////////////////////
//Inlezen van de TCP berichten en in een buffer plaatsen
void  ReceiveTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    int i = 0;
    if (B->pTcpHost){
        bool go = 1;
        B->ucRXBuffernumber = 0;
        char temp [B->ucMessageLength];
        while(go){
            if(B->ucRXBuffernumber < B->ucBufferSize){
                if (EMAC_Rx(B->pTcpHost,temp,_TCP_RECEIVE_,B->ucMessageLength)){
                    B->ucTotalRecMessages++;
                    B->ucRecMessages++;
                    int i = 0;
                    for (i = 0; i < (B->ucMessageLength); i++){
                        B->ucRXBuffer[B->ucRXBuffernumber][i] = temp[i];
                    }
                    B->ucRXBuffernumber++;
                }
                else{
                    go = 0;
                }
                //wait(10);
            }
            else go = 0;
        }
    }
}

////////////////////////////////////////////////////////////////////////////
//Controleer de status van TCP
```

```
void CheckStatusTcpIp(struct cmHardwareDataSetTcpIp *B)
{
    if (B->pTcpHost)
    {
       U32 temp;
       temp=EMAC_Tx(B->pTcpHost,0,_TCP_STATUS_,0);
       B->TcpIpStatus = temp &0xf;
    }
    else
    {
       B->TcpIpStatus = -1;
    }

    int i = 0;
    for(i = 0; i < 25; i++){
       B->cStatusTekst[i] = 0x00;
    }

    switch(B->TcpIpStatus)
    {
        case (-1):
        {
            sprintf(B->cStatusTekst,"Geen configuratie");
        }
        break;
        case 0:
        {
            sprintf(B->cStatusTekst,"Verbinding verbroken");
        }
        break;
        case 1:
        {
            sprintf(B->cStatusTekst,"Wacht op ACK van Server");
        }
        break;
        case 2:
        {
            sprintf(B->cStatusTekst,"Verbinding als Client");
        }
        break;
        case 4:
        case 5:
        {
           sprintf(B->cStatusTekst,"Verb. verbreken Client");
        }
        break;
        case 11:
        {
            sprintf(B->cStatusTekst,"Wacht op ACK van Client");
        }
        break;
        case 12:
        {
            sprintf(B->cStatusTekst,"Verbinding als Server");
        }
        break;
        case 15:
        {
            sprintf(B->cStatusTekst,"Verb. verbreken Server");
        }
        break;
    }
    B->bMessageFreqTimer = DelayUp(&(B->uliMessageFreqTimer), 1000, !(B->bMessageFreqTimer));
    if (B->bMessageFreqTimer){
        B->ucSendMessagesFreq = B->ucSendMessages;
        B->ucRecMessagesFreq = B->ucRecMessages;
        B->ucRecMessages = 0;
        B->ucSendMessages = 0;
    }

}

////////////////////////////////////////////////////////////////////////////
bool _FillTXbufferTcpIp(struct cmHardwareDataSetTcpIp *B, char TXMessage[], unsigned short int
MessageSize){
    unsigned int x;
    if((B->ucTXBuffernumber < B->ucBufferSize) && B->bTXBufferSending == 0){
```

```
        for(x = 0; x < MessageSize; x++){
            B->ucTXBuffer[B->ucTXBuffernumber][x] = TXMessage[x];
        }
        B->ucRXBufferMessageLength[B->ucTXBuffernumber] = MessageSize;
        B->ucTXBuffernumber++;
        //B->ucTXBufferSendNr = B->ucTXBuffernumber;
        return 1;
    }
    else{
        return 0;
    }
}

////////////////////////////////////////////////////////////////////////////
void  SendTXbufferTcpIp(struct cmHardwareDataSetTcpIp *B){
    int x = 0;
    if(B->ucTXBufferSendNr < B->ucTXBuffernumber){
        B->bTXBufferSending = 1;

        B->bSendTimer = DelayUp(&(B->uliSendTimer), 0, !(B->bSendTimer));
        if (B->bSendTimer){
        //for(x = 0; x < B->ucTXBuffernumber; x++){

            //B->ucTXBuffernumber--;
            unsigned char temp [B->ucMessageLength];
            int y = 0;
            for (y = 0; y < B->ucRXBufferMessageLength[B->ucTXBufferSendNr]; y++){
                temp [y] = B->ucTXBuffer[B->ucTXBufferSendNr][y];
                //B->ucTXBuffer[B->ucTXBuffernumber][y] = 0;
            }

             SendTcpIp(B, temp, B->ucRXBufferMessageLength[x]);
            B->ucTXBufferSendNr++;

        }
    }
    if(B->ucTXBufferSendNr >= B->ucTXBuffernumber){
        B->bTXBufferSending = 0;
        B->ucTXBufferSendNr = 0;
        B->ucTXBuffernumber = 0;
    }
}

////////////////////////////////////////////////////////////////////////////
void _AutoConnectieTcpIp(struct cmHardwareDataSetTcpIp *B) {

    B->bMakeConnection = DelayUp(&(B->uliTimerMakeConnection), 1000, !(B->bMakeConnection));
    if (B->bMakeConnection & (B->TcpIpStatus != 2))
    {
        _ConnectTcpIp(B);
    }
}
////////////////////////////////////////////////////////////////////////////
void  UpdateTcpIp(struct cmHardwareDataSetTcpIp *B){
    if(B->bMode) _AutoConnectieTcpIp(B);
    //if(button3){
        //button3 = 0;
        _SendTXbufferTcpIp(B);
         ReceiveTcpIp(B);
        CheckStatusTcpIp(B);
    //}
}
```

# HCM_CAN_LIB.c

```
#define CAN_TXBUFFER_LENGTH 20
#define CAN_RXBUFFER_LENGTH 20

#include "HCM_CAN_LIB.h";
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 20-08-2016

//inititalize can struct
void _InitCan(struct hcmDataSetCan* B, unsigned char CanPort, unsigned int bitRate, bool
Canv){
    B->uiBitRate = bitRate;
    B->ucTxbufferNr = 0;
    B->ucRxbufferNr = 0;
    B->ucTxbufferLength = CAN_TXBUFFER_LENGTH;
    B->ucRxbufferLength = CAN_RXBUFFER_LENGTH;
    B->bCanVersion = Canv;
    B->bCanPort = CanPort;
    B->bNewRXMessage = 0;
    if(B->bCanPort) canbase_init_2(4, bitRate);
    else canbase_init(4, bitRate);
    B->usiRecMessage = 0;

}

//fill TXbuffer. cant fill if TXbuffer is already full.
//example of temp.frame
//temp.frame 80040000 = 29 bit identifier and 4 bytes of data
//temp.frame 00080000 = 11 bit identifier and 8 bytes of data
//temp.frame 80020000 = 29 bit identifier and 2 bytes of data
void _FillTxBufferCan(struct hcmDataSetCan* B, unsigned long int ID, unsigned char data[],
unsigned char dataNr){
    if(B->ucTxbufferNr < B->ucTxbufferLength){
        CANBASE_MSG temp;
        temp.Frame = 0;
        if (B->bCanVersion){
            temp.Frame = 0x8000;
        }
        temp.Frame = temp.Frame + dataNr;
        temp.Frame <<= 16;

        temp.MsgID    = ID;

        unsigned char x = 0;
        temp.DatA =0;
        temp.DatB =0;
        unsigned char DataTemp[8] = {0, 0, 0, 0, 0, 0, 0, 0};

        for(x=0; x < dataNr; x++){
            DataTemp[x] = data[x];
        }
        temp.DatA +=  DataTemp[3];
        temp.DatA <<= 8;
        temp.DatA +=  DataTemp[2];
        temp.DatA <<= 8;
        temp.DatA +=  DataTemp[1];
        temp.DatA <<= 8;
        temp.DatA +=  DataTemp[0];

        temp.DatB +=  DataTemp[7];
        temp.DatB <<= 8;
        temp.DatB +=  DataTemp[6];
        temp.DatB <<= 8;
        temp.DatB +=  DataTemp[5];
        temp.DatB <<= 8;
        temp.DatB +=  DataTemp[4];
        /*
        for(x = 3; x > 0; x--){
                temp.DatA <<= 8;
                temp.DatA +=  DataTemp[x];
            }

        for(x = 7; x > 3; x--){
            temp.DatB <<= 8;
            temp.DatB +=  DataTemp[x];
```

```
        } */
        B->cmCanTxMessages[B->ucTxbufferNr] = temp;
        B->ucTxbufferNr = B->ucTxbufferNr + 1;
    }
}

//send all messages in TXbuffer. reset TXbuffernr
void  SendTxBufferCan(struct hcmDataSetCan* B){
    if(B->ucTxbufferNr > 0){
        unsigned int x = 0;
        for(x=0; x < B->ucTxbufferNr; x++){
            if(B->bCanPort)canbase_txmsg_2(&( B->cmCanTxMessages[x]));
            else canbase_txmsg(&( B->cmCanTxMessages[x]));
            //wait(1);
            //test++;

        }
        B->ucTxbufferNr = 0;
    }
}

//put a received message in temp and then put temp in RXbuffer, until all received messages
are stored in RXbuffer.
//if a message had bin received then set newRXmessage flag
void  ReadMessageCan(struct hcmDataSetCan* B){
    B->ucRxbufferNr = 0;
    B->bNewRXMessage = 0;
    bool temp = 1;
    while((temp == 1) && (B->ucRxbufferNr < CAN_RXBUFFER_LENGTH)){
        B->mesTemp.MsgID = 0;
        if(B->bCanPort)temp = canbase_rxmsg_2(&(B->mesTemp));
        else temp = canbase_rxmsg(&(B->mesTemp)); //return 1 if a message has bin recieved
        if(temp == 1){
            B->bNewRXMessage = 1;
            B->cmCanRxMessages[B->ucRxbufferNr].Frame = (unsigned long)B->mesTemp.Frame;
            B->cmCanRxMessages[B->ucRxbufferNr].MsgID = (unsigned long)B->mesTemp.MsgID;
            B->cmCanRxMessages[B->ucRxbufferNr].DatA = (unsigned long)B->mesTemp.DatA;
            B->cmCanRxMessages[B->ucRxbufferNr].DatB = (unsigned long)B->mesTemp.DatB;
            B->ucRxbufferNr = B->ucRxbufferNr + 1;
            B->usiRecMessage++;
        }

    }
}

//read received messages and send TXbuffer
void _UpdateCan(struct hcmDataSetCan* B){
     SendTxBufferCan(B);
    _ReadMessageCan(B);
}
```

# CONTROL_MODULES.c

```
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 20-07-2016



#ifndef DEF ALREADY DEFINED;
    #define DEF PRESS SENS 4
    #define DEF_OIL_QUA_SENS 4
    #define DEF_FLOW_SENS 2
#endif //DEF_ALREADY_DEFINED

#include "CM INTERN COM SEND LIB.c"
#include "CM_FLOW_SENS_LIB.c"
#include "CM_PRESS_SENS_LIB.c"
#include "CM_OIL_QUALITY_LIB.c"
#include "CM_J1939_LIB.c";



//#include "MICFILL_UNIT.h"

struct cmInternComSend stcmInternComSend;
struct cmPress stcmPress[DEF_PRESS_SENS];
struct cmOilQua stcmOilQua[DEF_OIL_QUA_SENS];
struct cmFlowSens stcmFlowSens[DEF_FLOW_SENS];
struct cmJ1939 stcmJ1939;

unsigned char ucUpdateCounter_GL = 0;

void _InitCM(){
    unsigned char x = 0;
    _InitInternComSend(&stcmInternComSend, &stcmHardwareDataSetTcpIp[0]);

     InitPress(1, &stcmPress[0], &sthcmIOData, &stcmInternComSend, 2);
     InitPress(2, &stcmPress[1], &sthcmIOData, &stcmInternComSend, 3);
    _InitPress(3, &stcmPress[2], &sthcmIOData, &stcmInternComSend, 4);
    _InitPress(4, &stcmPress[3], &sthcmIOData, &stcmInternComSend, 0);

     InitOilQua(5, &stcmOilQua[0], &sthcmDataSetRSMod, 0, &stcmInternComSend, 1, 0, 0);
     InitOilQua(6, &stcmOilQua[1], &sthcmDataSetRSMod, 0, &stcmInternComSend, 2, 0, 0);
    _InitOilQua(7, &stcmOilQua[2], &sthcmDataSetRSMod, 0, &stcmInternComSend, 3, 0, 0);
    _InitOilQua(8, &stcmOilQua[3], &sthcmDataSetRSMod, 0, &stcmInternComSend, 4, 0, 0);

     InitFlowSens(3, &stcmFlowSens[0], &sthcmIOData, &stcmInternComSend, 3, 0, 100, 0);
    _InitFlowSens(4, &stcmFlowSens[1], &sthcmIOData, &stcmInternComSend, 4, 0, 100, 0);

    _InitJ1939(1, &stcmJ1939, &sthcmDataSetCan, &stcmInternComSend);
}
void _RunCM(){
    unsigned char x = 0;
     UpdateInternComSend(&stcmInternComSend);
    stcmOilQua[ucUpdateCounter GL].stSet.bUpdateDone = 0;
    for(x =0; x < DEF_OIL_QUA_SENS; x++){
        _UpdateOilQua(&stcmOilQua[x]);
    }
    if(stcmOilQua[ucUpdateCounter GL].stSet.bUpdateDone == 1){
            ucUpdateCounter GL = ucUpdateCounter GL + 1;
            if(ucUpdateCounter GL > DEF_OIL_QUA_SENS-1){
                ucUpdateCounter_GL = 0;
            }
    }

    for(x =0; x < DEF_PRESS_SENS; x++){
        _UpdatePress(&stcmPress[x]);
    }
    for(x=0; x<DEF_FLOW_SENS; x++){
        _UpdateFlowSens(&stcmFlowSens[x]);
    }
    _UpdateJ1939(&stcmJ1939);

}
```

# CM_FLOW_SENS_LIB.c

```
#include "CM FLOW SENS LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

/*
initialize the cmFlowSens.
cmFlowSens* B: adres of the cmFlowSens stuct you want to initialize.
hcmIOData *C: adress of the hcmIOData struct for using the DI or AI values
Input: the number of the input the flowSens data is read from (ex AI 1 or DI 4)
Mode: read an analoque input or the nr of pulse of an digital input
fFlow = L/h = fOffset(value of DI or AI)*fConcersion

Conversion:
always in l/h
Liters per pulse to get the actual consumption in L/h
xx l/s @20mA - xx l/s @4ma)/16)*3600 to get the actual consumption in L/h

Offset
always in l/h
xx l/h @ 4ma is 60 l/h then ofset is 60
xx l/h @ 4ma is 0 l/h then ofset is 0
not used when using pulse mode then Offset =  0

ex:
flowmeter:  l/min @4ma  = 0
            0 l/min = 0 l/h;

            l/min @20ma = 40 l/min
            40 l/min = (40*60)= 2400 l/h

            range =  2400 - 0 = 2400 l/h
            (2400 l/h)/16 mA =  150 l/h per mA
            conversion = 150
            ofset = 0

ex2:
flowmeter:  2000 pulses/ml
            ml/pulse = 1/2000
            l/pulse = 1000/2000
*/

void _InitFlowSens(unsigned char ModuleNr, struct cmFlowSens* B, struct hcmIOData *C, struct
cmInternComSend *D, unsigned char Input, bool Mode, float Conversion, float Offset){
    B->stSet.ucModuleNr = ModuleNr;
    B->stSet.IO = C;
    B->stSet.pInternComSend = D;
    B->stSet.ucInputNr = Input;
    B->stSet.ucInputMode = Mode;
    B->stSet.fConversion = Conversion;
    B->stSet.fOffset = Offset;
    B->stData.bConnectionError = 0;
    B->stData.fFlow = 0;
}


// send datastruct as array over tcp ip to the central controller
void  SendDataFlowSens(struct cmFlowSens *B){
    if (B->stSet.pInternComSend != 0){
        unsigned char x = 0;
        if(B->stSet.pInternComSend->stSet.bPremToSend == 1){
            char b[200];//[sizeof(B->stData)+3];
            b[0] = 3;
            b[1] = 3;
            b[2] = B->stSet.ucModuleNr;
            memcpy(&b[3], &(B->stData), sizeof(B->stData));
            _SendDataInternComSend(B->stSet.pInternComSend, b, (sizeof(B->stData)+3));
        }
    }
}

//read the flowrate from the flow sensor
void _ReadFlowSens(struct cmFlowSens *B){
    if(B->stSet.ucInputMode == 0){
         float temp = ((B->stSet.IO->usiAI[B->stSet.ucInputNr]) & 0xFFE0);
```

```
          temp = temp/744; //form int value to 0-20mA
          temp = temp-4; //starts from 4ma
          if(temp < 0)B->stData.bConnectionError =1;
          else B->stData.bConnectionError = 0;
          temp = temp * (B->stSet.fConversion);
          B->stData.fFlow = B->stSet.fOffset + temp;
     }
     else if (B->stSet.ucInputMode == 1){
         B->stData.fFlow = B->stSet.fOffset + 36000*((B->stSet.IO-
>usiCounterRisingEdgePerSec[B->stSet.ucInputNr])* B->stSet.fConversion);
     }
}

//fFlow (L/h) = fOffset(value of DI or AI)*fConcersion
void  UpdateFlowSens(struct cmFlowSens* B){
     ReadFlowSens(B);
     _SendDataFlowSens(B);
}
```

# CM_PRESS_SENS_LIB.c

```
//comment
#include "CM PRESS SENS LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

void _InitPress(unsigned char ModuleNr, struct cmPress *B, struct hcmIOData *C, struct
cmInternComSend *D,  unsigned char AI){
    B->stSet.ucModuleNr = ModuleNr;
    B->stSet.IO = C;
    B->stSet.pInternComSend = D;
    B->stSet.ucAInr = AI;
    B->stData.fPressure = 0;
    B->stData.bConnectionError = 0;
}

void _SendDataPress(struct cmPress *B){
    if (B->stSet.pInternComSend != 0){
        unsigned char x = 0;
        if(B->stSet.pInternComSend->stSet.bPremToSend == 1){
            char b[200];//[sizeof(B->stData)+3];
            b[0] = 3;
            b[1] = 1;
            b[2] = B->stSet.ucModuleNr;
            memcpy(&b[3], &(B->stData), sizeof(B->stData));
            _SendDataInternComSend(B->stSet.pInternComSend, b, (sizeof(B->stData)+3));
        }
    }
}

void _ReadDataPress(struct cmPress *B){
    if (B->stSet.IO != 0){
        float temp = B->stSet.IO->usiAI[B->stSet.ucAInr];
        temp /= 744;
        temp -= 4;
        if(temp < 0) B->stData.bConnectionError = 1;
        else B->stData.bConnectionError = 0;
        temp *= 1.5625;
        B->stData.fPressure = temp;
    }
}

void _UpdatePress(struct cmPress *B){
    _ReadDataPress(B);
    _SendDataPress(B);
}
```

# CM_OIL_QUALITY_LIB.c

```
#include "CM OIL QUALITY LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

void _InitOilQua(unsigned char ModuleNr, struct cmOilQua *B, struct hcmDataSetRSMod *C, struct
hcmIOData *D, struct cmInternComSend *E, unsigned char DevNr, unsigned char AInr1, unsigned
char AInr2){
    B->stSet.ucModuleNr = ModuleNr;
    B->stSet.pModBus = C;
    B->stSet.pIO = D;
    B->stSet.pInternComSend = E;
    B->stSet.ucDevNr = DevNr;
    B->stSet.ucAInr[0] = AInr1;
    B->stSet.ucAInr[1] = AInr2;
    B->stSet.bUpdateDone = 1;
    B->stData.fOilTemp = 0;
    B->stData.fAmbientTemp = 0;
    B->stData.fOilQuality = 0;
    B->stSet.bMessageSend = 0;
    B->stSet.ucErrorCount = 0;
    B->stData.bConnectionError = 0;
}

void _ReadDataOilQua(struct cmOilQua *B){

    if (B->stSet.bUpdateDone == 0){
        float temp = ((B->stSet.pIO->usiAI[B->stSet.ucAInr[0]]) & 0xFFE0);
        temp = temp/744; //form int value to 0-20mA
        temp = temp-4; //starts form 4ma


        float temp2 = ((B->stSet.pIO->usiAI[B->stSet.ucAInr[1]]) & 0xFFE0);
        temp2 = temp2/744; //form int value to 0-20mA
        temp2 = temp2-4; //starts form 4ma

        if((temp<0)||(temp2<0)) B->stData.bConnectionError = 1;
        else B->stData.bConnectionError = 0;

        B->stData.fOilTemp = (temp * 10)-30;
        B->stData.fOilQuality = (temp2 * 5)-20;
        B->stSet.bUpdateDone = 1;
     }

}

void  ReqDataOilQua(struct cmOilQua *B){
    if (B->stSet.bUpdateDone == 0){
        if(B->stSet.bMessageSend == 0){
            if(_CreateMessageMod(B->stSet.pModBus, B->stSet.ucDevNr, 0x04, 0x00, 0x00, 0x00,
0x03))B->stSet.bMessageSend = 1;
        }
    }
}

void _RecDataOilQua(struct cmOilQua *B){
    unsigned short int x = 0;
    if((B->stSet.bMessageSend == 1) && (B->stSet.bUpdateDone == 0)){
        if ((B->stSet.pModBus->bNewRXMessage == 1)  && (B->stSet.pModBus->ucTXDevAddress == B-
>stSet.ucDevNr) && (B->stSet.pModBus->ucTXFunctionCode == 0x04)){

            for(x =0; x < B->stSet.pModBus->ucRxBufferLength; x++){
                if(B->stSet.pModBus->usiRXRegAddress[x] == 0x00){
                    B->stData.fOilTemp = (signed short int)(B->stSet.pModBus-
>usiRxRegValue[x]);
                    B->stData.fOilTemp /= 100;
                }
                if(B->stSet.pModBus->usiRXRegAddress[x] == 0x01){
                    B->stData.fAmbientTemp = (signed short int)(B->stSet.pModBus-
>usiRxRegValue[x]);
                    B->stData.fAmbientTemp /= 100;
                }
                if(B->stSet.pModBus->usiRXRegAddress[x] == 0x02){
                    B->stData.fOilQuality = (signed short int)(B->stSet.pModBus-
>usiRxRegValue[x]);
```

```
                    B->stData.fOilQuality /= 100;
                }
            }
            B->stSet.bUpdateDone = 1;
            B->stSet.bMessageSend = 0;
            B->stSet.ucErrorCount = 0;
            B->stData.bConnectionError = 0;
        }
        else if((B->stSet.pModBus->bExpectRecMessageError == 1) || ((B->stSet.pModBus-
>bNewRXMessage == 1)  && ((B->stSet.pModBus->ucTXDevAddress != B->stSet.ucDevNr) || (B-
>stSet.pModBus->ucTXFunctionCode != 0x04)))){
            B->stSet.bMessageSend = 0;
            B->stSet.bUpdateDone = 1;
            if(B->stSet.ucErrorCount < 5) B->stSet.ucErrorCount++;
            else{
                B->stSet.bMessageSend = 0;
                B->stData.fOilTemp = (-4*10)-30;
                B->stData.fAmbientTemp = (-4*10)-30;
                B->stData.fOilQuality = (-4*5)-20;
                B->stData.bConnectionError = 1;
            }
        }
    }
}

void  SendDataOilQua(struct cmOilQua *B){
    if (B->stSet.pInternComSend != 0){
        unsigned char x = 0;
        if(B->stSet.pInternComSend->stSet.bPremToSend == 1){
            char b[200];//[sizeof(B->stData)+3];
            b[0] = 3;
            b[1] = 2;
            b[2] = B->stSet.ucModuleNr;
            memcpy(&b[3], &(B->stData), sizeof(B->stData));
            _SendDataInternComSend(B->stSet.pInternComSend, b, (sizeof(B->stData)+3));
        }
    }
}

void _UpdateOilQua(struct cmOilQua *B){
    if(B->stSet.pIO != 0){
        _ReadDataOilQua(B);
    }
    if(B->stSet.pModBus != 0){
        _ReqDataOilQua(B);
        _RecDataOilQua(B);
    }
    _SendDataOilQua(B);
}
```

# CM_J1939_LIB.c

```
#define SPN_BUFFER_WIDTH 7
#include "CM_J1939_LIB.h";
//
//designer original scource: Arjan Ippel
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

//init
void _InitJ1939(unsigned char modelNr, struct cmJ1939 * B, struct hcmDataSetCan * C, struct
cmInternComSend *D){
    B->stSet.pCanbus = C;
    B->stSet.SPNBufferWidth = SPN_BUFFER_WIDTH;
    B->stSet.pInternComSend = D;
    B->stSet.ucModuleNr = modelNr;
    _InitSPN(B);
}

//load user defined SPN data. only spn data in this array will be recognised by CM J1939
void _InitSPN(struct cmJ1939 * B){
    unsigned char x = 0;
    for(x=0; x<B->stSet.SPNBufferWidth; x++){
        B->stData.stSPN[x].stSPNData.usiPGN = 0;
        B->stData.stSPN[x].stSPNData.usiSPN = 0;
        B->stData.stSPN[x].stSPNData.fValue = 0;
        B->stData.stSPN[x].stSPNData.ucScource = 0;
        B->stData.stSPN[x].stSPNSet.ucStartByte = 0;
        B->stData.stSPN[x].stSPNSet.ucEndByte = 0;
        B->stData.stSPN[x].stSPNSet.ucOffset = 0;
        B->stData.stSPN[x].stSPNSet.fMultiply = 0;
    }
    B->stData.usiSpnBufferWidth = SPN_BUFFER_WIDTH;

    //vehicle speed
    B->stData.stSPN[0].stSPNData.usiPGN = 65132;
    B->stData.stSPN[0].stSPNData.usiSPN = 1624;
    B->stData.stSPN[0].stSPNData.ucScource = 1;

    B->stData.stSPN[0].stSPNSet.ucStartByte = 7;
    B->stData.stSPN[0].stSPNSet.ucEndByte = 8;
    B->stData.stSPN[0].stSPNSet.ucOffset = 0;
    B->stData.stSPN[0].stSPNSet.fMultiply = 0.00390625;

    //engine torque
    B->stData.stSPN[1].stSPNData.usiPGN = 61444;
    B->stData.stSPN[1].stSPNData.usiSPN = 513;
    B->stData.stSPN[1].stSPNData.ucScource = 1;

    B->stData.stSPN[1].stSPNSet.ucStartByte = 3;
    B->stData.stSPN[1].stSPNSet.ucEndByte = 3;
    B->stData.stSPN[1].stSPNSet.ucOffset = -125;
    B->stData.stSPN[1].stSPNSet.fMultiply = 1;


    //Engine Coolant Temperature
    B->stData.stSPN[2].stSPNData.usiPGN = 65262;
    B->stData.stSPN[2].stSPNData.usiSPN = 110;
    B->stData.stSPN[2].stSPNData.ucScource = 1;

    B->stData.stSPN[2].stSPNSet.ucStartByte = 1;
    B->stData.stSPN[2].stSPNSet.ucEndByte = 1;
    B->stData.stSPN[2].stSPNSet.ucOffset = -40;
    B->stData.stSPN[2].stSPNSet.fMultiply = 1;

    //Engine Fuel Temperature
    B->stData.stSPN[3].stSPNData.usiPGN = 65262;
    B->stData.stSPN[3].stSPNData.usiSPN = 174;
    B->stData.stSPN[3].stSPNData.ucScource = 1;

    B->stData.stSPN[3].stSPNSet.ucStartByte = 2;
    B->stData.stSPN[3].stSPNSet.ucEndByte = 2;
    B->stData.stSPN[3].stSPNSet.ucOffset = -40;
    B->stData.stSPN[3].stSPNSet.fMultiply = 1;

    //Engine Fuel Delivery Pressure
    B->stData.stSPN[4].stSPNData.usiPGN = 65263;
```

```
    B->stData.stSPN[4].stSPNData.usiSPN = 94;
    B->stData.stSPN[4].stSPNData.ucScource = 1;

    B->stData.stSPN[4].stSPNSet.ucStartByte = 1;
    B->stData.stSPN[4].stSPNSet.ucEndByte = 1;
    B->stData.stSPN[4].stSPNSet.ucOffset = 0;
    B->stData.stSPN[4].stSPNSet.fMultiply = 4;

    //Engine Speed
    B->stData.stSPN[5].stSPNData.usiPGN = 61444;
    B->stData.stSPN[5].stSPNData.usiSPN = 190;
    B->stData.stSPN[5].stSPNData.ucScource = 1;

    B->stData.stSPN[5].stSPNSet.ucStartByte = 4;
    B->stData.stSPN[5].stSPNSet.ucEndByte = 5;
    B->stData.stSPN[5].stSPNSet.ucOffset = 0;
    B->stData.stSPN[5].stSPNSet.fMultiply = 0.125;

    //oil pressure in engine
    B->stData.stSPN[6].stSPNData.usiPGN = 65263;
    B->stData.stSPN[6].stSPNData.usiSPN = 100;
    B->stData.stSPN[6].stSPNData.ucScource = 1;


    B->stData.stSPN[6].stSPNSet.ucStartByte = 4;
    B->stData.stSPN[6].stSPNSet.ucEndByte = 4;
    B->stData.stSPN[6].stSPNSet.ucOffset = 0;
    B->stData.stSPN[6].stSPNSet.fMultiply = 4;


}

void _CalcValueJ1939(unsigned char ucData[8], struct SPN * C)
{
    float fReturnValue = 0;
    int lIndex = 0;
    for(lIndex = (C->stSPNSet.ucStartByte)-1; lIndex <= ((C->stSPNSet.ucEndByte)-1); lIndex++)
    {
        /*testbool[4] = testbool[4] +1;
        testbool[5] = 33;
        testbool[5] = lIndex + C->stSPNSet.ucStartByte; */
        fReturnValue +=  ((long unsigned int)ucData[lIndex]) * pow(0x100,(lIndex - (C-
>stSPNSet.ucStartByte-1)));
    }
    fReturnValue += C->stSPNSet.ucOffset;
    //fReturnValue *= 100;
    fReturnValue *= C->stSPNSet.fMultiply;
    C->stSPNData.fValue = fReturnValue;
}

//read a J1939 mesage from the Canbus
void _ReadMessageJ1939(struct cmJ1939 * B){
    if((B->stSet.pCanbus)->bNewRXMessage){
        unsigned char x = 0;
        CANBASE_MSG tempMessage;
        unsigned char PGNdata[8] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
        unsigned short int PGNnr = 0;
        unsigned short int PGNPriority = 0;
        unsigned short int PGNscource = 0;

        //testbool[0] = testbool[0] + 1;
        for(x=0; x < ((B->stSet.pCanbus)->ucRxbufferNr); x++){
            tempMessage.Frame = B->stSet.pCanbus->cmCanRxMessages[x].Frame;
            tempMessage.MsgID = (B->stSet.pCanbus)->cmCanRxMessages[x].MsgID;
            tempMessage.DatA = (B->stSet.pCanbus)->cmCanRxMessages[x].DatA;
            tempMessage.DatB = (B->stSet.pCanbus)->cmCanRxMessages[x].DatB;

            PGNPriority        = (int)(tempMessage.MsgID & 0xFF000000) >> 24;
            PGNnr            = (int)(tempMessage.MsgID & 0xFFFF00) >> 8;
            PGNscource   = (unsigned char)(tempMessage.MsgID & 0xFF);

            PGNdata[0]        = (unsigned char)((tempMessage.DatA >> 0)& 0xFF);
            PGNdata[1]        = (unsigned char)((tempMessage.DatA >> 8)     & 0xFF);
            PGNdata[2]        = (unsigned char)((tempMessage.DatA >> 16)    & 0xFF);
            PGNdata[3]        = (unsigned char)((tempMessage.DatA >> 24)    & 0xFF);
            PGNdata[4]        = (unsigned char)((tempMessage.DatB >> 0)     & 0xFF);
            PGNdata[5]        = (unsigned char)((tempMessage.DatB >> 8)     & 0xFF);
```

```
            PGNdata[6]        = (unsigned char)((tempMessage.DatB >> 16)    & 0xFF);
            PGNdata[7]        = (unsigned char)((tempMessage.DatB >> 24)    & 0xFF);
            for(x=0; x< (B->stSet.SPNBufferWidth); x++){
            //testbool[2] = testbool[2] + 1;
                if( ((B->stData.stSPN[x].stSPNData.usiPGN) == PGNnr) && (B-
>stData.stSPN[x].stSPNData.ucScource == PGNscource)){
                    //testbool[3] = testbool[3] + 1;
                    _CalcValueJ1939(PGNdata, &(B->stData.stSPN[x]));
                }
            }
        }
    }
}

//write a J1939 message in canformat. not used
void  WriteMessageJ1939(struct cmJ1939 * B, unsigned char data[], unsigned char  dataLength,
unsigned short int PGN, unsigned char  scource){

    unsigned long int  id =0;
    id = 0;
    id += 0x18;
    id <<= 16;
    id += PGN;
    id <<= 8;
    id += scource;
    _FillTxBufferCan((B->stSet.pCanbus), id, data, 8);
}

//send the canmessage
void  SendDataJ1939(struct cmJ1939 *B){
    if (B->stSet.pInternComSend != 0){
        unsigned char x = 0;
        if(B->stSet.pInternComSend->stSet.bPremToSend == 1){
            for(x = 0; x <B->stSet.SPNBufferWidth; x++){
                char b[200];//[sizeof(B->stData)+3];
                b[0] = 3;
                b[1] = 4;
                b[2] = B->stSet.ucModuleNr;
                b[3] = x;
                memcpy(&b[4], &(B->stData.stSPN[x].stSPNData), sizeof(B->stData));
                _SendDataInternComSend(B->stSet.pInternComSend, b, (sizeof(B->stData)+4));
            }
        }
    }
}


void  UpdateJ1939(struct cmJ1939 * B){
     ReadMessageJ1939(B);
    _SendDataJ1939(B);
}
```

# CM_INTERN_COM_SEND_LIB.c

```
#include "CM INTERN COM SEND LIB.h"
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

//init
void _InitInternComSend(struct cmInternComSend* B, struct cmHardwareDataSetTcpIp* C){
    B->stSet.TCPIP = C;
    B->stSet.bPremToSend = 0;
    B->stSet.bSendEndMes = 0;
}

//this function is called by every CM and EM when the want to send there datastruct
bool _SendDataInternComSend(struct cmInternComSend* B, char a[], unsigned char b){
     if(_FillTXbufferTcpIp(B->stSet.TCPIP, a, b)) return 1;
     return 0;
}

//if startmessage has bin received signal other CM and EM to send there data struct
//if they're done send a endmessage
void _RecDataInternComSend(struct cmInternComSend* B){
    if((B->stSet.bPremToSend == 1) /*|| (B->stSet.bSendEndMes == 1)*/){
        B->stSet.bPremToSend = 0;
        char temp[3] = {0, 0, 0};
        if (_SendDataInternComSend(B, temp, 3));/* B->stSet.bSendEndMes = 0;*/
        /*else B->stSet.bSendEndMes = 1; */
    }
    if(B->stSet.TCPIP->ucRXBuffernumber > 0){
        if((B->stSet.TCPIP->ucRXBuffer[0][0] == 0) && (B->stSet.TCPIP->ucRXBuffer[0][1] == 0)
&& (B->stSet.TCPIP->ucRXBuffer[0][2] == 0)){
            B->stSet.bPremToSend = 1;
        }
    }
}

void  UpdateInternComSend(struct cmInternComSend* B){
    _RecDataInternComSend(B);
}
```

# CM_INTERN_COM_REC_LIB.c

```
//asdasdadfs
#include "CM INTERN COM REC LIB.h"
struct cmInternComRec stcmInternComRec;
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016


//init
//add connection in the forloop
void _InitInternComRec(struct cmInternComRec* B, struct hcmDataSetSer* C, struct
cmHardwareDataSetTcpIp *D[], unsigned char connections){
    unsigned char x = 0;
    if (connections > 15) connections = 15;
    for(x = 0; x<connections; x++){
        B->stSet.TCPIP[x] = D[x];
    }
    B->stSet.pSer = C;
    B->stSet.ucNrOfConnections = connections;
    B->stSet.bReqData = 1;
    B->stSet.ucX = 0;
    B->stSet.bRXDelay = 0;
    B->stSet.ulRXDelayState = 0;
    B->stSet.ucRXdelayCounter = 0;
}

//sens startmessage to slave controller
void _ReqDataInternComRec(struct cmInternComRec* B){
    char a[6] = {0, 0, 0, 0, 0, 0};
    unsigned char x = 0;
    unsigned char y = 0;
    if(B->stSet.bReqData == 1){
        B->stSet.ucX++;
        if(B->stSet.ucX >=  B->stSet.ucNrOfConnections) B->stSet.ucX = 0;
        B->stSet.bReqData = 0;
        _FillTXbufferTcpIp(B->stSet.TCPIP[B->stSet.ucX], a, 3);
    }
}

//receive message form slave controller
//if stopmessage has bin received sens data to pc en let _ReqDataInternComRec send a new
startmessege to the other controller

void _RecDataInternComRec(struct cmInternComRec* B){
    unsigned char x = B->stSet.ucX;
    unsigned char y = 0;
        if(B->stSet.TCPIP[x]->ucRXBuffernumber > 0){
            B->stSet.ucRXdelayCounter = 0;
            for(y=0; y<(B->stSet.TCPIP[x]->ucRXBuffernumber); y++){
                unsigned char temp = (B->stSet.TCPIP[x]->ucRXBuffer[y][2])-1;
                unsigned char temp2 = (B->stSet.TCPIP[x]->ucRXBuffer[y][3]);

                switch(B->stSet.TCPIP[x]->ucRXBuffer[y][0]){
                    case 0:
                        if((B->stSet.TCPIP[x]->ucRXBuffer[y][1] == 0) && (B->stSet.TCPIP[x]-
>ucRXBuffer[y][2] == 0)){
                            B->stSet.bReqData = 1;
                        }
                    case 3:
                        switch(B->stSet.TCPIP[x]->ucRXBuffer[y][1]){
                            case 1:
                                memcpy(&(B->stData.stPressData[temp]),
                                &(B->stSet.TCPIP[x]->ucRXBuffer[y][3]),
                                sizeof(B->stData.stPressData[temp]));
                            break;
                            case 2:
                                memcpy(&(B->stData.stOilQuaData[temp]),
                                &(B->stSet.TCPIP[x]->ucRXBuffer[y][3]),
                                sizeof(B->stData.stOilQuaData[temp]));
                            break;
                            case 3:
                                memcpy(&(B->stData.stFlowData[temp]),
                                &(B->stSet.TCPIP[x]->ucRXBuffer[y][3]),
                                sizeof(B->stData.stFlowData[temp]));
                            break;
```

```
                            case 4:
                                memcpy(&(B->stData.stSPNdata[temp][temp2]),
                                &(B->stSet.TCPIP[x]->ucRXBuffer[y][4]),
                                sizeof(B->stData.stSPNdata[temp][temp2]));
                            break;
                        }
                    break;
                    case 4:
                        switch(B->stSet.TCPIP[x]->ucRXBuffer[y][1]){
                            case 1:
                                memcpy(&(B->stData.stFuelConsumptionData[temp]),
                                &(B->stSet.TCPIP[x]->ucRXBuffer[y][3]),
                                sizeof(B->stData.stFuelConsumptionData[temp]));
                                  SendPcDataInternComRec(B);
                            break;

                            case 2:
                            break;
                        }
                    break;
                }

            }
        }
        else{
            B->stSet.bRXDelay = DelayUp(&(B->stSet.ulRXDelayState), 10, !(B->stSet.bRXDelay));
            if(B->stSet.bRXDelay){
                B->stSet.ucRXdelayCounter = B->stSet.ucRXdelayCounter +1;
            }
            if (B->stSet.ucRXdelayCounter > 10){
                B->stSet.ucRXdelayCounter = 0;
                B->stSet.bReqData = 1;
            }
        }
}

//data to the pc in JSON string
void  SendPcDataInternComRec(struct cmInternComRec* B){
    memset(B->stData.ucDataString,0,sizeof(B->stData.ucDataString));
    unsigned char ucftemp[400];
    unsigned short int x =0;
    unsigned short int y =0;

    strcat (B->stData.ucDataString, "{");
    strcat (B->stData.ucDataString, "\"stPressData\":[");

    for(x = 0;  x<10; x++){
        sprintf(ucftemp, "{\"fPressure\":%'.2f, \"bConnectionError\":%d}",
                B->stData.stPressData[x].fPressure, B-
>stData.stPressData[x].bConnectionError);
        strcat (B->stData.ucDataString, ucftemp);
        if(x<9) strcat (B->stData.ucDataString, ",");
    }
    strcat (B->stData.ucDataString, "],");
    strcat (B->stData.ucDataString, "\"stOilQuaData\":[");
    for(x = 0;  x<10; x++){
        sprintf(ucftemp, "{\"fOilTemp\":%'.2f, \"fOilQuality\":%'.2f, \"fAmbientTemp\":%'.2f,
\"bConnectionError\":%d}",
                B->stData.stOilQuaData[x].fOilTemp,
                B->stData.stOilQuaData[x].fOilQuality,
                B->stData.stOilQuaData[x].fAmbientTemp,
                B->stData.stOilQuaData[x].bConnectionError);
        strcat (B->stData.ucDataString, ucftemp);
        if(x<9) strcat (B->stData.ucDataString, ",");
    }
    strcat (B->stData.ucDataString, "],");
    strcat (B->stData.ucDataString, "\"stFlowData\":[");
    for(x = 0;  x<10; x++){
        sprintf(ucftemp, "{\"fFlow\":%'.2f, \"bConnectionError\":%d}",
                B->stData.stFlowData[x].fFlow,
                B->stData.stFlowData[x].bConnectionError);
        strcat (B->stData.ucDataString, ucftemp);
        if(x<9) strcat (B->stData.ucDataString, ",");
    }
    strcat (B->stData.ucDataString, "],");
    strcat (B->stData.ucDataString, "\"stSPNdata\":[[");
    for(x = 0; x<2; x++){
```

```
        for(y = 0;  y<20; y++){
            sprintf(ucftemp, "{\"usiPGN\":%d, \"usiSPN\":%d, \"ucScource\":%d,
\"fValue\":%'.2f}",
                    B->stData.stSPNdata[x][y].usiPGN,
                    B->stData.stSPNdata[x][y].usiSPN,
                    B->stData.stSPNdata[x][y].ucScource,
                    B->stData.stSPNdata[x][y].fValue);
            strcat (B->stData.ucDataString, ucftemp);
            if(y<19) strcat (B->stData.ucDataString, ",");
        }
        if(x<1) strcat (B->stData.ucDataString, "], [");
        else strcat (B->stData.ucDataString, "]");
    }
    strcat (B->stData.ucDataString, "],");
    strcat (B->stData.ucDataString, "\"stFuelConsumptionData\":[");
    for(x = 0;  x<10; x++){
        sprintf(ucftemp, "{\"fFuelConsumption\":%'.2f, \"fTotalFuelConsumed\":%'.2f}",
                B->stData.stFuelConsumptionData[x].fFuelConsumption,
                B->stData.stFuelConsumptionData[x].fTotalFuelConsumed);
        strcat (B->stData.ucDataString, ucftemp);
        if(x<9) strcat (B->stData.ucDataString, ",");
    }
    strcat (B->stData.ucDataString, "]");
    strcat (B->stData.ucDataString, "}");
    strcat (B->stData.ucDataString, "\n");

    testint = strlen(B->stData.ucDataString);
    _SendCharSer(B->stSet.pSer, B->stData.ucDataString);

}



void _RunInternComRec(struct cmInternComRec* B){
    _ReqDataInternComRec(B);
    _RecDataInternComRec(B);
}
```

# EM_FUEL_CONSUMPTION.c

```
#include "EM FUEL CONSUMPTION.h"                          //asda
//designed original scource: Joris Groen
//futher implemented for SMMS and expanded by: Joris Groen
//Date: 25-08-2016

//init
void _InitFuelConsumption(unsigned char ModuleNr, struct emFuelConsumption* B,  struct
cmInternComSend *C, struct cmFlowSens* Inflow, struct cmFlowSens* Outflow,  bool Mode){
    B->stSet.ucModuleNr = ModuleNr;
    B->stSet.pInternComSend = C;
    B->stSet.pInFlow = Inflow;
    B->stSet.pOutFlow = Outflow;
    B->stSet.bMode = Mode;
    _ResetFuelConsumption(B);
}

//reset total fuel consumption
void _ResetFuelConsumption(struct emFuelConsumption* B){
    B->stData.fFuelConsumption = 0;
    B->stData.fTotalFuelConsumed = 0;
}

//calculate the fuelconsumption every 100ms
//the 36000 is to convert from l/h to l/100ms.
void _ProcessFuelConsumption(struct emFuelConsumption* B){
    if (B->stSet.bMode){
        if(B->stSet.pInFlow->stSet.IO->bChangePerSecInd){
            B->stData.fFuelConsumption = B->stSet.pInFlow->stData.fFlow;
            B->stData.fTotalFuelConsumed += (B->stData.fFuelConsumption)/36000;
        }
    }
    else{
        if(B->stSet.pInFlow->stSet.IO->bChangePerSecInd){
            B->stData.fFuelConsumption = ((B->stSet.pInFlow->stData.fFlow) - (B-
>stSet.pOutFlow->stData.fFlow));
            B->stData.fTotalFuelConsumed += (B->stData.fFuelConsumption)/36000;
        }
    }
}

//send datastruct as array over the tcp/ip
void _SendDataFuelConsumption(struct emFuelConsumption* B){
    if (B->stSet.pInternComSend != 0){
        unsigned char x = 0;
        if(B->stSet.pInternComSend->stSet.bPremToSend == 1){
            char b[200];
            b[0] = 4;
            b[1] = 1;
            b[2] = B->stSet.ucModuleNr;
            memcpy(&b[3], &(B->stData), sizeof(B->stData));
            _SendDataInternComSend(B->stSet.pInternComSend, b, (sizeof(B->stData)+3));
        }
    }
}

void _UpdateFuelConsumption(struct emFuelConsumption* B){
     SendDataFuelConsumption(B);
    _ProcessFuelConsumption(B);
}
```

## 14.8 Bijlage VSB hoofdstuk 5 Software

# 5 Software

In dit hoofdstuk worden enkele specifieke items beschreven omtrent de besturingsfilosofie van VMB, het schrijven van software en het opstellen van een I/O lijst.

## 5.1 Besturingsfilosofie VMB

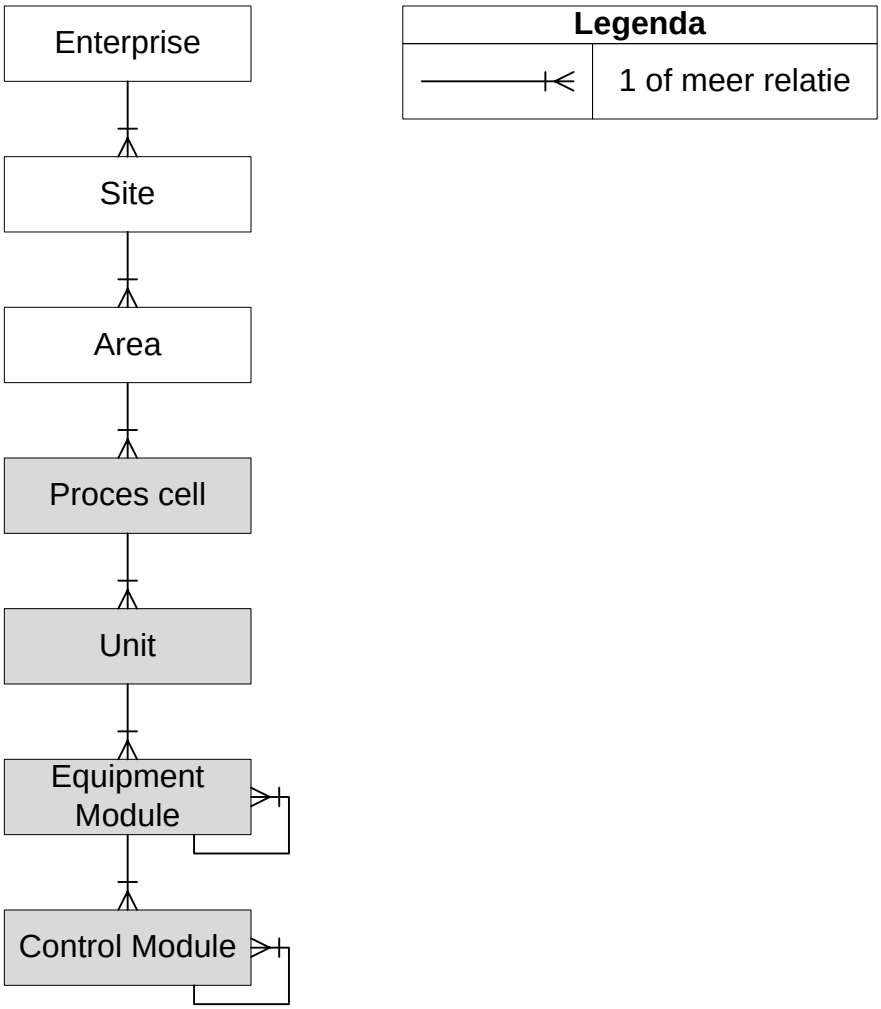
Waarom hanteren wij een gezamenlijke manier van software programmeren:

- Vereenvoudigen van overdragen software aan collega.
- Mogelijkheid om met meerdere programmeurs aan één project te werken.
- Eenduidige structuur.
- Oplossen van storingen wordt eenvoudiger.

De basis van de besturingsfilosofie bij VMB is S88.01. S88.01 is een internationale standaard voor het beschrijven van productieprocessen. S88.01 is begonnen met het bieden van standaard terminologie voor de automatisering van batch processen en modellen voor het modulair indelen van processen en recepten.

Hoewel S88.01 is ontwikkeld voor batch processen, kan het ook worden toegepast bij continue en discrete processen. Denk bij continue processen bijvoorbeeld aan olieraffinaderijen, waterzuivering, gaswinning, recycling automaterialen.
Discrete processen worden vaak gekenmerkt door individuele of een afzonderlijke productie-eenheid. Discrete processen kom je bijvoorbeeld tegen in de automobiel industrie, meubels, speelgoed, smartphones, etc.

Het fysieke model van S88.01 is weergegeven in Figuur 3.




**Figuur 3: fysieke model S88.01**

- **Enterprise:** Een organisatie die werkzaamheden coördineert tussen een of meer 'sites'. (Philips)
- **Site:** Een onderdeel van een enterprise die wordt geïdentificeerd door een fysieke, geografische of logische locatie. (Philips Eindhoven)
- **Area:** Een onderdeel van een site die wordt geïdentificeerd door een fysieke, geografische of logische locatie. (Philips afdeling R&D)
- **Process cell:** Een process cell bevat alle apparatuur om een of meerdere batch(es) te maken. (process installatie)
- **Unit:** Een unit voert acties uit op een batch of een gedeelte van een batch, bestaat uit een of meerdere modules en werkt onafhankelijk van andere units. (tank met warmtewisselaar en invoer en uitvoer leidingen)
- **Equipment module:** Een equipment module is een logische samenstelling van een of meerdere control modules die gezamenlijk verantwoordelijk zijn voor een process actie. (warmtewisselaar, weegtank, gekoppelde bewegingen)

- **Control module:** Een control module is het laagste level in het fysieke model waarmee basic control gerealiseerd kan worden. Basic control:
    - Mode auto/hand
    - Regelingen
    - Alarm/meldingen
    - Commando's start/stop/open sluit
    - Etc.

    Een control module voert nooit proces acties uit. Voorbeelden van control modules zijn klep, motor.

Naast deze het fysieke model zijn er nog **common resources**. Common resources zijn meestal equipment modules of control modules die door meerdere units kunnen worden gebruikt. Binnen de common resources maken we weer onderscheid tussen **exclusive-use resources** en **shared-use resources**.

Exclusive-use resources zijn resources die maar door één (of in iedergeval beperkt(!)) unit tegelijk kan worden gebruikt. Voorbeelden van Exclusive-use resources zijn: kleppen/motoren die door meerdere units aangestuurd kunnen worden.

Voor exclusive-use resources is arbitrage nodig om te bepalen wie de resource mag gebruiken. Voorbeelden van arbitrage zijn:

- Wie het eerste komt wie het eerste maalt
- Meerdere verzoeken,
    - Reserveringensysteem
    - Prioriteiten systeem
    - Rule based
- Pre-emption (onderbreking/voorrang)
- Scheduling (tijd gereleateerd)

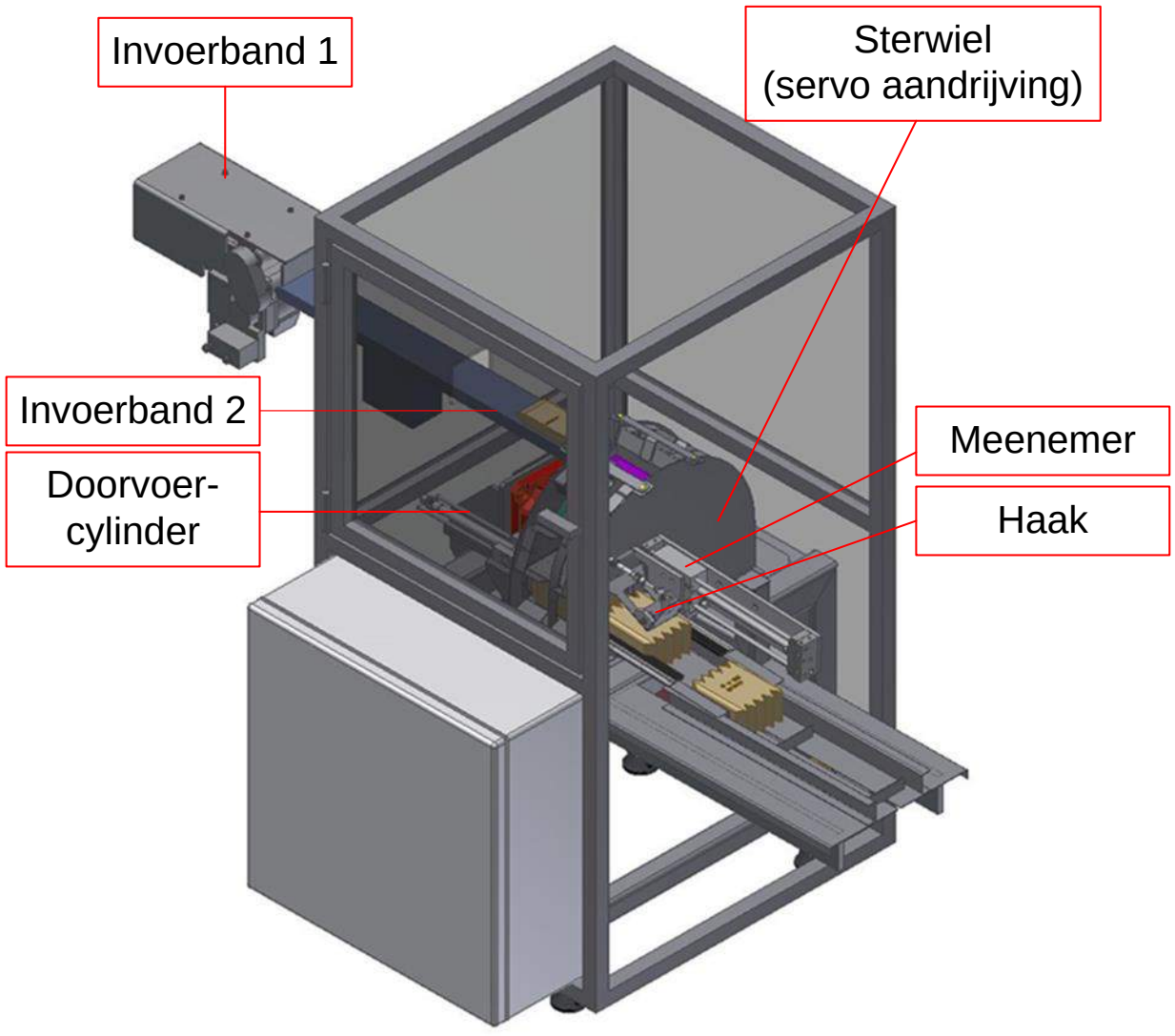
Shared-use resources zijn common resources die door meerdere units tegelijk kunnen worden gebruikt (zonder beperking!). Voorbeeld: perslucht installatie, grondstoffen toevoer, koeling/verwarming, etc.

Uit de praktijk blijkt dat de software projecten bij VMB vooral op of Unit niveau zijn (één PLC, één machine). Echter zijn de start projecten geschikt voor het aansturen van een process cell (één PLC, meerdere machines)

Om de filosofie te verduidelijk te maken we gebruik van een voorbeeld project, de 'Wheely'. De Wheely is een machine voor het verzamelen van vijf enkele geflowwrapte producten om deze vervolgens 90 graden te roteren en te transporteren naar de volgende flow wrapper waar deze vijf producten tezamen worden verpakt als multipack.

De producten worden aangevoerd vanuit een flow wrapper en verenkeld verder getransporteerd middels twee invoerbanen voor de Wheely. De producten worden

verzameld in één van de acht producthouders van het sterwiel. Nadat vijf producten zijn verzameld draait het sterwiel 45 graden. Zodra de producten 90 graden zijn geroteerd, drukt een doorvoercylinder de vijf producten uit de producthouder op een deadplate. De vijf producten worden vervolgens door een tweetal cylinders op het juiste moment getransporteerd naar de invoerbaan van de multipack flow wrapper, zodat de producten per vijf stuks worden verpakt.



Als we het fysieke model loslaten op de Wheely krijgen we het volgende:

Unit: Wheely
- Equipment modules:
  - Invoer
    - Control modules:
      - Motor
      - FO met Motor
  - Draaien
    - Control modules:
      - Servoregelaar met servomotor
      - Doorvoercylinder (pneumatische cylinder)
  - Uitvoer
    - Control modules:
      - Meenemer (pneumatische cylinder)
      - Haak (pneumatische cylinder)

Om tot dit model te komen hanteren we acht criteria:
- **Doel:** alle onderdelen van de module moeten hetzelfde doel dienen.
- **Gebruik:** de interactie met andere modules; wordt het gebruikt door andere modules of werkt het onafhankelijk?
- **Portabiliteit:** de mogelijkheden om de module elders te gebruiken (dupliceren/verplaatsen)
- **Flexibiliteit:** zorg dat de module geen beperking is in het totale ontwerp.
- **Onafhankeljk:** zorg voor op zichzelf staande modules
- **Uitbreidbaar:** spreek voor zich
- **Isolatie:** een module moet "zijn eigen boontjes kunnen doppen".
- **Fysiek beperkt:** bekijk met welke fysieke beperkingen rekening moet worden gehouden.

Het fysieke model van S88.01 wordt zo veel mogelijk toegepast in de software projecten.