Beuk engineering

Technische automatisering | Besturingstechniek | Elektrotechniek

SCRIPTIE

Afstudeerstage

Dennis Slingerland
09003851

4-6-2014

Bestemd voor: Gecommiteerde

Samenvatting

Deze scriptie heeft betrekking tot de afstudeerstage van Dennis Slingerland bij Beuk engineering, in Leimuiden.
In deze scriptie wordt de aandacht gevestigd op het onderzoeken, opbouwen en het uitwerken van de afstudeeropdrachten met betrekking tot industriële automatisering. Hierbij zal als eerst een omschrijving gegeven worden over het bedrijf; wat voor bedrijf betreft het, wat zijn de werkzaamheden, waar is Beuk engineering in gespecialiseerd. Daarbij wordt er iets verteld over het tot stand komen van de opdrachten; hoe is Beuk engineering aan deze opdrachten gekomen en waarom.
Daarna zullen beide opdrachten uitgebreid en inhoudelijk uitgelegd worden. Bij deze uitleg zal als eerst verteld worden over welke onderzoeken en analyses er verricht zijn om opdrachten te onderbouwen. Aansluitend zal vertelt worden hoe de opdrachten opgebouwd zijn en hoe deze functioneren in de praktijk.
Aan het begin van de stage werd in het plan van aanpak een aantal verwachte resultaten opgesteld met betrekking tot de opdrachten. In deze scriptie worden de verwachte resultaten vergeleken met de uiteindelijke resultaten. Uit deze resultaten worden een aantal conclusies getrokken met betrekking tot de opdracht. Aan de hand van de getrokken conclusies zijn er een aantal aanbevelingen opgesteld met betrekking tot de opdrachten.

Inhoud

1. Inleiding

Beuk engineering richt zich binnen de markt op het gebied van technische automatisering, besturingstechniek en elektrotechniek. Het bedrijf automatiseert en onderhoudt al 20 jaar lang machines en installaties. Beuk engineering werkt voornamelijk voor machinebouwers waarbij het bedrijf zorgt voor de besturing en de installatie van de machines. Qua werkzaamheden houdt Beuk zich bezig met engineering, tekenwerk, elektrotechniek, paneel- kastenbouw, PLC SCADA datalog en industrieel onderhoud.

De eerst afstudeeropdracht betrof het maken van een library waarbij veel gebruikte functies binnen Beuk engineering in opgeslagen zou worden. Op het gebied van industriële automatisering wordt er veel gebruik gemaakt van PLC's. Binnen deze techniek bestaat er een breed arsenaal aan controllers en hun bijbehorende applicatie software´s. Voor een bedrijf zoals Beuk is het van belang dat voor elke machine die gebouwd wordt, een geschikte PLC beschikbaar is. Daarbij moet de PLC voldoen die aan de eisen van de klant voor een minimale prijs. Daarbij moet niet vergeten worden dat niet elke machine op dezelfde manier aangestuurd wordt. Het is daarom aan de leveranciers van PLC's van belang om in te spelen op deze eisen. Deze zijn continue bezig met het verbeteren van hun applicatie softwares en de controllers. Binnen Beuk wordt veel gebruik gemaakt van Unitronics PLC's. Daarbij heeft Unitronics vrij recent een PLC op de markt gebracht met een applicatie software waarbij het mogelijk is data functiebouwstenen op te bouwen. Deze mogelijkheid is gunstig voor een programmeur omdat deze slechts éénmalig een stukje software hoeft op te bouwen waarna deze zeer gemakkelijk in andere projecten gebruikt kan worden.

De tweede opdracht betrof het onderzoeken naar dataoverdracht tussen PC en PLC. Het doel van deze opdracht was een tekst bestand van een PC over te dragen aan een PLC waarbij deze het bestand omzet in data. De PLC kan de data aanpassen en de data als nieuw bestand terug plaatsen op de PC. Een klein deel van de software binnen Beuk wordt uitbesteed aan specialisten. Dit wordt gedaan omdat de betreffende programmeurs van sommige situaties weinig kennis in huis hebben een geen tijd om zich daar in te verdiepen.

2. Methodologie

In dit hoofdstuk zullen beide afstudeeropdrachten uitgebreid aan de orde komen. Aller eerst wordt in de hoofdstukken van bij beide opdrachten uitgebreid het doel van de opdracht omschreven met daarbij de vragen; hoe is men aan deze opdracht gekomen, waarom is deze opdracht uitgewerkt, ect. Daarna worden de betreffende onderzoeken verwoord die gedaan zijn op het gebied van de opdrachten. Hierin wordt de betreffende informatie vermeld en uitgebreid uitgelegd. Daarna wordt de opbouw van de betreffende soft/hardware uitgelegd. Let wel, binnen deze afstudeerstage betrof het voornamelijk software matige opdrachten. Na de opbouw zal de werking van de opdrachten uitgelegd worden.

2.1 Opdracht Functieblok library (Unitronics)

Op het gebied van automatisering is het voor een programmeur van groot belang dat deze een complete software kan opbouwen in een minimale tijd. Veel leverancies van PLC software's proberen in te spelen op de wensen van de klant. Binnen Beuk engineering is Unitronics één van deze leveranciers. Unitronics heeft vrij recent de Unistream PLC op de markt gebracht met de betreffende Unilogic software. In dit concept is er veel rekening gehouden met het snel en efficiënt maken van de software. Eén van deze concepten was de mogelijkheid om functieblokken te ontwikkelen. Wanneer een functieblok opgebouwd is kunnen deze hergebruikt worden in nieuwe projecten door de functies te ex- importeren. Op die manier hoeft men veel gebruikte stukken software niet opnieuw op te implementeren.

Deze opdracht is toegewezen met het idee om alle hardware componenten, die gebruikt zijn in eerder installaties, te verwerken in een library binnen Unilogic. Bij deze installatie betrof het menginstallatie voor bouwbedrijven met het doel materialen te doseren en te mengen. Wanneer een nieuwe installatie wordt aangeschaft en opgebouwd kan het gebeuren dat er nieuwe hardware componenten worden toegevoegd en/of oude hardware componenten worden verwijderd. Het nadeel van aanpassingen maken in een proces waarbij de hardware componenten in de ladder programma verwerkt zijn, is dat deze het proces verloop kunnen verstoren. Wanneer er gebruik gemaakt wordt van functie blokken , moeten deze dusdanig opgebouwd worden, dat deze het proces niet kunnen verstoren wanneer deze geïmplementeerd worden.

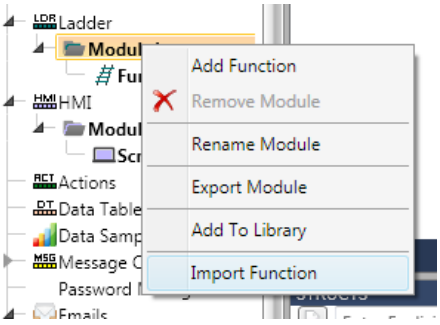
Het uiteindelijke doel van deze opdracht was, na afloop van de stage dat wanneer een nieuwe installatie gemaakt moest worden de software geprogrammeerd kon worden in de nieuwe Unistream PLC's met de bijbehorende functieblok library.

2.1.1 Onderzoek

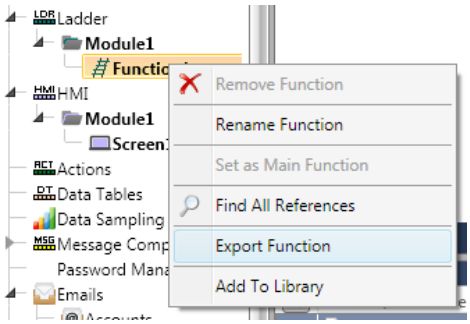
Binnen Beuk is men nog niet eerder bezig geweest met de software van Unilogic. De Unilogic software is een nieuwe uitgebrachte software special voor de Unistream PLC. Voor het verwerken van de opdracht moest als eerste stap de software uitgezocht worden qua functioneren.

2.1.1.1 Unilogic

De ladder programma's worden opgebouwd in verschillende functies, waarbij één functie de *main* functie is. Functies kunnen aangemaakt worden binnen een module (hierbij is er ook één *main* module). Wanneer een functie opgebouwd is, is er de mogelijkheid de betreffende functie te ex/importeren, wat het doel is van deze opdracht.

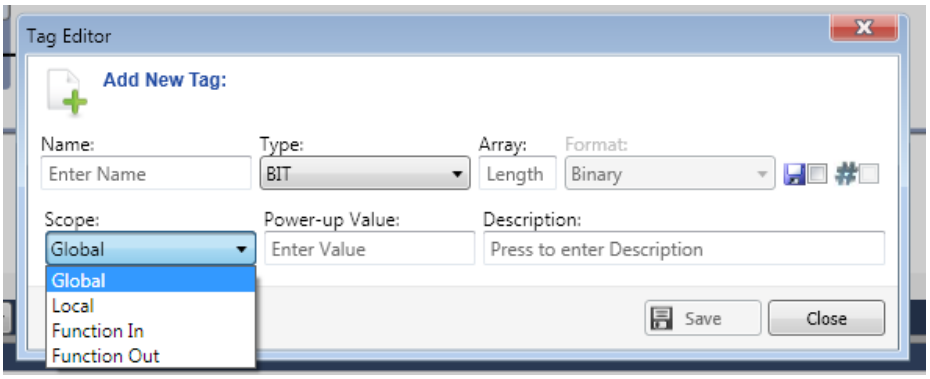


Figuur 1a, importeren van een functie in het project`



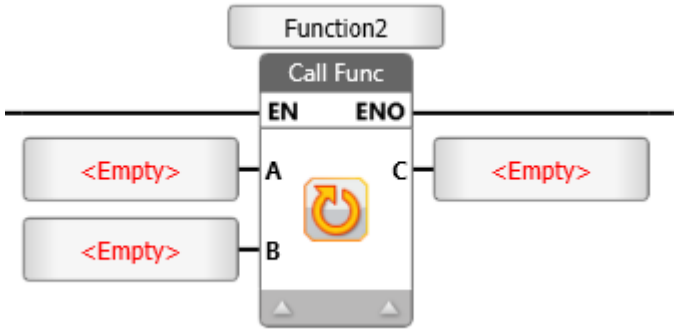
Figuur 1b, exporteren van een functie uit een project

Binnen de functie wordt het programma of dezelfde manier opgebouwd als in andere applicatie software's, met verschillende componenten. Aan elk component kan een *tag* gedefinieerd worden. Het datatype is afhankelijk van de betreffende ladder component. Daarnaast wordt ook de mogelijkheid geboden een scope te definiëren. Deze scope kan met drie verschillende waardes gedefinieerd worden, zie *figuur 2*. De scope bepaald waar het betreffende component wel en niet gebruikt kan worden.



Figuur 2, tag editor

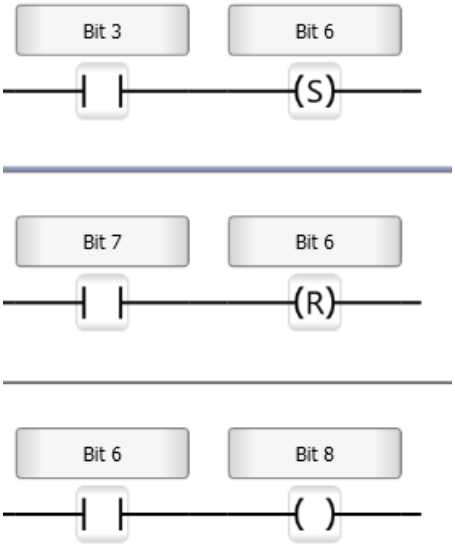
Ladder componenten met een *global* scope mogen over het gehele project aangeroepen worden en de *local*, *function in* en *function out* (de lokale scopes) alleen binnen de betreffende functie. Bij het opbouwen van een functie kunnen waardes doorgegeven worden aan de input van de functie (*function in*). Met de lokale waardes kunnen deze de waardes verwerken en eventueel aangepast worden. Met de functie outputs (*function out*) kunnen de waardes doorgegeven worden.



Figuur 3, een functie met lokale scopes. 2 function in en 1 function out

Hier treed probleem nummer één op. Binnen applicatie software's die ladder beschreven zijn, wordt er de mogelijkheid geboden om het proces te monitoren met een *online* functie. Hierbij kan men, bij elk ladder component zien wat de huidige waarde of status is. Binnen Unilogic is dit alleen mogelijk bij de globale scopes.

Door dit probleem moesten de functie blokken opgebouwd worden met globale scopes. Het nadeel van globale scope is dat deze niet bij de call van de functie gedefinieerd kunnen worden. Bij het opbouwen van de blokken moest er duidelijk aangegeven worden welke ladder componenten de inputs waren en welke de outputs. Wanneer de blokken succesvol getest waren, konden de scopes omgezet worden tot lokale scope. Hierbij zou, voor somminge ladder componenten de scope globaal blijven omdat deze bij meerder functies aangeroepen moesten worden, bij voorbeeld een storings bitjes.
Bij het testen hiervan trad probleem nummer twee op. Een lokale scope kan niet normaal aangestuurd worden door een globale. Dit probleem trad op in een opgebouwde functie toen een alarm niet geactiveerd werd terwijl er een storing in de functie optrad.
Het idee hierachter was dat een globale storings bit in elke functieblok getriggerd kon worden wanneer in de betreffende functie een storing optrad. De globale bit zou in de betreffende storings functie een alarm triggeren door een lokale bit te setten. Deze kon pas gereset worden wanneer visueel een alarm reset werd gegeven.
Om dit probleem na te trekken is de betreffende functie opgebouwd in een stuk test software. Ook hieruit bleek dat het wel mogelijk was met een lokale scope een globale aan te sturen maar een globale niet een lokale. In figuur 4 is de betreffende stuk ladder van de test software te zien.

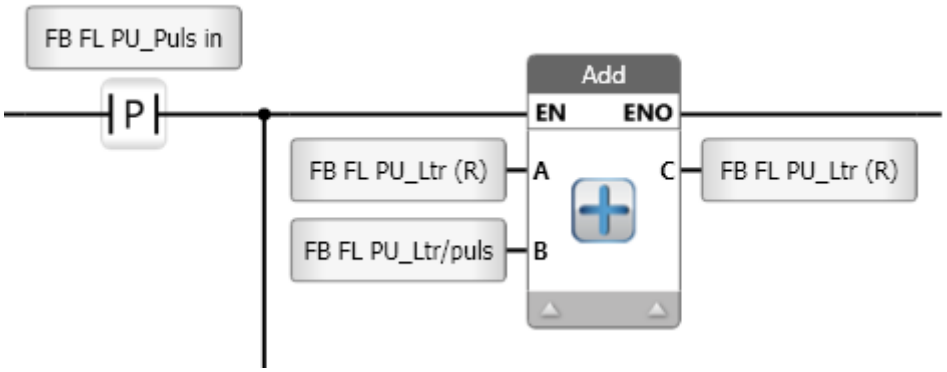


Figuur 4, ladder test software

In deze test software is *Bit 3* een globale scope en de andere bitjes zijn de lokale scopes. De uitvoer moest echter zijn dat *Bit 6* hoog gemaakt zou worden wanneer *Bit 3* hoog is en kan pas laag gemaakt worden wanneer *Bit 7* hoog is. Tijdens de test werd *Bit 6* hoog wanneer *Bit 3* hoog is en werd laag wanneer *Bit 3* laag is. Dit concludeerd dat de betreffende componenten met een lokale scopes niet de functie kan uitvoeren waar ze oorsprokelijk voor bedoeld.

Naast het feit dat de communicatie tussen globale en lokale scopes niet goed functioneert is er op het gebied van scopes nog een nadeel. Sommige ladder componenten kunnen niet aangestuurd worden door lokale scopes. Dit was eveneens opgevallen bij het omzetten van de scopes in de functie blokken.
In een aantal functie blokken wordt bij het uitrekenen van waardes gebruik gemaakt van *positive transition*. De *positive transition* wordt gebruikt omdat bij sommige tags een waarde toegevoegd moest worden bij de huidige waarde van die tag. Daarna moet de nieuwe waarde terug gegeven worden aan dezelfde tag. Dit kan alleen succesvol worden uitgevoerd met een *positive transition*. Hierbij wordt éénmalig de berekening uitgevoerd wanneer in een scan de betreffende *positive*

transition coil hoog is. In figuur 5 is een stukje software van de *plus flowmeter* te zien waarbij dit principe geïmplementeerd is.

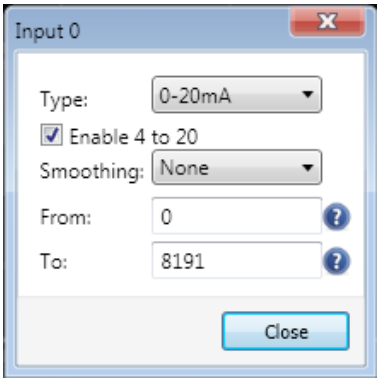


*Figuur 5, uitrekenen van het aantal liters aan de hand van liter/puls en een **ptc***

In figuur 5 was de *positive transition contact* ***FB FL PU_Puls in*** een globale scope. Toen deze omgezet moest worden naar een lokale scope was dit niet mogelijk.

Dit hield in dat de software niet functioneerde zoals het zou moeten functioneren en dat het maken van functieblokken met lokale scopes beperkt wordt tot bepaalde componenten.

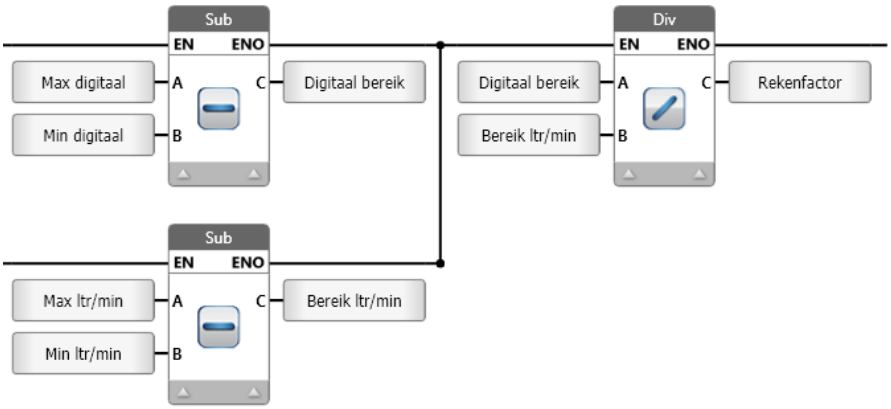
Naast het feit dat de software niet naar behoren functioneerde waren er ook voordelen. Eén van deze voordelen was dat, bij een analoge in/output de programmeur zelf kon bepalen welke digitale waarde aan een analoge waarde werd gekoppeld.



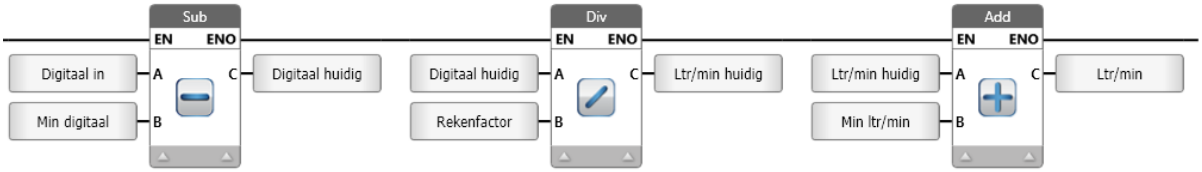
Figuur 6, het scherm voor het afstellen van een analoge input

In figuur 6 is te zien dan de programmeur zelf de type input kan bepalen en het bereik ervan. Hierbij kan gekozen worden tussen 0-20 mA of 0-10V waarbij 4-20mA ook mogelijk is door deze aan te vinken. Het digitale bereik heeft een minimum van -20000 en een maximum van 20000. Dit betekend dat de eventuele minimale of maximale waarde van een sensor (bij een minimale of maximale analoge waarde) dezelfde waarde als digitale waarde gedefinieerd kan worden. Het grote voordeel hiervan is dat de gemeten/ gegeven, analoge waarde niet omgerekend hoeft te worden. Of te wel, de in/ uitkomende waarde is de gemeten/ gegeven waarde. Het ligt voor de hand dat de betreffende programmeur voor deze optie kiest. Op deze manier zal er binnen de software continue maar één integer/ float gebruikt te hoeven worden in plaats van een complete bouwsteen waarbij de programmeur meerdere waardes moet opgeven. Dit geeft geen garantie dat, wanneer het digitale bereik gelijk wordt gemaakt aan de bereik van de sensor ook daadwerkelijk de goede waarde gegeven wordt. Daarnaast moet onthouden worden; hoe kleiner het digitale bereik wordt, hoe onnauwkeuriger de gemeten waarde zal zijn.

De applicatie software zelf geeft bij de hardware configuratie een default digitale waarde. Deze heeft een bereik van 0 tot 8191. Er bestaat een mogelijkheid dat de waardes afwijken.
Voor de opdracht werd bij het opbouwen van de functiebouwstenen in alle gevallen uitgegaan van de default of afwijkende digitale waardes. Hierdoor moest een stukje software ontwikkeld worden waarbij het digitale bereik en het bereik van de sensor variabel kon zijn maar de eind waarde altijd correct was. In figuren 7 & 8 zijn stukken ontwikkelde software voor de analoge flowmeter te zien die zorgen voor de juiste waarde, ongeacht het digitale bereik en het bereik van de sensor. Let wel, alle berekeningen worden met real waardes berekend om de uitkomst zo nauwkeurig mogelijk uit te kunnen rekenen. Hierbij zullen in de software de inputs eerst geconverteerd worden van een *integer* naar een *real* datatype.



Figuur 7, uitreken van een rekenfactor



Figuur 8, het aantal liter per minuut uitreken aan de hand van de rekenfactor

In figuur 7 wordt het digitale bereik en het bereik van de sensor uitgerekend door de minimale waarde van de maximale waarde af te trekken. Daarna wordt het bereik van de sensor gedeeld door het bereik van de digitale input waarna een rekenfactor uitgerekend wordt. Met deze rekenfactor kan de huidige, digitale input omgerekend worden naar een sensor waarde.
In figuur 8 is te zien dat de huidige digitale rekenwaarde wordt uitgerekend door de minimale digitale waarde van de inkomende, digitale waarde af te trekken. Daarna wordt de huidige digitale waarde gedeeld door de rekenfactor waarbij de rekenwaarde voor het aantal liter per minuut wordt berekend. Om het juiste aantal liters per minuut uit te rekenen moet nog het minimaal aantal liter per minuut bij de huidig aantal liters per minuut opgeteld worden.

Voorbeeld:
De default digitale waarde wordt aangehouden (min 0, max 8191) en we hebben een analoge flowmeter waarbij de minimale waarde 2 ltr/min is en de maximale waarde 40 ltr/min. Dit betekend dat, wanneer de digitale input 0 is, het aantal liter per minuut 2 is en wanneer de digitale waarde 8191 is, het aantal liter per minuut 40 is. De rekenfactor is hierbij als volgt:

$$\frac{8191-0}{40-2}=\frac{8191}{38}=215{,}55$$

Dit beteken dat de betreffende digitale input gedeeld moet worden door 215,55 om de juist uitkomst te krijgen. Nu betreft als eerste de digtale input waarde 0. Dit betekent dat de uitkomt 2 liter per minuut moet bedragen.

$$(0 - 0) = \frac{0}{215{,}55} = 0 + 2 = 2\ ltr/min$$

Hier is te zien dat het van belang is, dat de minimale waarde van de sensor bij de huidige waarde wordt opgeteld. De uitkomst is correct maar nu wordt de digitale input waarde de helft van het bereik (4095, dit is geen 4095,5 omdat de binnen komende input een integer is die converteert wordt). Dit betekend dat het aantal liters per minuut de helft van het bereik moet zijn waarbij de minimale waarde opgeteld moet worden. Deze gaan we uitrekenen.

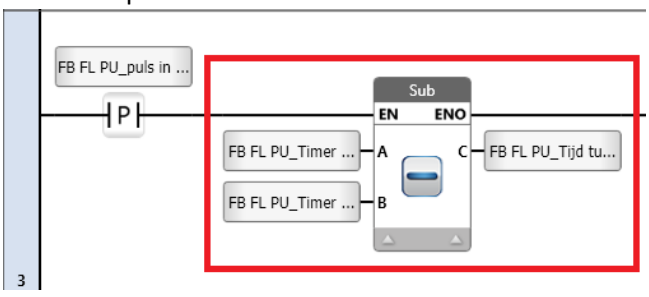
$$(4095 - 0) = \frac{4095}{215{,}55} = 19 + 2 = 21\ ltr/min$$

Dit klopt ook. Nu nog 1 maal maar dan de maximale, digitale waarde.

$$(8191 - 0) = \frac{8191}{215{,}55} = 38 + 2 = 40\ ltr/min$$

Ook hier is te zien dat dit klopt. Dit concept zal in alle functieblokken terug komen waarbij een sensor gekoppeld is aan een analoge in en uitgang maar ook bij een high speed counter. Hierbij wordt in plaatst van een minimale en maximale digitale waarde een minimale en maximale frequentie opgegeven. Deze zijn in de specificaties van de betreffende component terug te vinden.

Naast een analoge aansturing en aansturing doormiddel van een high speed counter zijn er ook sensoren die pulsen uitsturen. Wanneer een sensor pulsen uitstuurt, vereist het een digitale input op de PLC. In de PLC betekent het niets meer dan dat een bitje hoog en laag wordt wanneer de sensor iets detecteert. Bij een sensor met pulsen is het de bedoeling dat de hoeveelheid, per tijdseenheid gegeven wordt aan de hand van het aantal pulsen die bij het proces worden afgegeven en de PLC binnen komen. Hierbij hoeft de programmeur alleen de gespecificeerde hoeveelheid per puls op te geven. Wanneer de PLC deze pulsen binnen krijgt zal de functieblok de gewenste waardes, visueel weergeven.
Voordat er waardes gegeven worden moet het proces gestart zijn waarbij materiaal langs de sensor gaat lopen. Hierbij zal een bit hoog gemaakt worden vanaf een voorgaand stuk PLC programma. Wanneer deze actief is zal het functieblok wachten tot er een puls binnen komt en wordt er een timer gestart. Alle berekening worden gedaan aan de hand van het aantal liters per minuut. Hierbij wordt eerst het aantal liters per minuut bij 1 puls per seconden berekend (gewoon het aantal liters per puls vermenigvuldigen met 60). Daarna zal een timer aflopen van 1000 naar 0 milliseconden. Wanneer er een puls binnen komt vanaf het starten van het proces wordt er gekeken naar de tijd tussen de pulsen.



Figuur 9, meting tijd tussen pulsen

Op deze manier zullen de waardes die uitgaan, nauwkeuriger gemeten worden. Per puls zal een *positive transition coil* getriggerd worden. De betreffende *positive transition contact* wordt hoog die eerst ervoor zorgt dat de huidige timer waarde wordt afgetrokken van de totaal waarde van deze timer. Dus wanneer het proces start gaat de timer lopen van 1000 naar 0 en wanneer er bijvoorbeeld na 200 ms een puls binnen komt, zal de huidige timer waarde 800 ms zijn. Deze wordt van de 1000 afgetrokken waarbij er de waarde 300 uitkomt. Wanneer er na 1 seconden geen puls binnengekomen is wordt er een aparte *real* tag verhoogd met de waarde 1000. Op deze manier wordt er bijgehouden of een puls pas na meerder secondes binnen komt.

Voorbeeld:
Een puls flowtransmitter geeft aan dat er per puls 3,5 liter voorbij komt. Nu meet de flowtransmitter een flow met een constante frequentie van 5 Hz (5 pulsen per seconden). De timer zal dus van 1000 ms gaan lopen en bij 800 ms gereset worden. Hierbij is dus de tijd tussen 2 pulsen 200 ms (1000 - 800). Dit betekend dat 3,5 * 5 = 17,5 liter per seconden de flowtransmitter voorbij gaat. Omgerekend moet de flow 17,5 * 60 = 1050 liter per minuut bedragen.
In de software wordt deze waarde verkregen door tijd tussen pulsen te delen door het aantal liters per minuut bij 1 puls per seconden (3,5 * 60 = 210 liter per minuut bij 1 puls per seconden). Omdat de gemeten tijd is milliseconden is, zal voor het berekenen van het aantal liter per minuut de waarde nog vermenigvuldigt moeten worden met waarde 1000.

$$\frac{210}{200} = 1{,}05 * 1000 = 1050\ liter\ per\ minuut$$

2.1.1.2 De componenten

De functie blokken zijn aan de hand van een flowchart van een bestaande menginstallatie gemaakt, **zie bijlage A**. Bij deze installatie wordt een mengsel gemaakt door de toevoer en verwerking van verschillende materialen. Dit alles wordt gedaan door de communicatie en aansturing van verschillende componenten. Allereerst moest het procesverloop in kaart gebracht worden. Hierbij kon nagegaan worden welke componenten elkaar aansturen tijdens het proces. Het procesverloop van de menginstallatie is te zien in **bijlage B**. Hierbij is vanaf het eindmengsel terug gewerkt naar de begin materialen. Bij elk stukje proces is nagegaan welke component betrokken is in dat stukje proces. Op deze manier is duidelijk te zien welk component van invloed is op een ander. Voor de werking van het betreffende componenten is ook op internet naar documentatie gezocht (zie hoofdstuk *literatuur*). In deze datasheets wordt duidelijk aangegeven wat de inputs en outputs zijn van het betreffende component en hoe deze eventueel aangestuurd moet worden. Bij verschillende componenten wordt er gebruik gemaakt van een puls, analoog signaal of een frequentie. In de datasheets is wordt bij componenten met een puls aangegeven, de hoeveelheid per puls. Bij analoge signalen of frequenties gegeven de minimale hoeveelheid bij een minimaal analoog signaal of minimale frequentie en bij maximale hoeveelheid de maximale waardes. De componenten werden in een lijst opgesteld met daaraan de betreffende in/outputs en de betreffende datatype, **zie bijlage C**. Hierbij zijn niet alle voorkomende componenten verwerkt. Hieronder is te zien welke componenten opgebouwd zijn in functie blokken en welke niet maar belangrijker nog, waarom?

Opgebouwde componenten:
- Noodstop/trekkoord
- Flowtransmitter
- Plaatweger
- Motor
- Motor frequentie geregeld

- Motor softstarter geregeld
- Temperatuurmeting
- Level transmitter

De noodstop en trekkoord hebben dezelfde functie en zijn allebei van belang omdat deze ervoor moeten zorgen dat, wanneer deze geactiveerd worden, het proces stil komt te staan. Het proces mag dan pas weer opgestart worden als de gebruiker de hardware herstelt heeft en een reset heeft afgegeven.
De flowtransmitter werd opgebouwd omdat hier een stuk reken en meetwerk gedaan moet worden om de juiste hoeveelheid liters als uitkomst te krijgen.
De plaatweger werd ook opgebouwd. Deze zou qua opbouw het zelfde idee hebben als de flowtransmitter, alleen zou hier verschillende waardes anders uitgerekend om de juiste uitvoer te krijgen. Dit was het idee aan het begin. Naar mate de stage vorderde bleek deze exact hetzelfde te functioneren als de flowtransmitter.
Een motor is een veel gebruikte component in de besturingstechniek. Deze werd opgebouwd omdat motoren pas aangestuurd mogen worden wanneer daar een signaal voor afgegeven wordt.
Daarnaast zijn er nog twee varianten op de motor waarbij een stuk rekenwerk achter zit, zoals de frequentie geregelde motor en de soft starter.
Voor de temperatuurmeting en leveltransmitter geld hetzelfde als de flowmeter en de plaatweger. Deze zal dezelfde structuur krijgen alleen zit ook hier ander rekenwerk achter.

Niet opgebouwde componenten:

- Draaisluis
- Level switch
- Speedsensor
- Drukschakelaar
- Standenmelder
- Ventilator
- Doseerpomp
- Klep elektrisch
- Bandweger
- Deurschakelaar

Sommige van deze componenten zoals de draaisluis, de doseerpomp, de ventilator en de deurschakelaar, worden niet opgebouwd omdat ze een onderdeel zijn van een opgebouwde component. Deze componenten (op de deurschakelaar na) worden aangestuurd op dezelfde manier als de motor, zowel frequentie als relais gestuurd. De deurschakelaar heeft dezelfde functionaliteit als de noodstop. Hierdoor hoeft alleen maar een stuk software voor de aansturing van een motor en noodstop gemaakt te worden. Daarnaast worden sommige componenten niet gebruikt omdat ze alleen maar een aan/uit functionaliteit hebben, zoals de level switch, de drukschakelaar, de standenmelder en de elektrische klep. De programmeur hoeft hier alleen een simpel contactje in de software te definieren. Als laatste zijn er componenten met een eigen stuk software die kant en klaar geleverd worden. Deze leveren een juiste waarde aan de PLC waarbij niets berekend hoeft te worden, zoals de bandweger en de speedsensor.

Naast deze hardware matige componenten moest er ook een functieblok ontwikkeld worden voor het verwerken van storingen. Wanneer in een component een fout optreed moet er een alarmsignaal afgegeven worden zodat de gebruiker weet dat een storing opgetreden is.

Na het opzoeken van deze informatie zijn de functie bouwstenen eerst op papier uitgewerkt. Daarna zijn de functies dusdanig opgebouwd dat ze voldeden aan alle specifieke eisen die gesteld waren aan

het betreffende component. Bijvoorbeeld, wanneer een component zowel digitaal als analoog aangestuurd kan worden, zullen beide aansturingen in een bouwsteen geïmplementeerd worden. In de eerste instantie zouden alle aansturingen in één blok verwerkt worden. Dit is niet gedaan om de volgende redenen:

- De overzicht van de programmatuur zou verloren gaan.
- Het programma zou onnodig netwerken scannen die totaal niet gebruikt worden.
- De kans op storingen zou groter zijn.

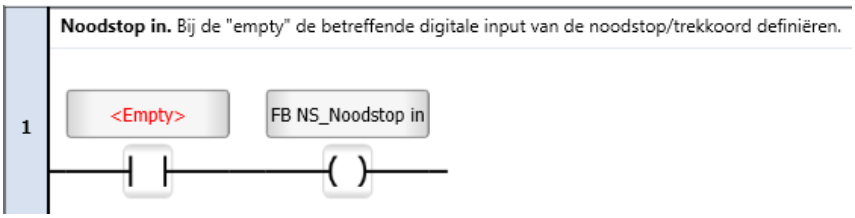
2.1.2 De opbouw

Bij het opbouwen van de blokken was het de bedoeling dat de programmeur zijn gewenste inputs en outputs definieert. De functieblok zou qua programmering dusdanig zijn opgebouwd dat deze de juiste waarde uitstuurt. Vanwegen het feit dat de functieblokken niet met lokale scopes opgebouwd kon worden, is er binnen de functies een bepaalde structuur gehanteerd. Binnen Unilogic is het mogelijke stukken netwerk op te splitsen in verschillende regio's, in Unilogic Regions genoemd. Dit is gedaan om ervoor te zorgen dat de betreffende programmeur weet waar de inputs gedefinieerd moet worden, waar het proces van de component aangestuurd wordt, waar de betreffende waardes omgerekend worden en waar de outputs gedefinieerd moeten worden. Voor sommige functies is er ook een beveiling geïmplementeerd. Deze komen terug in functies waarbij een waarde uitgestuurd wordt die het proces aantal bijhoud.

Figuur 10, opdeling regio's binnen de functieblokken

2.1.2.1 Inputs

De regio met functie inputs is heel simpel opgebouwd. In vergelijking met functies waarbij lokale scopes worden gedefinieerd bij de 'call' van de functie (zie figuur 3) worden de global scopes binnen de functie gedefinieerd. Hierbij moet de programmeur voor de functie inputs waar "empty" staat de betreffende I/O tags, functie waardes of globale tags te definiëren. In de functie wordt in de comments aangegeven waar de betreffende inputs voor bedoeld zijn. In figuur 8 is een input te zien van een *noodstop*. Hierbij moet de programmeur bij empty de betreffende digitale I/O definieren die getriggerd wordt wanneer een noodstop is ingedrukt.



Figuur 11, input noodstop

2.1.2.2 Aansturing

Bij de regio aansturing wordt binnen elke functie het betreffende proces verwerkt en eventuele metingen verricht. Deze regio is als volgt opgebouwd:

- Uitrekenen van een rekenfactor (componenten analoog of frequentie aangestuurd).
- Controle of de regeling actief is (alle componenten). Daarna de betreffende, inkomende waarde uitrekenen aan de hand van de input en rekenfactor en een timer starten voor de meting (componenten puls, analoog of frequentie aangestuurd).
- Aan de hand van de timer de huidige hoeveelheid uitrekenen en de timer resetten (componenten puls, analoog of frequentie aangestuurd).
- Controleren op storing (alle componenten).
- Triggeren van een alarm bij storing.
- Reset van cumulatieve waardes.
- Als de regeling inactief is, zorgen dat er geen waardes worden weergegeven op het scherm (dit voor componenten waarbij de waardes visueel worden weergegeven).

2.1.2.3 Omrekening

In de regio omrekening worden alle gemeten waardes omgerekend naar de gewenste waardes. Hierbij worden de betreffende waardes uitgerekend en terug geconverteerd naar integers. Binnen alle functieblokken worden de waardes doormiddel van real datatypes uitgerekend. Op deze manier zijn de metingen nauwkeuriger.

2.1.2.4 Outputs

De outputs zijn hetzelfde opgebouwd als de inputs waarbij hier de betreffende outputs aangestuurd moeten worden door de outputs van de functieblok

2.1.2.5 Beveiliging

Bij sommige functies kan een waarde niet gereset worden. Hierbij blijft de betreffende waarde oplopen tot een interger ‘over de kop’ kan gaan. Binnen deze beveiliging wordt de betreffende waarde gereset voor deze over de kop kan gaan. Elke keer dat een integer gereset wordt zal een counter met waarde één verhoogd worden.

2.1.2.6 Omschrijving opbouw componenten

Voor elke functieblok zal een functie omschrijving gegeven worden. Zoals eerder vermeld zijn de in en outputs van de componenten terug te vinden in **bijlage C**. In **bijlage D** is de opgebouwde software van de functieblokken terug te vinden.

Signalering storingen
In deze functie zal een alarm signaal afgegeven worden als in één van de functie blokken een storing optreed. Hierbij kan de programmeur zelf bepaalden hoelang een signaal actief moet zijn en hoelang deze inactief moet zijn. Bij een reset zal de signalering gereset worden. Deze wordt weer geactiveerd wanneer de betreffende component nog in storing staat en de tijd voor het her activeren van de signalering verstreken is.

Noodstop/trekkoord
In dit functie blok wordt gekeken of een noodstop ingedrukt wordt. Wanneer deze actief is zal een storing bit hoog worden en zal een signaal afgegeven worden aan het blok *signalering storingen*.
Inputs:

Flowtransmitter puls
Hier wordt het aantal liters berekend aan de hand van binnenkomende pulsen. In de specificaties van de flowtransmitter wordt aangegeven hoeveel liter per puls voorbij gaat en de programmeur zal deze waarde definiëren bij de inputs. Aan de hand van de pulsen zal het aantal liters per minuut gemeten worden. Het aantal liters per minuut zal omgerekend worden naar liters per uur en kubieke meters per uur. Daarnaast wordt bij elke puls het aantal liters en kubieke meters opgeteld. Wanneer de regeling actief is en er geen pulsen gemeten worden zal een storingbit hoog worden en zal een signaal afgegeven worden naar de functie blok *signalering storingen*.

Flowtransmitter analoog
Deze heeft dezelfde functie als de puls flowtransmitter alleen hier worden de waardes gemeten en berekend aan de hand van een binnenkomende analoog signaal. In de specificaties wordt het aantal liter per minuut gegeven bij een minimale en maximale analoge waarde. De programmeur moet hierbij een maximale en minimale digitale waarde opgeven die hij gedefinieerd heeft in de hardware configuratie. Daarnaast moet hij het maximale en minimaal aantal liters per minuut opgeven. Aan de hand van deze gegevens kan een rekenfactor berekend worden. Met deze rekenfactor wordt de digitale waarde van het binnenkomende signaal omgerekend naar huidig aantal liters per minuut. Hier treed een storing op wanneer de regeling actief is en de digitale waarde van het binnenkomende signaal kleiner is dan de minimale digitale waarde.

Flowtransmitter high speed counter
Hier worden de waardes berekend aan de hand van een binnenkomende frequentie. Hierbij moeten de maximale en minimale frequentie en liters per minuut opgegeven worden. Deze waardes zijn terug te vinden in de specificaties van de flowtransmitter. Hier treed een storing op wanneer de regeling actief is en de binnenkomende frequentie lager is dan de minimale frequentie.

Plaatweger puls, analoog en high speed counter
De puls plaatweger, analoog plaatweger en high speed counter plaat weger worden op dezelfde manier aangestuurd als die van de flowtransmitter. Het verschil hierbij is dat het aantal kilogrammen

per minuut, kilogrammen per uur, ton per uur, totaal aantal kilogrammen en totaal aantal ton wordt berekend.

Motor
Hier wordt een motor aangestuurd wanneer daar een signaal voor afgegeven wordt. Deze zal in storing gaan wanneer de thermische beveiliging wordt aangeroepen. De hardware matige thermische beveiliging zit aangesloten op een digitale input. Bij de outputs moet de digitale output gedefinieerd worden waar het motor relais op aangesloten is.

Motor frequentie geregeld
In vergelijk met een gewone motor wordt een frequentie geregelde motor aangestuurd met een analoog signaal. Hierbij kan visueel de snelheidspercentage bepaald worden waarbij een analoog uitgangssignaal daarop wordt aangepast. Hier is ook de mogelijkheid om de motor in tegenovergestelde richting te laten draaien. Net als de gewone motor wordt een storings bit hoog wanneer de thermische beveiliging van de motor ingeschakeld wordt.

Motor soft starter geregeld
Deze wordt op dezelfde manier als de relais aangestuurde motor. Het verschil is, is dat er eerst een vrijgave gegeven moet worden voordat de motor kan gaan draaien.

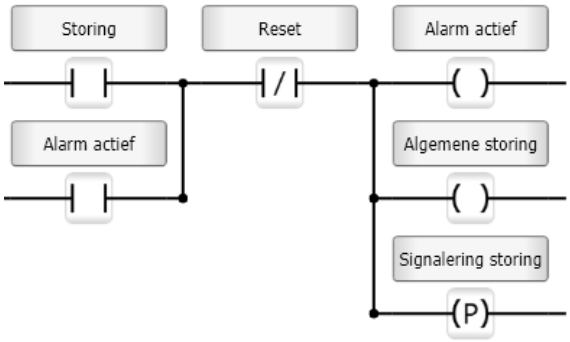
Temperatuurmeting
Hier wordt door een binnenkomende, analoge waarde de temperatuur gemeten. Aan de hand van een ingestelde temperatuur kan de temperatuur geregeld worden door een verwarming of koeling te activeren. Hierbij moet bij de uitgang de betreffende verwarming of koeling gedefinieerd worden. Ook is er de mogelijkheid om een correctie waarde in te stellen.

Level transmitter
Hier wordt met een analoog signaal de inhoud van een opslag gemeten. Deze kan in vier stadiums staan; vol, bestel, waarschuwing en leeg. Bij elke stadium kan een percentage over de totale inhoud gedefinieerd worden. Wanneer de inhoud deze percentage bereikt heeft zal één van deze stadiums actief zijn, bijvoorbeeld vol is 95%. Wanneer de inhoud 95% van de opslag is, wordt de toevoer stop gezet. Bij besteld moet de toevoer naar de opslag gestart worden. Bij een waarschuwing zal er visueel aangegeven worden dat de opslag bijna leeg is. Bij leeg moet de toevoer tot het proces stop gezet worden.

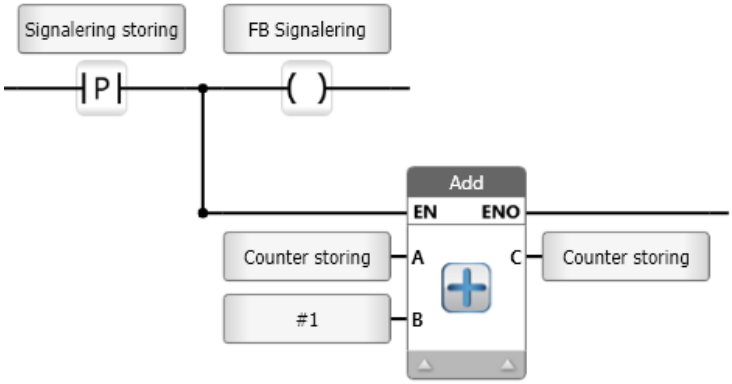
2.1.3 De werking

Nu zal de werking van de software worden uitgelegd. Hierbij zal bij elk stukje gerefereerd worden naar de betreffende bijlage, regio en rung nummer.

Signalering storingen
In elke functieblok is een systeem geïmplementeerd waarbij de signalering in de functie *signalering storingen* getriggerd wordt.



Figuur 12, systeem voor het triggeren van een alarm



Figuur 13, vervolg alarm met storings counter

In figuur 12 zal, wanneer een storing optreed, een alarm bit binnen de functie getriggerd worden waarna de alarm bit de storing bit overbrugt. Hierdoor zal het alarm actief blijven tot een reset van het component gegeven wordt. Daarnaast wordt een algemene storing bit getriggerd. De algemene storing bit zorgt ervoor dat de betreffende alarm signaal in de functieblok *signalering storing* weer geactiveerd moet worden (*FB SS_Her activatie signalering*) wanneer het alarm van het component niet gereset is maar de signalering wel gereset is. Met de *positive transition* signalering storing wordt in figuur 13 een storingscounter opgeteld. Deze houd het aantal storingen die in een component optreed bij. Deze *positive transition* zal ook een bit voor de functieblok *Signalering storing* triggeren.

Binnen het functieblok wordt de software alsvolgt doorlopen:

Bijlage D, functieblok signalering storing, regio aansturing
Rung 1:
De signalerings bit ***FB SS_Signalering*** wordt geset wanneer één van de componenten het *storing* bit ***FB SS_Signalering*** triggerd. Wanneer een visuele reset gegeven is maar de component nog in storing staat zal via de *positive transition* ***FB SS_Her activatie*** de betreffende signalering weer geactiveerd worden totdat de betreffende component gereset is.

Rung 2:
Het signalerings bit zal gereset worden wanneer visueel een reset gegeven worden en daarbij de bit ***FB SS_Reset signalering*** getriggerd wordt.

Rung 3:
Het uitgaande signalerings bit ***FB SS_Signalering uit*** zal de digitale output aansturen wanneer het signalerings bit hoog is en de timer.out 1 hoog is.

Rung 4:
De eerste timer gaat lopen wanneer de timer2.out niet hoog is. Hierbij zal de uitgaande signalerings bit in rung 3 niet actief zijn. Wanneer de input gedefinieerde tijd verstreken is zal de signalering actief zijn.

Rung 5:
De tweede timer gaat lopen wanneer de eerste timer klaar is. Deze zal, wanneer bij de input gedefinieerde tijd verstreken, de eerste timer reset waardoor het signaal inactief zal.

Rung 6:
Wanneer het algemene storings bit uit de betreffende component nog hoog is, en de signalering is gerest gaat, een derde timer lopen.

Rung 7:
Wanneer de gedefinieerde tijd van de derde timer verstreken is zal de signalerings bit in rung 2 weer geset.

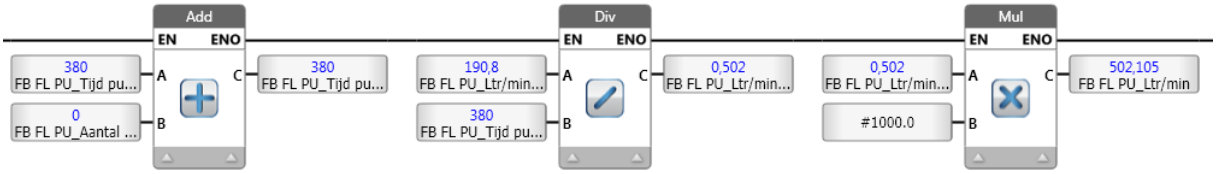
Noodstop/trekkoord
Bijlage D, functieblok Noodstop/trekkoord
In de rungen 1 & 2 wordt een alarm getriggerd zoals weergegeven is in de figuren 12 & 13. Het storings bit wordt getriggerd wanneer via de digitale input, de noodstop of trekkoord contact maakt.

Flowtransmitter puls
Bijlage D, functieblok Flowtransmitter puls, regio aansturing
Rung 1:
Wanneer de flowmeter actief is, gaat de timer voor de meeting lopen. Deze staat ingesteld op 1000 milliseconden (1 seconde)

Rung 2:
Hier wordt de *positive transition* ***FB FL PU_Puls in*** bij iedere binnenkomende puls in, getriggerd.

Rung 3:
Hier wordt het aantal liters per minuut uitgerekend door de tijd tussen 2 pulsen te meten. In rung 1 is de timer actief.
Wanneer er na elke seconden geen puls gedetecteerd is wordt de waarde van ***FB FL PU_Aantal milli sec*** met 1000 milliseconden opgehoogd.
Wanneer de flowtransmitter een puls heeft afgegeven, wordt als eerste de huidige tijd van de timer van de ingestelde tijd (1000 milliseconden) afgetrokken en wordt in de tag ***FB FL PU_Tijd tussen puls*** geplaatst. Let op! Dit wordt gedaan omdat timers van hun ingestelde waarde terug lopen naar 0.
Daarna wordt het aantal milliseconden uit de tag ***FB FL PU_Aantal milli sec*** bij de waarde van de tijd tussen pulsen opgeteld. Hierbij hebben we de totale gemeten tijd hebben tussen twee pulsen. In rung 1 uit regio omrekening wordt het aantal liters per minuut uitgerekend bij 1 puls per seconden. Na de gemeten tijd te hebben verkregen wordt het aantal liter per minuut bij 1 puls per seconden gedeeld door de gemeten tijd. Hierdoor wordt het huidig aantal liters per minuut verkregen. Deze waarde zal met een factor 1000 nog gecorrigeerd moeten worden omdat de tijd tussen pulsen in milliseconden gemeten is en de waarde van aantal lites per minuut bij 1 puls per seconden in seconden berekend is. Daarna wordt het aantal milli seconden weer op nul gezet en de timer gereset.

Voorbeeld:
Het aantal liter per puls is 3,18. Het aantal liter per minuut bij 1 puls per seconden is 3,18 * 60 = 190,8. Na 380 milliseconden wordt een puls afgegeven (de huidige tijd van de timer is 620 en de waarde van ***FB FL PU_Aantal milli sec*** is 0). Het huidig aantal liters per minuut is als volgt:



Figuur 14, uitvoer software

Rung 4:
Hier wordt een alarm getriggerd wanneer de doseerpomp actief is en de ingestelde alarm tijd verstreken is.

Rung 5:
Hier wordt het aantal storingen die in de flowtransmitter optreed bijgehouden.

Rung 6:
Wanneer de flowtransmitter niet actief is, wordt de waarde 0 naar de tags ***FB FL PU_Aantal milli sec*** en ***FB FL PU_Ltr/min*** geschreven. Hierdoor wordt er visueel de waarde 0 gegeven voor het aantal liter per minuut, liter per uur en kubieke meters per uur.

Rung 7:
Wanneer visueel een reset wordt gegeven voor de cumulatieve waardes, dan worden de cumulatieve tellers weer waarde 0.

Bijlage D, functieblok Flowtransmitter puls, regio omrekening
Rung 1:
Het aantal liters per minuut bij 1 puls per seconden wordt uitgerekend door de gedefinieerde aantal liters per puls te vermenigvuldigen met waarde 60.

Rung 2:
Het aantal liters per uur wordt berekend aan de hand van de gemeten aantal liter per minuut uit rung 3 in deregio aansturing.

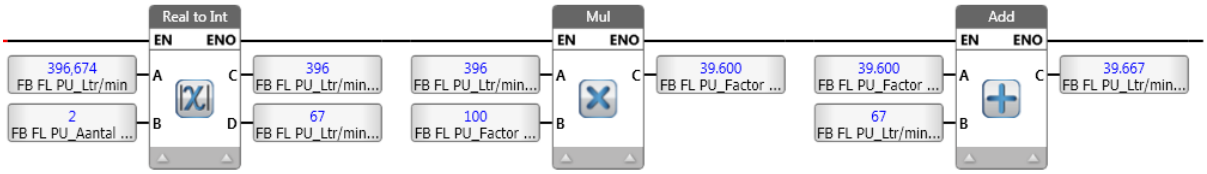
Rung 3:
Het aantal kubieke meter per uur wordt berekend aan de hand van het aantal liters per uur.

Rung 4:
Bij elke *positive transition* van rung 2 regio aansturing, wordt de waarde van aantal liter per puls opgeteld bij cumulatief en totaal aantal liters en kubieke meters. Het totaal aantal liters en kubieke meters zal worden gedeeld door de, bij de input gedefinieerde factor. Deze factor wordt gebruikt om ervoor te zorgen dat er visueel geen grote getallen worden weergegeven (denk hierbij aan de toerenteller van een auto. Het aantal toeren is gelijk aan het getal maal 1000).

Rung 5:
Hier wordt de factor voor het aantal decimalen berekend. Visueel kan het aantal decimalen van een tag mee gegeven worden. Hierbij wordt bij die tag waarde een komma geplaatst afhankelijk van het gewenste aantal decimalen.
Bijvoorbeeld tag waarde is 230. Bij een decimale waarde 2, wordt het visueel weergegven als 2,30. 3 decimalen wordt weergegeven als 0,230 enzovoort. De programmeur geeft bij de inputs aan hoeveel decimalen visueel weergegeven wordt. Als deze 2 decimalen achter de komma gebruikt dan wordt de rekenfactor 10^2 = 100.

Rung 6:
Hier wordt de *real* waarde van het aantal liter per minuut geconverteerd naar twee *integer* waardes, een geheel en een decimaal getal. Het aantal decimalen wordt bepaald aan de hand van de input gedefinieerde aantal decimalen waardes. Het geheel getal wordt vermenigvuldig met de decimale factor zoals in rung 5 beschreven is. Daarna wordt het decimale getal opgeteld en geplaats in een uitgaande display waarde voor het aantal liters per minuut.

Voorbeeld:
De real waarde is 396,674 liter per minuut en het aantal decimalen is 2. Hierbij is het gehele getal 396 en het decimale getal is 67. De waarde van de factor is 10^2 = 100. De gehele getal wordt nu 396*100 = 39600. Hier wordt het decimale getal bij opgeteld; 39600 + 67 = 39667. De *integer* waarde is nu 39667 waarbij deze visueel wordt weergegeven als: 396,67.



Figuur 15, uitvoer software

Rung 7:
Hier worden dezelfde stappen gezet als rung 6 maar nu voor de display waarde van het aantal liter per uur.

Rung 8:
Hier wordt de display waarde voor het aantal kubieke meters per uur berekend.

Rung 9:
Hier wordt de display waarde voor het cumulatief aantal liters berekend.

Rung 10:
Hier wordt de display waarde voor het totaal aantal liters berekend.

Rung 11:
Hier wordt de display waarde voor het cumulatief aantal kubieke meters berekend. Hierbij wordt nog aan het einde de waarde door 1000 gedeeld omdat de *real* waarde, het aantal liters zijn en kubieke meter, is het aantal liters gedeeld door 1000.

Rung 12:
Hier wordt de display waarde voor het totaal aantal kubieke meters berekend. Ook hier wordt aan het einde de waarde door 1000 gedeeld.

Bijlage D, functieblok Flowtransmitter puls, regio beveiliging
Rung 1:
Hier wordt gekeken of de waarde van het totaal aantal liters groter is dan de limiet van de betreffende integer. Hierbij zal een *positive transition* getriggerd worden voor rung 2.

Rung 2:
Wanneer de *positive transition* hoog is krijgt de *integer* van de totaal aantal liters de waarde 0 en wordt een counter met waarde 1 verhoogd. Op deze manier wordt er voorkomen dat de totaal aantal liters, die tijdens het proces verbruikt is, verloren gaat doordat een *integer* 'over de kop' gaat.

In de rungen 3 & 4 wordt het zelfde principe uitgevoerd als beschreven is in de rungen 1 & 2 maar dan voor de totaal aantal kubieke meters.

Flowtransmitter analoog
Bijlage D, functieblok Flowtransmitter analoog, regio aansturing
Rung 1:
Hier wordt de betreffende rekenfactor berekend die eerder op pagina 8, figuur 7 uitgelegd is.

Rung 2:
Aangezien de berekeningen met *real* waardes wordt berekend wordt zal hier de binnenkomende, digitale waarde van een *integer* geconverteerd worden naar een *real* waarde.

Rung 3:
Hier wordt de binnenkomende waarde omgerekend naar het aantal liters per minuut. Zie pagina 8, figuur 8.

Rung 4:
Bij de inputs heeft de programmeur een tijd in milliseconden gedefinieerd waarbij de gemeten waardes bereken moeten worden. Deze waarde wordt ook geconverteerd naar een *real* waarde voor verdere berekeningen.

Rung 5:
Wanneer de metingstijd verstreken is wordt de ingestelde metingstijd vermenigvuldig met het aantal liter per milliseconden. Het aantal liters per milliseconden wordt gedeeld door 1000 waarbij het huidige aantal liters verkregen wordt. De timer van de meting wordt gereset en er wordt een *positive transition* getriggerd om het aantal liters op te tellen in rung 4 van de region omrekening.

Rung 6:
Wanneer de pomp actief is wordt hier gekeken of de inkomende digitale waarde kleiner is dan de minimale waarde. Wanneer dit het geval is meet de sensor geen flow. Hierbij zal een timer gestart worden.

Rung 7:
Hier wordt een alarm getriggerd wanneer de tijd van de timer in rung 6 verstreken is.

Rung 8:
Counter storingen

Rung 9:
Visueel de waarde 0 uitsturen wanneer de flowtransmitter inactief is.

Rung 10:
Reset van de cumulatieve waardes.

Bijlage D, functieblok Flowtransmitter analoog, regio omrekening
Hier wordt net als de puls flowtransmitter de waarde uit/omgerekend. Het verschil met de puls flowtransmitter is, dat vanuit het aantal liters per minuut, het aantal liters per seconden wordt berekend. Deze wordt gebruikt in rung 5 in de regio aansturing om het aantal liters te berekenen. Daarnaast is voor het totaal aantal liters en kubieke meters een vergelijking toegevoegd om de afronding op de display tegen te gaan.

Flowtransmitter analoog, regio beveiliging
Deze functioneert hetzelfde als beschreven is bij de puls flowtransmitter.

Flowtransmitter high speed counter
Dit functieblok is precies hetzelfde opgebouwd als de analoge flowtransmitter. Het verschil hierbij is dat aan de hand van een maximale en minimale frequentie het aantal liters berekend wordt.

Plaatweger
De gehele plaatweger is qua functioneren opgebouwd als de flowtransmitter. Dit bleek voor zowel de puls, analoog als de high speed counter. Hier wordt bij de puls, het aantal kilo per puls ingevoerd, bij de analoge plaatweger wordt het minimaal en maximaal aantal kilogrammen per minuut opgegeven en bij de high speed counter, de minimale en maximale frequentie.

Motor
Bijlage D, functieblok motor regio aansturing
Rung 1:
Hier wordt de digitale output aangestuurd wanneer een start signaal gegeven en er geen thermische beveiliging opgetreden is. Op de digitale output zit het relais van de motor aangesloten.

Rung 2 & 3
Triggeren van een alarm en counter storing.

Motor frequentie aangestuurd
Bijlage D, functieblok motor frequentie regio aansturing
Rung 1:
Hier wordt via de digitale output de frequentie regelaar vrijgegeven wanneer het vrijgave bitje ***FB MF_Vrijgave signaal*** hoog is en er geen thermische beveiliging opgetreden is.

Rung 2:
Hier wordt de motor in een bepaalde richting opgestart. Wanneer het start signaal gegeven is wordt er gekeken of de motor in tegenovergestelde richting moet gaan draaien door het bitje ***FB MF_Achteruit signaal***. Wanneer deze niet actief is en de thermische beveiliging is niet ingetreden, wordt de motor via de digitale uitput in de gewone draairichting opgestart. Wanneer het achteruit signaal wel actief is zal de motor in tegenovergestelde richting draaien.

Rung 3:
Hier wordt de rekenfactor berekend aan de hand van de minimale en maximale digitale waarde en snelheidspercentage.

Rung 4:
Het visueel ingestelde snelheidspercentage wordt geconverteerd naar een *real* waarde.

Rung 5:
Hier wordt het analoge uitgangssignaal uitgerekend aan de hand van het ingevoerde percentage en de rekenfactor.

Rung 6:
De digitale waarde voor de analoge ouput wordt met een *integer* aangestuurd. De digitale waarde is in *real* uitgerekend. Hier wordt de *real* waarde geconverteerd naar een *integer*.

Rung 7:
Triggeren van een alarm wanneer het thermische relais ingeschakeld is.

Rung 8:
Counter storing.

Motor softstarter
Bijlage D, functieblok Motor softstarter, regio aansturing
Rung 1:
Hier wordt met het vrijgave signaal via een digitale output, de soft starter vrij gegeven.

Rung 2:
Hier wordt de motor gestart. Hierbij wordt bepaald of de motor in de normale of tegenover gestelde draairichting moet draaien.

Rung 3 & 4:
Triggeren van een alarm en counter storing

Temperatuur meting
Bijlage D, functieblok Temperatuur meting, regio aansturing
Rung 1:
Uitrekenen van de rekenfactor aan de hand van de digitale bereik en bereik sensor.

Rung 2:
Hier worden de digitale waarde van de analoge input en de gewenste temperatuur geconverterteerd naar *real* waardes.

Rung 3:
Hier worden de ingevoerde correctie temperatuur en de factor voor decimalen allebei geconverteerd. Daarna wordt de factor voor decimalen gedeeld door de correctie temperatuur. Hierdoor kan de correctie met de juiste waarde bij de gemeten temperatuur worden opgeteld.

Rung 4:
De temperatuur wordt berekend door de betreffende digitale waarde te vermenigvuldigen met de rekenfactor van rung 1. Hier wordt bij de berekende temperatuur, de correctie waarde opgeteld.

Rung 5:

Hier wordt de gewenste temperatuur gedeeld door de decimale factor. Dit wordt gedaan om de waardes in de rungen 6 & 7 op de goede manier uit te rekenen.

Rung 6:
Hier word een tiende waarde bij de gewenste temperatuur opgeteld. Deze factor is de maximale afwijking van de temperatuur. Wanneer de gemeten temperatuur hoger is dan de maximale afwijking dan wordt de koeling geactiveerd.

Rung 7:
Hier wordt een tiende van de gewenste temperatuur afgetrokken. Deze factor is de minimale afwijking. Wanneer de gemeten temperatuur lager is dan de minimale afwijking, dan wordt de verwarming geactiveerd.

Rung 8:
Hier wordt een timer gestart wanneer de digitale waarde van het analoge signaal kleiner is dan de minimale digitale waarde.

Rung 9 & 10:
Triggeren van een alarm wanneer de gedefinieerde tijd van de timer uit rung 7 verstreken is. Hier wordt de waarde van de storings counter met waarde 1 verhoogd bij elke storing.

Bijlage D, functieblok temperatuur meting, regio omrekening
Rung 1 & 2
Hier wordt de factor voor het aantal decimale uitrekend waarna de gemeten temperatuur wordt omgerekend naar een display waarde.

Leveltransmitter
Bijlage D, functieblok Leveltransmitter, regio aansturing
Rung 1:
De rekenfactor wordt uitgerekend door het bereik van de analoge input en het bereik van de inhoud.

Rung 2:
De digitale waarde van het inkomende analoge signaal converteren naar een *real* waarde.

Rung 3:
De totale hoeveelheid uitrekenen aan de hand van de rekenfactor uit rung 1.

Rung 4:
Een bit voor de timer in rung 5 triggeren, wanneer de inhoud groter is dan de opgegeven percentage over de totale opslag.
Bijvoorbeeld:
Een tank van 100 liter. Wanneer de opgegeven percentage 95% is wordt een timer gestart wanneer de inhoud groter is dan 95 liter.

Rung 5:
Timer starten wanneer de bit uit rung 4 hoog is.

Rung 6:
Wanneer de timer klaar is, de toevoer naar de opslag stop zetten.

Rung 7:
Een signalering afgeven dat de toevoer moet starten wanneer de inhoud kleiner is dan het opgegeven percentage.

Rung 8:
Visueel een waarschuwing triggeren wanneer de inhoud kleiner is dan het opgegeven percentage.

Rung 9:
De toevoer naar het proces stopzetten wanneer de inhoud kleiner is dan het opgegeven percentage.

Bijlage D, functieblok Leveltransmitter, regio omrekening
Rung 1:
Het opgegeven percentages voor de stadiums; vol, bestel waarschuwing en leeg omreken naar een waarde, aan de hand van de opgegeven inhoud.

Rung 2:
De factor voor het omrekenen van de decimalen bepalen.

Rung 3:
De huidige inhoud van een *real* converteren naar een *integer* voor de display.

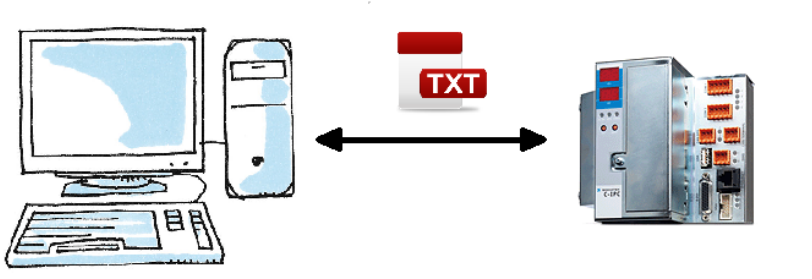
2.2 Opdracht Dataoverdracht (Sigmatek)

Beuk engineerging heeft voor bestaande machines (De Kitty's) software in Lasal class 2 laten ontwikkelingen door een software specialist.
Deze software functioneert als de server tussen de verschillende Kitty machines. De software kan een tekstbestand met proces gegevens van een PC afhalen. De inhoud van een tekst bestand wordt uitgelezen en als data opgeslagen binnen de PLC. De Kitty's kunnen de data bij de server opvragen en binnen halen. Wanneer ze over de data beschikken kunnen ze de data aanpassen. De aangepaste data kan weer terug gestuurd wordt naar de server. Na afloop zal de server de data weer terug omzetten tot een tekstbestand en deze terug plaatsen op de PC van de klant. De klant kan dan de aangepaste tekstbestand achter zijn monitor uitlezen.

Voor deze opdracht was het de bedoeling het concept omtrent dataoverdracht tussen PC en PLC uit te zoeken.
Hierbij was het eerst de bedoeling dit concept te uit te werken met een Siemens S7 300 PLC. Dit was helaas niet mogelijk omdat voor de PLC een module aangeschaft moest worden die veel te duur was. Om deze reden is voor de opdracht overgestapt naar een Sigmatek PLC omdat bij deze een communicatie module standaard is ingebouwd. Aan de hand van de bestaande server software moets het basis principe voor dataoverdracht achterhaald worden. Daarna moest de werking van de software verwerkt worden in een document.
Het doel hiervan was dat de programmeurs van Beuk engineering straks zelfstandig weten hoe dataoverdracht tussen PC en PLC functioneert en ze zelf een systeem kunnen opbouwen.

2.2.1 Onderzoek

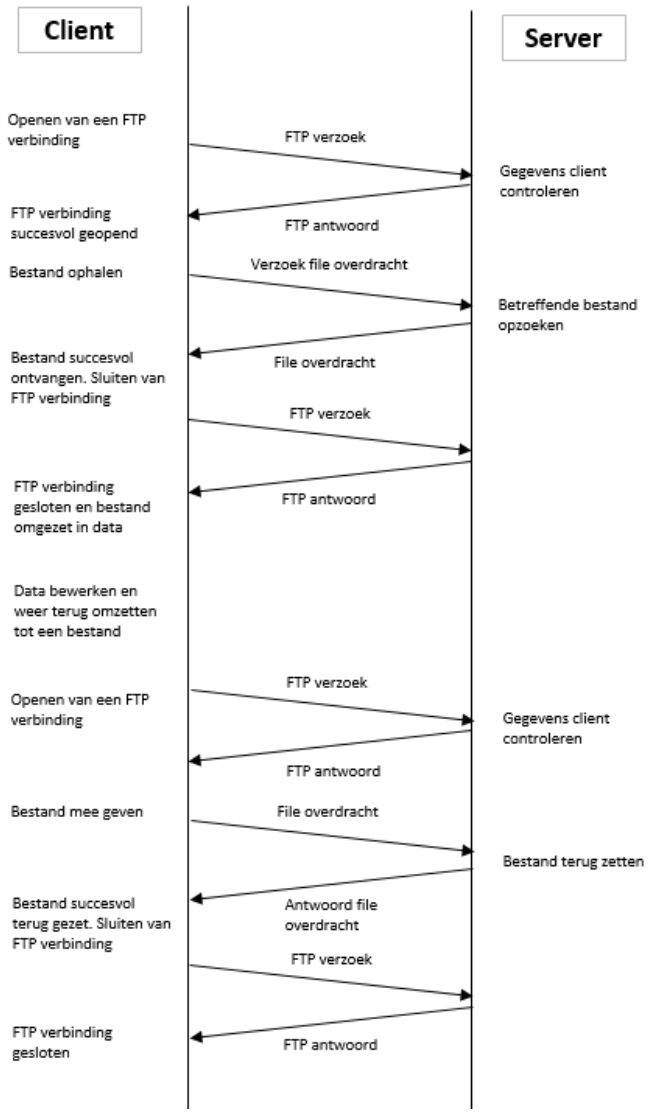
Het idee achter bestand overdracht is vrij simpel. Er is een client (bijvoorbeeld een Sigmatek c-ipc PLC) die een tekst bestand wil ophalen en er is een server (bijvoorbeeld een PC) die over het betreffende bestand beschikt. De client wil de gegevens van een tekstbestand uitlezen en de uitgelezen data kunnen veranderen. De aangepaste data moet terug geplaatst worden in het tekstbestand. Daarna zal de PLC het bestand weer terug op de harde schijf van de PC wordt gezet. De gebruiker kan hierbij eventueel het bestand weer aanpassen.



Figuur 16, globaal idee voor dataoverdracht.

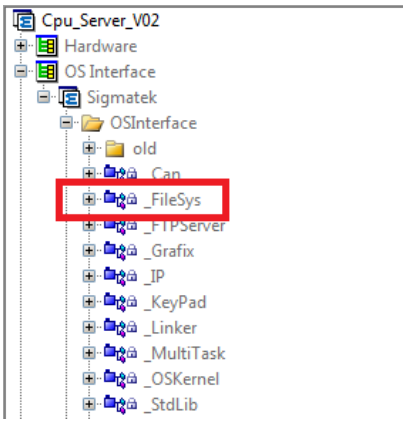
De werking van dataoverdracht is toch complexer. Zowel de PC als PLC zijn twee aparte controllers waar niet zomaar een connectie tussen is.
Voordat een client het bestand kan ophalen moet deze eerst toegang hebben tot een netwerk van de betreffende server, via bijvoorbeeld ethernet. Hierbij zal de client in hetzelfde IP bereik moeten zitten voordat deze met de server kan verbinden. Dit kan via de applicatie software van de PLC gedaan worden. De PLC moet beschikken over een ethernet aansluiten. Verschillende PLC's kunnen uitgebreid worden met speciale modules die over een ethernet aansluiting beschikken.

Wanneer de PLC toegang heeft tot het netwerk heeft deze nog niet de mogelijkheid om een bestand uit te lezen. Dit kan alleen wanneer een FTP verbinding tussen de client en server gelegd is. FTP (File Transfer Protocol) is een protocol die zorgt voor de het uitwisselen van een bestand tussen computers. FTP maakt deze verbinding over een TCP gebaseerde netwerk.
Zowel de PC als de PLC hebben beide vanuit zichzelf niet de mogelijkheid om een FTP verbinding op te bouwen.
Voordat een PC een FTP verbinding kan opbouwen moet deze beschikken over een FTP server. Een FTP server is een software pakket die vanaf internet gedownload kan worden en geïnstalleerd op de PC. In deze software wordt verbindingen gelegd met clients waarbij deze de aanvragen van de betreffende clients verwerkt.
De PLC moet voor het opbouwen van een FTP verbinding beschikken over een speciale module waar de FTP protocol in verwerkt is. Daarnaast beschikt deze module ook over een interne harde schijf. Deze harde schijf is dusdanig ontworpen dat een PLC als losstaande controller het bestand op kan slaan. Daarna kan met de applicatie software functies worden aangeroepen die het bestand uit kunnen lezen en omzetten in data. De functies zijn door de software ontwikkelaars van de applicatie software al ontwikkeld en in een speciale library gedefinieerd. Hierbij betreft het twee verschillende librarys, één met FTP functies en één met functies voor uitlezen van files.
Wanneer met de FTP functies het bestand binnen gehaald is en met de file functies het bestand is omgezet in data, dan kan de data met de normale PLC functie bewerkt worden.

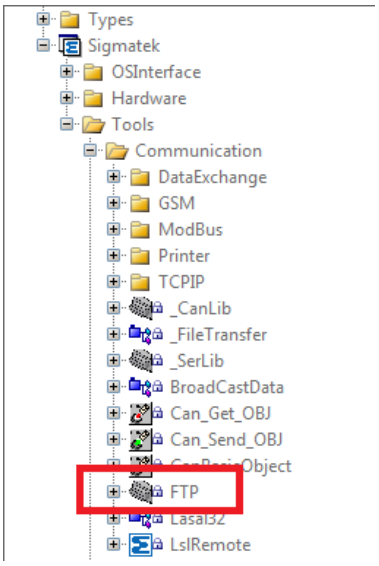


Figuur 17, overzicht van stappen voor het overdragen van een bestand.

Voor de opdracht werd er gebruik gemaakt van de Sigmatek PLC; de C-IPC. Deze PLC heeft een ingebouwde ethernet aansluiting en beschikt over de FTP protocol. Hiervoor hoefde geen modules aangeschaft te worden. Deze PLC maakt gebruik van de applicatie software Lasal class 2.
Lasal is een object georiënteerde applicatie software.
Binnen deze software zijn twee standaard library's gedefinieerd; FTP als _FileSys (voor het uitlezen van bestanden). Deze kunnen simpel gekopieerd worden naar het project, waarna de programmeur de mogelijkheid krijgt de betreffende functie aan te roepen.



Figuur 18a, _FilSys uit de library OS Interface.



Figuur 18b, FTP uit de Tools library.

In **bijlag E** zijn de betreffende omschrijvingen te vinden van de functies uit beide library's.

Aan de hand van het proces verloop uit figuur 17 worden een aantal FTP en File functies aangeroepen. Hieronder is te lezen hoe deze functies aangeroepen worden en welke inputs mee gegeven moeten worden.

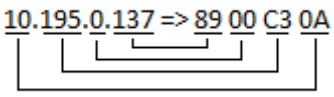
2.2.1.1 FTP functies

Voor het openen van een FTP verbinding wordt de functie *FTP_CONNECT* aangeroepen.

```
c_ftp.FTP_CONNECT(udIP:=, pUser:=, pPassword:=);
```

Figuur 19, functie aanroep voor het maken van een ftp verbinding.

Hierbij moeten er drie waardes meegegeven worden.
De variabel **udIP** zal, voor het maken van een FTP verbinding een hexadecimale waarde moeten verkrijgen aan de hand van het IP adres. Deze waarde wordt als volgt berekend.



Figuur 20, omrekenen van het ip-adres naar hexadecimale waarde.

In de FTP software op de PC kunnen gebruikers gedefinieerd worden. Hierbij wordt een gebruikers naam en een wachtwoord ingesteld. In de Lasal software moet bij deze functie beide mee gegeven worden. De gebruikers naam wordt meegegeven met de variabele **pUser** en het wachtwoord met **pPassword**.
In de standaard gedefinieerde functie worden de variabelen meegegeven en zal via het operating systeem een verbinding worden gelegd met de FTP server op de PC.

```
FUNCTION GLOBAL FTP::FTP_CONNECT
VAR_INPUT
  udIP     : UDINT;
  pUser    : ^CHAR;
  pPassword : ^CHAR;
END_VAR
VAR_OUTPUT
  dRetVal    : DINT;
END_VAR

//only connect, if there is no existing connection
if (bLoggedIn = FALSE) then
  dFTP_Socket := OS_FTP_Connect(udIP, pUser, pPassword);

  if (dFTP_Socket >= 0) then
    bLoggedIn := TRUE;      //connection established
    dRetVal := 0;
  else
    dRetVal := dFTP_Socket;   //otherwise return the error-code
  end_if;
else
  dRetVal := -10;         //there is already a connection existing
end_if;

END_FUNCTION //GLOBAL C_FTP::CONNECT
```

Figuur 21, de standaard gedefinieerde functie voor het maken van een ftp verbinding.

Wanneer de verbinding succesvol opgebouwd is, zal de 'return value' waarde 0 krijgen. Wanneer deze een negatieve waarde terug krijgt geeft dit aan dat er een storing opgetreden is.

Voordat het bestand vanaf de FTP server opgehaald kan worden zal eerst de transmissie bepaald moeten worden. Dit wordt gedaan door de functie *FTP_TRANSFTYPE*.

```
c_ftp.FTP_TRANSFTYPE(bASCII:=, bFile:=);
```

Figuur 22, functie aanroep voor het bepalen van de transmissie.

Hierbij wordt met de boolean **bASCII** bepaald of de transmissie in ASCII (waarde is 1) of binair (waarde is 0) zal zijn.
Met de boolean **bFile** wordt bepaald of het betreffende bestand de buffer is (waarde is 1) of de RAM geheugen de buffer (waarde is 0).

Wanneer de transmissie bepaald is kan de betreffende bestand met de functie *FTP_GET* opgehaald worden.

```
c_ftp.FTP_GET(pBuf_File:=, udLength:=, pServerFile:=);
```

Figuur 23, functie aanroep voor het ophalen van een bestand.

Hierbij wordt in de variabele **pBuf_File** het pad naar de interne harde schijf bepaald waarin het opgehaalde bestand wordt opgeslagen.
Met de variabele **udLength** wordt de maximale lengte van het bestand meegegeven.
In de variabele **pServerFile** wordt het pad naar het bestand, op de PC mee gegeven.
Dit geld ook voor de functie *FTP_PUT* maar dan wordt het bestand van de interne harde schijf naar de PC terug gezet.

```
c_ftp.FTP_PUT(pBuf_File:=, udLength:=, pServerFile:=);
```

Figuur 24, functie aanroep voor het terug sturen van het bestand.

Voor het sluiten van een FTP verbinding wordt de functie *FTP_CLOSE* aangeroepen.

```
c_ftp.FTP_CLOSE();
```

Figuur 25, functie aanroep voor het sluiten van de FTP verbinding.

2.2.1.2 File functies

Wanneer het bestand succesvol opgeslagen is op de interne harde schijf van de PLC kan het bestand omgezet worden in data. Hierbij zijn de volgende functies van belang:

Voor het openen van het bestand moet de functie *FileOpen* aangeroepen worden.

```
FileIO.FileOpen(path:=, create:=, Attribs:=);
```

Figuur 26, functie aanroep voor het openen van het bestand.

Hierbij wordt bij de variabele **path** het pad op de interne harde schijf gegeven. Deze moet hetzelfde zijn als het pad die gedefinieerd is in de variabele **pBuf_File** van de *FTP_GET* functie. Bij de *integer* **create** kan er gekozen worden uit drie waardes:

0: openen van een bestaande file.
1: creëer een nieuwe file ook als deze al bestaat.
2: gebruik de gebruiker gedefinieerde attributen van parameter “attrib”

Bij de variabele **Attribs** kan de gebruiker een attribuut definieerde voor het openen van het bestand. Hierbij kan hij bepalen of het bestand alleen gelezen mag worden, deze geschreven mag worden, ect. Wanneer bij de *integer* **create** aangegeven wordt dat een bestaande bestand geopend wordt zal de variabele **Attribs** met waarde 0 gedefinieerd moeten worden.
Wanneer het bestand geopend is zal deze bij de ‘return value’ een unieke waarde terug geven die het betreffende bestand aangeeft. Dit wordt de *Handle* genoemd.

Wanneer men het bestand wil uitlezen zal de functie *FileRead* aangeroepen moeten worden.

```
FileIO.FileRead(handle:=, data:=, size:=);
```

Figuur27, functie aanroep voor het uitlezen van het bestand

Hierbij moet bij de variabele **handle** de betreffende *Handle* meegegeven worden die verkregen is bij het openen van het bestand. De uitlezen data wordt opgeslagen in de variabele **data**. Bij **size** wordt de grote van de uitgelezen data bepaald.
Voor het terug schrijven naar het bestand wordt de functie *FileWrite* aangeroepen. Hierbij worden dezelfde parameters gebruikt als bij de functie *FileRead*.

```
FileIO.FileWrite(handle:=, data:=, size:=);
```

Figuur 28, functie aanroep voor het schrijven naar het bestand

Wanneer men klaar is met het uitlezen en het schrijven van het bestand wordt met de functie *FileClose* het bestand weer gesloten.

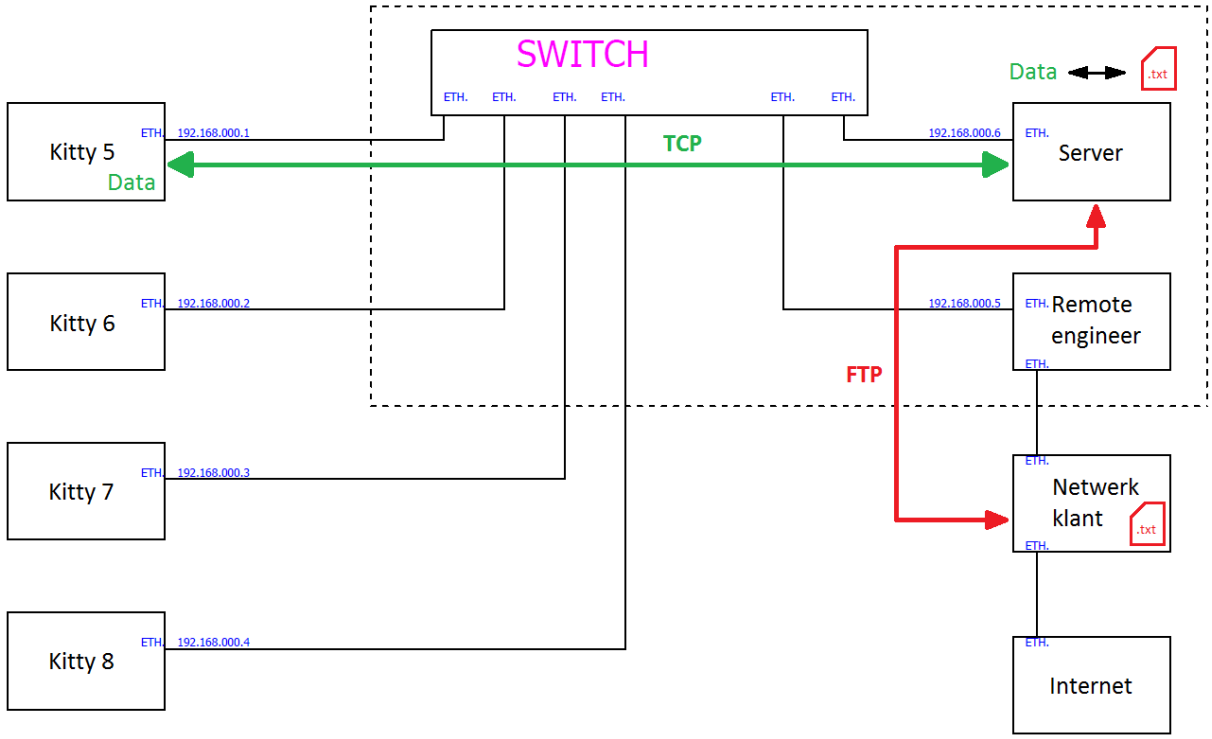
```
FileIO.FileClose(handle:=);
```

Figuur 29, functie aanroep voor het sluiten van het bestand

Hierbij moet de betreffende *Handle* meegegeven worden die verkregen is bij het openen van het bestand.

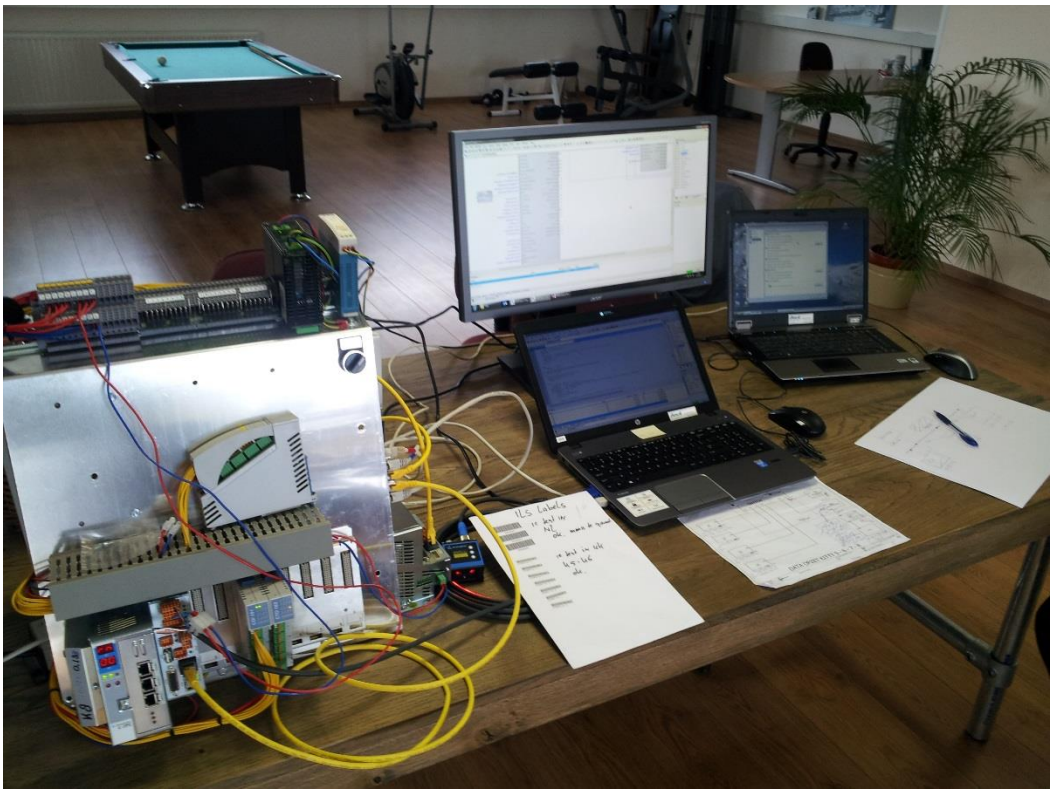
2.2.2 De opbouw

Bij het uitzoeken van de gemaakte software is het systeem voor data overdracht uitgebreider dan is uitgelegd in het hoofdstuk *onderzoek*.
Hierbij wordt er tussen de Kitty machines en de server gebruik gemaakt van een TCP verbinding. Deze verbinding wordt gebruikt om data tussen de Kitty's en de server uit te wisselen.
De server is de FTP client en het netwerk van de klant is de FTP server. Hierbij haalt de server software het tekstbestand op bij de klant en deze om in data. De Kitty's vragen de data op bij de server en passen deze aan.



Figuur 30, Datanetwerk Kitty's 5-6-7-8

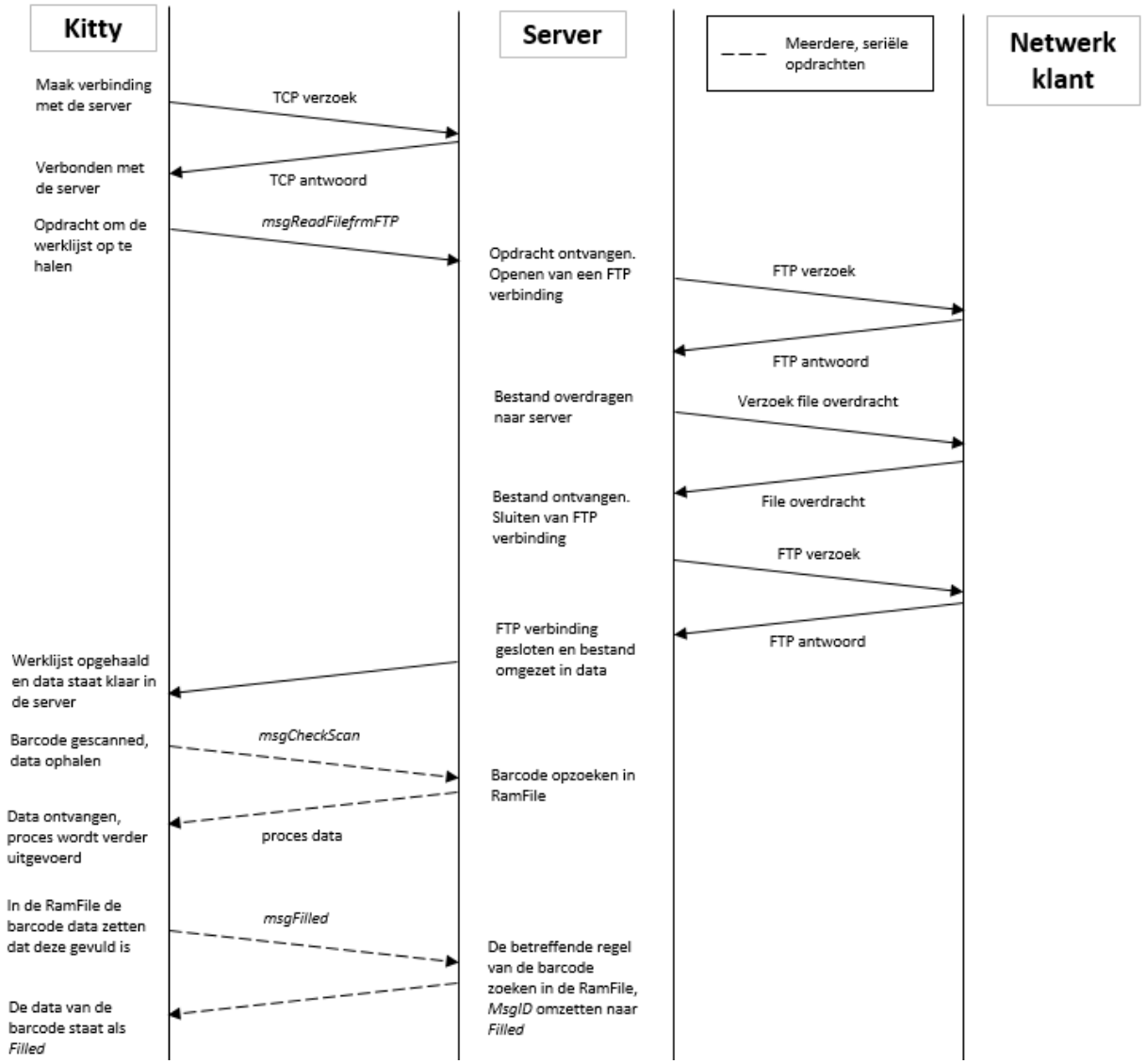
Voor het testen wordt de software van zowel de server als een kitty machine wordt één laptop gebruikt. Beide softwares worden in aparte PLC gedownload. Daarnaast wordt een tweede laptop gebruikt als FTP server.



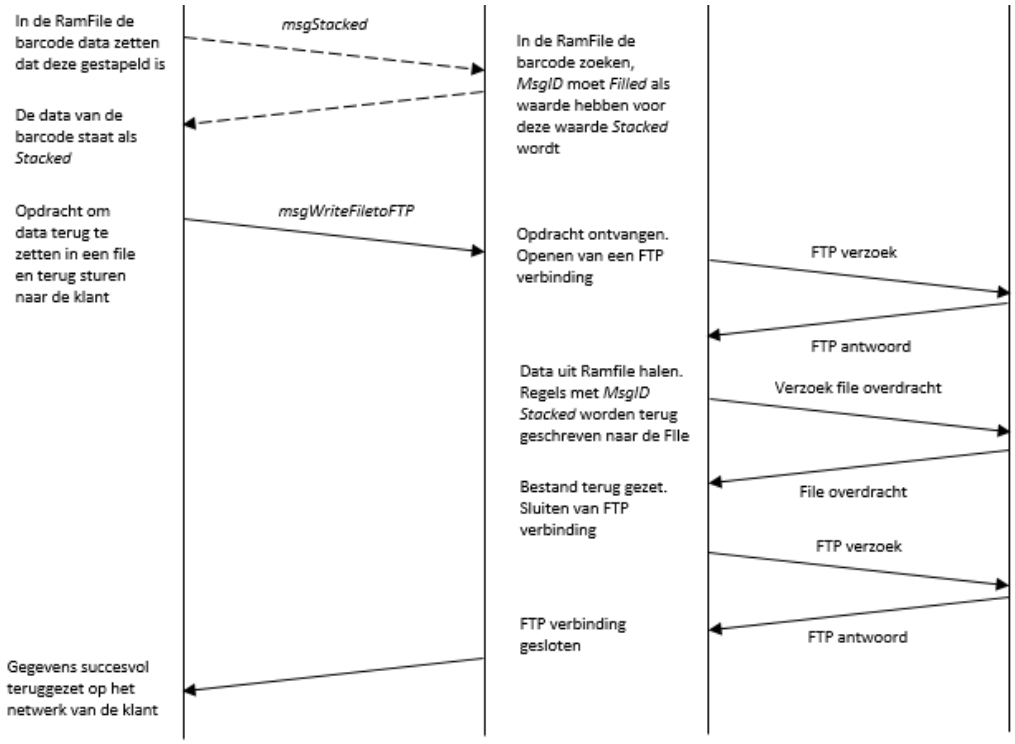
Figuur 31, totale test opstelling

2.2.3 De werking

Binnen dit systeem moet het programma een aantal stappen doorlopen voordat er dat data opgehaald kan worden, aangepast en teruggeplaatst.
Aller eerst wordt een TCP verbinding opgezet worden tussen de Kitty en de Server.
Als deze met elkaar verbonden zijn kan via een display bij de Kitty een commando gegeven worden om de werklijst op te halen bij het netwerk van de klant (in het geval van de test opstelling wordt alleen de betreffende waarde gestuurd die de functie triggerd).
Wanneer de Kitty toegang heeft tot de werklijst in de server kunnen er buisjes met een barcodes gescand worden waarbij, aan de hand van de betreffende barcode, een bepaalde handeling uitgevoerd moet worden (hier wordt alleen een barcode gescand). Na deze handeling zal een schaaltje met de betreffende inhoud en verdunning klaar staan waarbij de gegevens op het schaaltje geprint wordt. Binnen de software wordt na de print functie, in de data aangegeven dat een schaaltje gevuld is. Wanneer er meerdere schaaltjes gevuld zijn worden deze opgestapeld en wordt ook dit in de data verwerkt met een waarde “stacked”. De data kan pas omgezet worden tot een tekstbestand als de parameter deze waarde bevat. In het onderstaande schema wordt een overzicht gegeven over hoe het proces doorlopen worden vanaf begin tot eind.



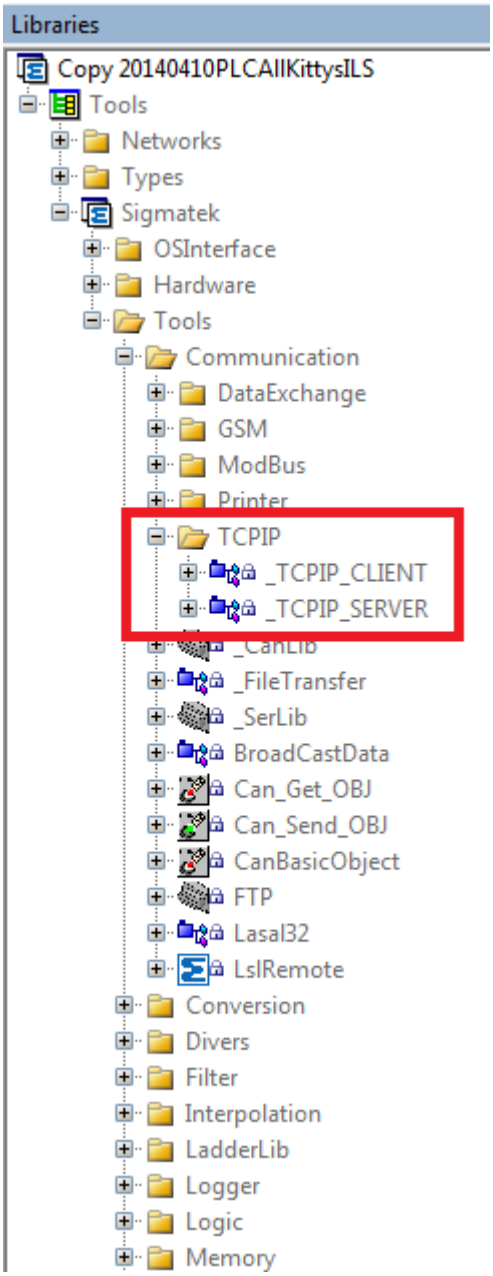
Figuur 32a, Proces verloop dataoverdracht



Figuur 32b, vervolg dataoverdracht

TCP opbouw Kitty

Voordat er data uitgewisseld kan worden tussen client (Kitty) en server moet er een TCP verbinding gemaakt worden. Binnen de Sigmatek software *Lasal Class* is een standaard library gedefinieerd waarin verschillende mogelijkheden tot communicatie beschreven zijn. Hierin zijn ook de TCP classes **_TCPIP_CLIENT** en **_TCPIP_SERVER** gedefinieerd. Zoals de namen al aangeven moet de **_TCPIP_CLIENT** class gekopieerd worden in het project van de client (in dit geval de Kitty) en **_TCPIP_SERVER** in het project van de server. Wanneer classes in het project netwerk geplaats worden heeft het programma toegang tot de TCP functies



Figuur 33, locatie TCPIP classes in de standaard library.

Binnen de Kitty software wordt de verbinding, met de server gemaakt doormiddel van de functie *AddConnection*. Om een verbinding te leggen met de server moeten de volgende inputs voor de functie gedefinieerd worden; **pIPAddress**, **pCallBackThis**, **pCallbackMeth** en **udTimeout**. De output wordt gedefinieerd als **dHandle**.

```
160  FUNCTION VIRTUAL GLOBAL TCPInterface::Background
161    VAR_INPUT
162      EAX    : UDINT;
163    END_VAR
164    VAR_OUTPUT
165      state (EAX)    : UDINT;
166    END_VAR
167
168  //  mOnline := tcp.IsConnected(dHandle:=dHandle);
169    tcp := tcp.Control.read();
170    online2 := (tcp = _STATE_RECV) | (tcp = _STATE_Send);
171    online := online2;
172    if dHandle <> 0 then
173      testonline := tcp.IsConnected(dHandle:=dHandle);
174    else
175      testonline :=0;
176    end_if;
177    if WaitForConnection & ops.tAbsolute - udToTimer > 300 then
178   //   Online :=0;
179    end_if;
180
181    case stepper of
182
183    0:
184      if HvZTest then
185        dHandle := tcp.AddConnection(pIPAddress:="10.10.150.10", pCallbackThis:=this, pCallbackMeth:=#MyCallBack() , udTimeout:=0);
186        Addcntr +=1;
187      else
188        dHandle := tcp.AddConnection(pIPAddress:="10.0.0.30", pCallbackThis:=this, pCallbackMeth:=#MyCallBack() , udTimeout:=0);
189        Addcntr +=1;
190      end_if;
191      WaitForConnection :=0;
192      RetryCntr :=0;
193      stepper +=1;
194      udTimer := ops.tAbsolute;
```

Figuur 34, functie aanroep voor het maken van een TCP verbinding.

De input **pIPAddress** moet het IP adres van de betreffende server bevatten. De input **pCallbackThis** bevat een *this-pointer* naar de object die aangeroepen moet worden. De input **pCallbackMeth** bevat een pointer naar de *Call back* methode. Hiermee kan aangeven worden waar de verstuurde data van de server binnen moet komen. Voor deze methode zijn de transfer parameters **pData** en **udSize** nodig. In de software van de Kitty zal de functie *MyCallBack* aangeroepen worden. De input **udTimeout** is de tijd in milliseconden waarbij de verbinding gesloten moet worden. Wanneer deze waarde **0** heeft, wordt deze optie uitgeschakeld. De output **dHandle** is een pointer naar de connectie data zoals de Socket, IP adres, timout, ect.

Om ervoor te zorgen dat de server weet met welke client hij moet communiceren, moet aan de kant van de client het IP adres meegegeven worden. Deze kan opgevraagd worden via het operating system. Dit wordt gedaan met de aanroep van **OS_CILGet()**. Hierbij wordt alle **tcp_user** gegevens opgevraagd en via een pointer in de globale **lsl_tcp_user** struct geplaatst.

```
5   FUNCTION VIRTUAL GLOBAL MyIP::Init
6     VAR
7       IPBuffer       : ARRAY[0..15] of CHAR;
8       udLen          : UDINT;
9       lsl_tcp_user     :^LSL_TCP_USER;
10    END_VAR
11
12    if _firstscan then
13
14        OS_CILGet("TCP_USER", #lsl_tcp_user);
15        OS_TCP_USER_IPINFO(1, 1, #IP);
16
17      //IP-Addr in dotted format
18      OS_TCP_USER_ULONGTOSTR(#ipbuffer[0], sizeof(ipbuffer), IP);
19
20      //write IP-Addr to String
21      udLen := _StrLen(#ipbuffer[0]);
22      strMyIP.writedataoff(udLen, 0, #ipbuffer[0]);
23
24      len :=_strlen(src:=#ipBuffer[0]);
```

Figuur 35, aanroep van functie OS_CILGet

Name	Value	SetValue	Format	Type	Address
lsl_tcp_user	16#004C5948		Default	^lsl_tcp_user	%m0662FCB4
^	LSL_TCP_USER		Default	STRUCT	%m004C5948
udVersion	9		Default	UDINT	%m004C5948
tcp_user_socket	16#0047FB70		Default	^void	%m004C594C
tcp_user_closesocket	16#00480142		Default	^void	%m004C5950
tcp_user_connect	16#0048023E		Default	^void	%m004C5954
tcp_user_listen	16#00480780		Default	^void	%m004C5958
tcp_user_accept	16#004809A9		Default	^void	%m004C595C
tcp_user_recv	16#00480CC1		Default	^void	%m004C5960
tcp_user_send	16#00480EBD		Default	^void	%m004C5964
tcp_user_shutdown	16#00481281		Default	^void	%m004C5968
tcp_user_getpeerip	16#00481308		Default	^void	%m004C596C
tcp_user_getpeerport	16#004813FD		Default	^void	%m004C5970
tcp_user_getsockip	16#0048144F		Default	^void	%m004C5974
tcp_user_getsockport	16#00481544		Default	^void	%m004C5978
tcp_user_toip	16#00481596		Default	^void	%m004C597C
tcp_user_strtoulong	16#004816BF		Default	^void	%m004C5980
tcp_user_ulongtostr	16#00481709		Default	^void	%m004C5984
tcp_user_getCOMLinkPort	16#0048175E		Default	^void	%m004C5988
tcp_user_geterrorstring	16#00481817		Default	^void	%m004C598C
tcp_user_netstat	16#00481D93		Default	^void	%m004C5990
tcp_user_nread_available	16#004810A4		Default	^void	%m004C5994
tcp_user_socket_ex	16#00480097		Default	^void	%m004C5998
tcp_user_getservbyname	16#004817D6		Default	^void	%m004C599C
tcp_user_ipinfo	16#00482192		Default	^void	%m004C59A0
tcp_user_setsockopt	16#00480C5C		Default	^void	%m004C59A4
tcp_user_ping	16#00482205		Default	^void	%m004C59A8
udp_user_socket	16#0048220F		Default	^void	%m004C59AC
udp_user_bind	16#004822D2		Default	^void	%m004C59B0
tcp_user_recvfrom	16#00482393		Default	^void	%m004C59B4
tcp_user_sendto	16#004825D9		Default	^void	%m004C59B8
tcp_user_ipinfo_reserved	16#00482015		Default	^void	%m004C59BC
tcp_user_select	16#004827FD		Default	^void	%m004C59C0
tcp_user_ioctlsocket	16#00482B1D		Default	^void	%m004C59C4
tcp_user_initFastInterface	16#454C4449		Default	^void	%m004C59C8
tcp_user_sendfast	16#00000000		Default	^void	%m004C59CC
tcp_user_getLinkStatus	16#444E4550		Default	^void	%m004C59D0

Figuur 36, struct lsl_tcp_user

Met de functie **OS_TCP_USER_IPINFO()** wordt het IP adres van de client via de **tcp_user_ipinfo** pointer uit de **lsl_tcp_user** struct opgehaald en in de variabele, waar het adres van **IP** naar verwijst, geplaatst. Met de functie **OS_TCP_USER_ULONGTOSTR()** wordt de variabele **IP** omgezet een volledige dot formaat IP adres (xxx.xxx.xxx.xxx) en in de variabele **ipbuffer** geplaatst.

```
5   FUNCTION VIRTUAL GLOBAL MyIP::Init
6     VAR
7       IPBuffer       : ARRAY[0..15] of CHAR;
8       udLen          : UDINT;
9       lsl_tcp_user     :^LSL_TCP_USER;
10    END_VAR
11
12    if _firstscan then
13
14        OS_GILGet("TCP_USER", #lsl_tcp_user);
15        OS_TCP_USER_IPINFO(1, 1, #IP);
16
17    //IP-Addr in dotted format
18    OS_TCP_USER_ULONGTOSTR(#ipbuffer[0], sizeof(ipbuffer), IP);
19
20    //write IP-Addr to String
21    udLen := _StrLen(#ipbuffer[0]);
22    strMyIP.writedataoff(udLen, 0, #ipbuffer[0]);
23
24    len :=_strlen(src:=#ipBuffer[0]);
```

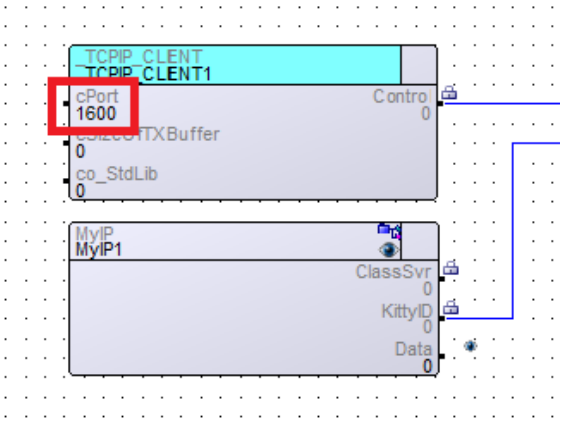
Figuur 37, aanroep van de functies OS_TCP_USER_IPINFO en OS_TCP_USER_ULONGTOSTR

Om in het netwerk de betreffend IP adres te weergeven wordt de IP adres uit **ipbuffer** geschreven naar een string.

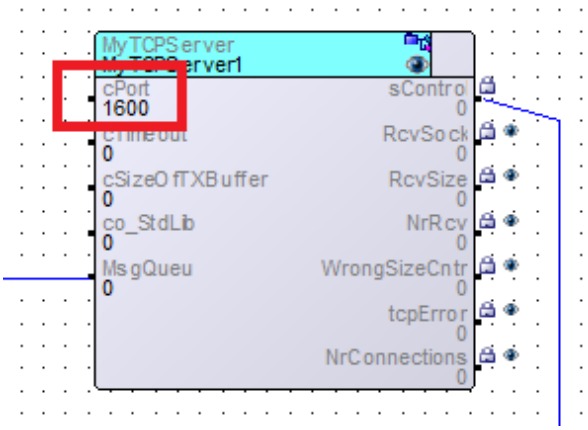
```
12    if _firstscan then
13
14         OS_CILGet("TCP_USER", #lsl_tcp_user);
15         OS_TCP_USER_IPINFO(1, 1, #IP);
16
17       //IP-Addr in dotted format
18       OS_TCP_USER_ULONGTOSTR(#ipbuffer[0], sizeof(ipbuffer), IP);
19
20       //write IP-Addr to String
21       udLen := _StrLen(#ipbuffer[0]);
22       strMyIP.writedataoff(udLen, 0, #ipbuffer[0]);
23
24       len :=_strlen(src:=#ipBuffer[0]);
25
26       case IPBuffer[len-1] of
```

Figuur 38, IP-adres naar een string

Om ervoor te zorgen dat de *TcpClient* succesvol verbonden wordt met de *TcpServer* moet bij beide classes dezelfde poort waarde gedefinieerd worden.



Figuur 39a, TCPIP Client in de Kitty software



Figuur 8b, TCPIP server in Server software

Wanneer de Kitty verbonden is met de server, zal de server **Control** (van het blok **_TCPIP_CLIENT**) in **_STATE_Send** of **_STATE_RECV** staan, afhankelijk of de Kitty data verzend of data ontvangt. Server **Online** krijgt waarde 1 als de server **Control** in één van deze twee states staat.

```
160   FUNCTION VIRTUAL GLOBAL TCPInterface::Background
161     VAR_INPUT
162       EAX    : UDINT;
163     END_VAR
164     VAR_OUTPUT
165       state (EAX)    : UDINT;
166     END_VAR
167
168   //  online := tcp.IsConnected(dHandle:=dHandle);
169     tcp := tcp.Control.read();
170     online2 := (tcp = _STATE_RECV) | (tcp = _STATE_Send);
171     online := online2;
172     if dHandle <> 0 then
173       testonline := tcp.IsConnected(dHandle:=dHandle);
174     else
175       testonline :=0;
176     end_if;
```

Figuur 9, functie Background in blok TCPInterface, controle connectie

Met de functie **IsConnected()** is het ook mogelijk te controleren of de client en server verbonden zijn met elkaar. Hierbij moet de **dHandle** (de return waarde van **AddConnection**) mee gegeven worden. Als er een connectie is zal de return waarde (in dit geval **testonline**) **true** worden. In de software van Kitty wordt **testonline** verder niet gebruikt.

```
160  FUNCTION VIRTUAL GLOBAL TCPInterface::Background
161    VAR_INPUT
162      EAX    : UDINT;
163    END_VAR
164    VAR_OUTPUT
165      state (EAX)    : UDINT;
166    END_VAR
167
168  //  mOnline := tcp.IsConnected(dHandle:=dHandle);
169    tcp := tcp.Control.read();
170    online2 := (tcp = _STATE_RECV) | (tcp = _STATE_Send);
171    online := online2;
172    if dHandle <> 0 then
173      testonline := tcp.IsConnected(dHandle:=dHandle);
174    else
175      testonline :=0;
176    end_if;
177    if waitforconnection & ops.tAbsolute - udTimer > 300 then
```

Figuur 41, controle connectie

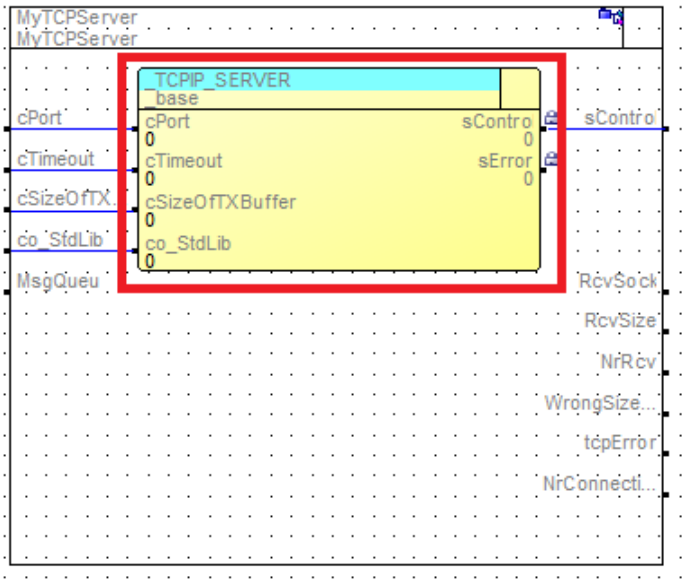
Om de verbinding met de server te verbreken moet de functie **DelConnection()** aangeroepen moet worden. In het blok **TCPInterface** zal deze aangeroepen moeten worden wanneer de server **Stepper** de waarde **3** heeft. Voor deze functie moet de betreffende **dHandle** (de return waarde die mee gegeven is bij het maken van een TCP verbinding) mee gegeven worden.

```
220    3:
221
222      tcp.DelConnection(dHandle:=dHandle);
223      DelCnt := 1;
224      udTimer := ops.tAbsolute;
225      stepper +=1;
```

Figuur 42, aanroep van functie DelConnection

TCP opbouw Server

In de software van de server is het blok **MyTCPServer** een overerving van de class **_TCPIP_SERVER**. Daarbij is de base class **_TCPIP_SERVER** ondergebracht in het netwerk van het blok. Dit is niet specifiek nodig voor een TCP verbinding.

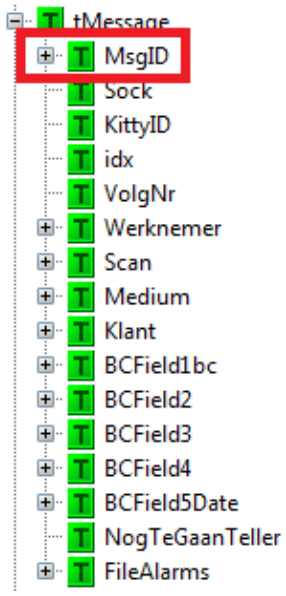


Figuur 43, netwerk van MyTCPServer

Voor het maken van een verbinding met de *TcpClient* hoeft er in de software van de server niets geprogrammeerd te worden, dit is standaard geïmplementeerd in **_TCPIP_SERVER** class. Dit geld ook voor het binnen halen van data. Voor het uitlezen van de data zal de functie **Response()** aangeroepen moeten worden.

Bestand ophalen via FTP

Zoals in figuur 32 wordt aangegeven moet de Kitty de opdrachten in een bepaalde volgorde doorlopen. Deze opdrachten wordt in het blok **TCPInterface** verwerkt door een bepaalde functie aan te roepen. Voor elke opdracht is er een andere functie. Alle gegevens worden in standaard bericht geplaatst. Deze is als volgt opgebouwd:



Figuur 44, structuur bericht

De betreffende opdracht wordt in de variabele ***MsgID*** van het bericht geplaatst. Deze variabele wordt bij de server uitgelezen waarbij de specifiek opdracht wordt uitgevoerd.
Wanneer een opdracht opgegeven is wordt het gehele bericht in een buffer geplaatst.

```
342  FUNCTION GLOBAL TCPInterface::ReadFilefrmFTP
343  VAR
344    TmpMsg   : tMessage; // 1
345  END_VAR
346
347
348      if Online then // 2
349        FileFunctionBusy :=1;  // 3
350        _memset(dest:=#TmpMsg, usByte:=0, cntr:=sizeof(TmpMsg)); // 4
351        ID := ID.Read(); // 5
352        TmpMsg.KittyID := ID; // 6
353        TmpMsg.MsgID := msgReadFilefrmFTP; // 7
354        AddMsgToBuffer(Msg:=#TmpMsg); // 8
355        LogMessage(Msg:=TmpMsg);    // 9
356      end_if;
357  END_FUNCTION
```

Figuur 45, functie voor het opgeven van de opdracht van de server. In dit geval het ophalen en inlezen van het bestand.

De opdrachten worden via de TCP verbinding verstuurd naar de server. In het blok **TCPInterface** wordt op de achtergrond de functie **Background** om de 100 ms gescand. In deze functie wordt er gekeken of er een verbinding is tussen de *TCPclient* en *TCPserver*. Wanneer deze met elkaar verbonden zijn wordt de functie **ReadMsgFrmBuffer()** aangeroepen die het bericht uit de buffer haalt en in een variabele **SendMsg** plaats.

Met de functie **SendData** wordt het bericht van de Kitty naar de server gestuurd. Hierbij moet het betreffende bericht meegegeven worden aan de variabele **pData**. De grote van het bericht wordt meegegeven aan de variabele **udSize**. Bij de variabele **dHandle** wordt de betreffende *handle* van de connectie mee gegeven. Bij de *integer* **bDirect** wordt bepaald of het bericht direct gestuurd moet

worden (waarde is 1) of eerst in een buffer geplaatst moet worden en pas de volgende cyclus verstuurd moet worden (waarde is 0) .

In dit geval wordt het bericht in **SendMsg** verstuurd met de grote van de type **tMessage**. De **handle** is de betreffende *handle* die verkregen is bij het opbouwen van de TCP verbinden en de data moet direct verzonden worden.

```
if Online then
  if ReadMsgFrmBuffer(Msg:=#SendMsg) then   //hebben we data?
    if SendMsg.MsgID$udint = 123 then
      SendResult := tcp.SendData(pData:=#SendMsg, udSize:=1, dHandle:=dHandle, bDirect:=1);
    else
      SendResult := tcp.SendData(pData:=#SendMsg, udSize:=sizeof(tMessage), dHandle:=dHandle, bDirect:=1);
    end_if;
    if SendResult = TCP_CLT_SEND_OK then
      DeleteMsgFromBuffer();
    else
      SendResultError := SendResult;
      ReSendCntr1 +=1;
    end_if;
  end_if;
end_if;
```

Figuur 46, versturen van de data naar de server

In de software van de server wordt, na dat het bericht verzonden is, de functie **Response** aangeroepen die het bericht ontvangt. Deze kijkt of het bericht de juiste grote heeft waarna deze de functie **DecodeMsg** aanroept. Deze functie leest het betreffende bericht uit en slaat deze op in de variabele **ActMsg**. In een speciaal blok waar de berichten behandeld worden zal de ontvangen **MsgID** uitgelezen worden. Hier wordt bepaald de betreffende functie aangeroepen die de opdracht moet uitvoeren.

```
32    1:
33      case ActMsg.MsgID of
34        msgCheckScan:
35          Stepper := 100;
36
37        msgGetIdx:
38          Stepper := 200;
39
40        msgFilled:
41          Stepper := 300;
42
43        msgStacked:
44          Stepper := 400;
45
46        msgGetNext:
47          Stepper := 500;
48
49        msgReadFilefrmFTP:
50          Stepper := 600;
51
52        msgWriteFiletoFTP:
53          Stepper := 700;
54
55        msgEraseDB:
56          Stepper := 800;
57
58        msgReadFilefrmUSB:
59          Stepper := 900;
60
61        msgWriteFileToUSB:
62          Stepper := 1000;
63
64        msgReqPrintData:
65          Stepper := 1100;
66
```

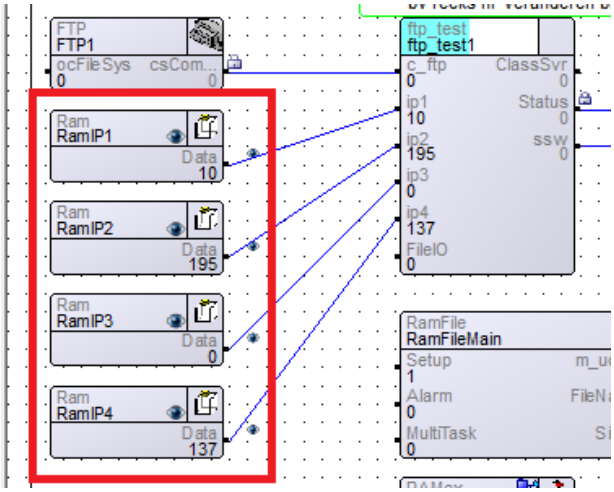
Figuur 47, lijst met opdrachten

De eerste opdracht (**msgReadFilefrmFTP**) betref uit ophalen en uitlezen van het bestand. Hierbij zal eerst een FTP verbinding opgebouwd worden met de FTP server.

```
1:
  Status :=0; // 1
  ip1 :=ip1.read(); // 2
  ip2 :=ip2.read();
  ip3 :=ip3.read();
  ip4 :=ip4.read();


  udIP := to_udint((ip4 * 16777216) + (ip3 * 65536 ) + (ip2 * 256) + ip1); // 3

    pUser := "kitty"; // 4
    pPassword :="ftpWorksheets1"; // 5


  result := C_FTP.FTP_CONNECT(udIP:= udIP, pUser:= pUser, pPassword:= pPassword); // 6
  if result < 0 then // 7
    Status := result; // 8
    C_FTP.FTP_CLOSE(); // 9
    ssw := 0; // 10
    return;
  else
    ssw :=10; // 11
  end_if;
```

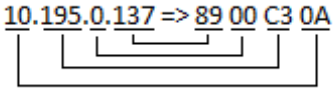
Figuur 48, opbouwen van een FTP verbinding

Hierbij wordt het IP-adres van de klant bepaald door een 4 RAM geheugens die zich in het netwerk begeven.



Figuur 59, netwerk waar het IP-adres van de klant gedefinieerd is.

Het IP-adres wordt bepaald door de waardes van alles **IP's** bij elkaar op te tellen, waarbij eerst **IP4** wordt vermenigvuldigen met **16777216** (2^24), **IP3** met **65526** (2^16), **IP2** met **256** (2^8). De uitkomst zal in dit geval de volgende waarde verkrijgen; **2298478592** + **0** + **49920** + **10** = **2298528522**. Omgerekend is de hexadecimale waarde **8900C30A**.



In figuur 48 is ook te zien dat de betreffende, gedefinieerde gebruikersnaam en wachtwoord zijn opgegeven als "kitty" en "ftpWorksheets1"

Wanneer de verbinding succesvol is opgebouwd wordt in de volgende stap het bestand opgehaald.\

```
136       10: //Kopieer file van server naar de IPC
137
138         result := C_FTP.FTP_TRANSFTYPE(0,1);  // 1, received data to a file
139
140         udLength := 80;
141         pBuf_File := "c:\\data\\limstoki.txt";  // 2, dit is in the cipc c schijf
142         pServerFile:="/Worklist.txt";  // 3, dit is de lokatie in het netwerk van de klant
143         result := C_FTP.FTP_Get(pBuf_File, udLength, pServerFile); // 4
144
145   //eerst oude old verwijderen
146
147         if result > 0 then  //rename file
148
149             c_ftp.FTP_DELE(pFile:="Worklist.txt");  // 5
150
151   //tijdelijk niet renamen voor test
152           Status := 0; // 6
153           if Status = 0 then
154             status :=1; // 7
155           else
156             Status := -2; // 8, renamen lukt niet
157           end_if;
158         else
159           Status := -1;   // 9, wrklst.k2 bestaat niet
160         end_if;
161
162         C_FTP.FTP_CLOSE(); // 10
163         ssw :=0; // 11
164
```

Figuur 60, ophalen van het bestand

Wanneer het bestand succesvol opgehaald is wordt met de functie *FTP_DELE* heb betreffende bestand bij de klant verwijderd. Op deze manier hebben de andere Kitty's niet de mogelijkheid om het bestand nogmaals op te halen.

Daarna zal een functie aangeroepen worden die het betreffende bestand uitleest en data in een speciale RAM geheugen plaats.

```
228   FUNCTION DataHandlerServer::Read_File
229   VAR
230     i : dint;
231   END_VAR
232
233   //Lees de worklist en stop deze in rfMain
234     error :=0; // 1
235     File.Handle := FileIO.FileOpen(path:="c:\\data\\limstoki.txt", create:=0, Attribs:=0); // 2
236     File.Offset :=0; // 3
237     File.Length := FileIO.FileLength(handle:=file.handle); // 4
238     valid :=1; // 5
239     idx :=0; // 6
240     writeidx :=0; // 7
241
242     Year := Year.read(); // 8
243     DayNr := DayNr.read(); // 9
244
245     if file.length > 0 then // 10
246       while file.offset < file.length & valid = 1 do // 11
247         Offset := ReadLine(Line:=(#lines[0])$^char, MaxLength:=200); // 12
248         if Offset < 0 then
249         // einde file
250           valid := false; // 13
251         else
252           File.Offset += Offset; // 14
```

Figuur 61, functie Read_file

Is stap 2 uit figuur 61 is te zien dat heb bestand met een *FileOpen* geopend wordt. In stap 3 wordt een variabele **offset** aan het begin gedefinieerd met waarde 0. Deze is heel belangrijk omdat de betreffende regels uit het bestand in de RAM geheugen worden geplaats doormiddel van deze *offset*. Wanneer dit niet gedaan wordt zal elke oude regel overschreven worden door de nieuwe regel. Door de offset zal de alle data als één grote regel achter elkaar in de RAM geplaatst worden.
In stap 12 wordt de functie **ReadLine** aangeroepen.
In deze functie wordt met de functie *ReadFile* elke regel uit het tekstbestand uitgelezen.

```
345  FUNCTION DataHandlerServer::ReadLine
346    VAR_INPUT
347      Line   : ^CHAR;
348      MaxLength    : DINT;
349    END_VAR
350    VAR_OUTPUT
351      NrRead   : DINT;
352    END_VAR
353    var
354      ii   :dint;
355    end_var;
356
357    ii :=0; // 12.1
358
359    _memset(line,0,to_udint(MaxLength)); // 12.2
360    while ii < MaxLength do
361      FileIO.FileRead(handle:=File.Handle, data:=(Line+ii), size:=1); //12.3
362      if (line+ii)^ = 10 then
363        exit; // 12.4
364      end_if;
365      ii +=1; // 12.5
366    end_while;
367    if ii = MaxLength then
368      ii := -Maxlength; //12.6
369    end_if;
370    NrRead := ii; //12.7
371
372  END_FUNCTION
```

Figuur 62, functie voor het uitlezen van de regels

In deze functie wordt 200 maal één karakter van het tekstbestand uitgelezen. Na het uitlezen van de karakter wordt de waarde **ii** met 1 verhoogd zodat de volgende karakter uitgelezen kan worden.

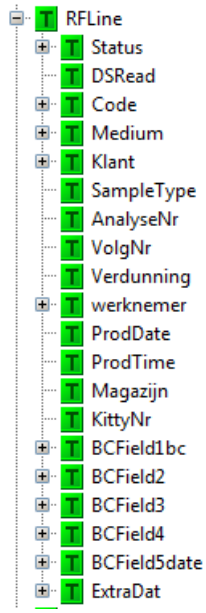
De complete uitgelezen regel wordt in de array **lines** gestopt. Daarna wordt de offset in stap 14 van figuur 61 per regel met waarde 200 verhoogd.

```
253  Loop:
254          RFMain.GetDataAt(p_us_data:=#Line$usint, ud_size:=sizeof(Line), ud_at:=writeidx * sizeof(Line)); // 15
255          if Line.Status = _empty then // 16
256            // Not used, fill in
257            _memset(dest := #Line,usByte :=0,cntr:= sizeof(line));  // 17, clear line
258
259            Line.Status := _fileread; // 18, ingelezen
260            ptr^[writeidx].Status := _fileread; // 19
261            Line.DSRead := (1000 * Year) + DayNr; // 20
262
263            //dest,src,cntr
264            _memcpy((#Line.Code)$^char,(#lines[1])$^char,10); // 21
265            _memcpy((#Line.BCField1bc)$^char,(#lines[0])$^char,24); // 22
266
267  //TODO medium checked op leading spaties en zonodig schuiven.
268
269            _memcpy((#Line.Medium)$^char,(#lines[61])$^char,6); // 23
270
271          //SampleType to db
272            _memset(dest := #tmpstr,usByte :=0,cntr:= 16);  // 24, clear tmpstr
273            _memcpy(#tmpstr,(#Line.BCField1bc[0])$^char,1);  // 25
274            Line.SampleType := to_usint(F.ConvAscii2Value(Source:=#tmpstr$usint)); // 26
275            ptr^[writeidx].SampleType := Line.SampleType; // 27
276
277            // 28, Analyse Nr to db
278            _memset(dest := #tmpstr,usByte :=0,cntr:= 16);  //clear tmpstr
279            _memcpy(#tmpstr,(#Line.BCField1bc[1])$^char,8);
280            Line.AnalyseNr := F.ConvAscii2Value(Source:=#tmpstr$usint);
281            ptr^[writeidx].AnalyseNr := Line.AnalyseNr;
282
283            // 29, VolgNr to db
284            _memset(dest := #tmpstr,usByte :=0,cntr:= 16);  //clear tmpstr
285            _memcpy(#tmpstr,(#Line.BCField1bc[9])$^char,2);
286            Line.VolgNr := F.ConvAscii2Value(Source:=#tmpstr$usint);
```

Figuur 63, vervolg van de functie Read_File

Bij elke regel wordt in het RAM geheugen gekeken of de betreffende locatie in de RAM gevuld is. Dit wordt gedaan met de functie *GetDataAt*. Hierbij wordt de opgehaalde gegevens in de variabele **Line** geplaats. De grote van de data die opgehaald moet worden is de grote van de variabele **Line** zelf. De offset binnen de RAM geheugen wordt bepaald door een index waarde **writeidx** te vermenigvuldigen met de grote van de variabele **Line**.

Elke regel in het tekstbestand heeft een eigenstukje data, zie figuur 64.



Figuur 64, type RFLine

Wanneer een regel niet eerder gevuld is zal de functie *Read_File* de betreffende regel uit elkaar halen en alle types uit figuur 64 apart opvullen en de opgehaalde data uit het bestand.

```
288             // 30, Verdunning to db
289             _memset(dest := #tmpstr,usByte :=0,cntr:= 16);  //clear tmpstr
290             _memcpy(#tmpstr,(#Line.BCField1bc[11])$^char,1);
291             Line.Verdunning := F.ConvAscii2Value(Source:=#tmpstr$usint);
292             ptr^[writeidx].Verdunning := to_usint(Line.Verdunning);
293
294   //De print velden:
295             _memcpy(ptr1:=#Line.BCField2, ptr2:=#lines[24], cntr:=6); // 31
296             _memcpy(ptr1:=#Line.BCField3, ptr2:=#lines[30], cntr:=6); // 32
297             _memcpy(ptr1:=#Line.Klant, ptr2:=#lines[36], cntr:=6); // 33
298
299             //Extra data to db
300             _memset(dest:=#Line.ExtraDat[0], usByte:=32, cntr:=129); // 34
301             _memset(dest:=#Line.werknemer[0], usByte:=32, cntr:=3); // 35
302             _memcpy(ptr1:=#Line.ExtraDat[0], ptr2:=#Lines[70], cntr:=129); // 36
303             for i := 0 to 129 Do // 37
304               if line.ExtraDat[i] = 13 | line.ExtraDat[i] = 10 then
305                 Line.ExtraDat[i] := 32;
306               end_if;
307             end_for;
308
309             RFMain.SetDataAt(p_us_data:=#Line$usint, ud_size:=sizeof(Line), ud_at:=writeidx * sizeof(Line)); // 38
310
311           else // 39
312             writeidx +=1;
313             if writeidx < (MaxLines-1) then
314               goto loop;
315             else
316               Error := 1;   //Database vol
317               goto l_error;
318             end_if;
319           end_if;
320           //dest,src,cntr
321
```

Figuur 65, vervolg functie Read_File

In stap 38 van figuur 65 wordt de betreffende data van de regel met een **SetDataAt** terug geplaats in de RAM geheugen.
Bij elke uitgelezen regel wordt de waarde van **writeidx** met waarde 1 verhoogd. Hierdoor wordt een nieuwe regel achter de laatste regel in het RAM geheugen geplaats.

Wanneer het bestand succesvol uitgelezen is en de betreffende data in de RAM geheugen is opgeslagen zal er een bericht terug gestuurd worden naar de Kitty.

```
//*************************************************************
//   ReadFilefrmFTP
  600:
    DataH.FileAlarms.Write(input:=0);
    DataH.ssv.Write(input:=10);
    Stepper +=1;

  601:
    if DataH.ssv.Read() = 0 then
      SendResult := tcp.SendData(pData:=#ActMsg, udSize:=sizeof(ActMsg), dSock:=ActMsg.Sock, bDirect:=1);
      if SendResult > 0 then
          //mooi, klaar
          MsgQue.DeleteMessage();
          Stepper :=0;
       else
          //Status is aangepast in de database, alleen de kitty veet het nog niet.
          udTOTimer := ops.tAbsolute;
          RetryCntr :=0;
          Stepper +=1;
       end_if;
    end_if;
```

Figuur 66, terug sturen van de data wanneer het bestand opgehaald is

In de Kitty zal aangegeven worden dat het bestand is opgehaald. Hierna zal gegevens uitgewisseld worden aan de hand van barcodes

Bestand terug sturen via FTP

Tijdens het proces worden de betreffende regels met data uitgelezen en worden de nieuwe data terug geplaatst in het RAM geheugen in de software van de server. Wanneer het bestand terug geschreven moet worden in de software van de Kitty de opdracht **msgWriteFiletoFTP** naar de server toe gestuurd worden.
In de software wordt de betreffende opdracht binnen gehaald en zal de functie *Write_File* aangeroepen worden.

```
FUNCTION DataHandlerServer::Write_File
var
  wline    : RFLine;
  eLine    : RFLine;     //empty line
  i : dint;
end_var;

//Eerst checken of het vorige file al verstuurt is
    error :=0; // 1
    File.Handle := FileIO.FileOpen(path:="c:\\data\\Petri.txt", create:=0, Attribs:=0); // 2
    File.Offset :=0; // 3
    filebestaatal := 0; // 4
    if File.Handle > 0 then // 5, file bestaat al
      writeState := -1;
      FileIO.FileClose(File.Handle);
      filebestaatal := 1;
    else
      FileIO.FileClose(File.Handle); // 6
      writeState := 1;     // 7, Busy
      _memset(dest := #eLine,usByte :=0,cntr:= sizeof(RFLine)); // 8, clear eLine, lege regel

//File nu openen om te schrijven

      Datawritten := false;  // 9
      _memset(dest := #wstring,usByte :=32,cntr:= sizeof(wstring));  // 10, Fill line with spaces
      for iii := 0 to (MaxLines-1) do  // 11
        if ptr^[iii].Status = _stacked then // 12
          _memset(dest := #wstring,usByte :=32,cntr:= sizeof(wstring));  // 13, Fill line with spaces

          RFMain.GetDataAt(p_us_data:=#wLine$usint, ud_size:=sizeof(Line), ud_at:=iii * sizeof(Line)); // 14
//Clear data in DataBase
          RFMain.SetDataAt(p_us_data:=#eLine$usint, ud_size:=sizeof(Line), ud_at:=iii * sizeof(Line)); // 15

//Ook in de ram database wissen:
          ptr^[iii].status := _empty; // 16
          ptr^[iii].AnalyseNr :=0; // 17
          ptr^[iii].Verdunning :=0; // 18
          ptr^[iii].SampleType :=0; // 19

          _memcpy(ptr1:=#wstring[0], ptr2:=#wline.BCField1bc[0], cntr:=24); // 20
          _memcpy(ptr1:=#wstring[24], ptr2:=#wline.BCField2[0], cntr:=6); // 21
          _memcpy(ptr1:=#wstring[30], ptr2:=#wline.BCField3[0], cntr:=6); // 22
          _memcpy(ptr1:=#wstring[36], ptr2:=#wline.BCField4[0], cntr:=6); // 23
          _memcpy(ptr1:=#wstring[42], ptr2:=#wline.BCField5date[0], cntr:=19); // 24

          _memcpy(ptr1:=#wstring[61], ptr2:=#wline.Medium[0], cntr:=6); // 25
```

Figuur 67, functie Write_FIle

In deze de wordt de betreffende bestand weer geopend. De offset wordt weer gedefinieerd met waarde 0.
Eerst zal er gekeken worden welke regels terug geschreven mogen worden.
Bij het ophalen van het bestand wordt de status van een geschreven regel op **_fileread** gezet.
Wanneer een barcode gescand wordt zal de betreffende regel waarde barcode gelokaliseerd is, status **_scanned** krijgen waarna de procesgegevens naar de betreffende Kitty gestuurd wordt.
Daarna wordt de status **_Filled** wanneer een gescande schaaltje gevuld is en status **_stacked** als het schaaltje opgestapeld is.
Bij stap 12 uit figuur 67 wordt per regel gekeken of deze de status **_stacked** bevat. Wanneer dit het geval is wordt de betreffende regel uit de RAM geheugen gehaald met een *GetDataAt* en in de variabele **wLine** gezet. Deze variabele is hetzelfde opgebouwd als de variabele **Line** bij het lezen van het bestand (zie figuur 64).
Daarna wordt de betreffende regel met niets gevuld in de RAM geheugen. Dit wordt gedaan door de variabele **eLine** die gevuld is met niets. Deze variabele is ook hetzelfde opgebouwd als de variabele **Line**.
Daarna wordt de status **_empty** gemaakt zodat de betreffende regel in de RAM met nieuwe data gevuld kan worden wanneer een nieuwe bestand opgehaald wordt.
Alle data uit de variabele **wLine** wordt in de string **wstring** geplaatst waarbij elke parameter uit de variabel een verschillende array waarde krijgt. De array waarde is afhankelijk van de opbouw van de regel in het tekst bestand.

```
842             _memcpy(ptr1:=#wstring[67], ptr2:=#wline.Werknemer[0], cntr:=3); // 26
843             _memcpy(ptr1:=#wstring[70], ptr2:=#wline.ExtraDat[0], cntr:=129); // 27
844
845             for i := 0 to 197  do // 28
846               if wstring[i] = 0 then
847                 wstring[i] := 32;
848               end_if;
849             end_for;
850
851             wstring[197] := 13; // 29, return
852             wstring[198] := 10; // 30, linefeed
853             wstring[199] := 0; // 31
854             if Datawritten = false then
855               File.Handle := FileIO.FileOpen(path:="c:\\data\\Petri.txt", create:=1, Attribs:=0); // 32
856               File.Offset :=0; // 33
857               Datawritten := true; // 34
858             end_if;
859             NrWritten := FileIO.FileWrite(handle:=File.Handle, data:=#wstring, size:=199); // 35
860
861           ////einde nieuwe opbouw
862
863         end_if;
864       end_for;
865
866       FileIO.FileClose(File.Handle); // 36
867       writeState := 2;    // 37, ready
868
869     end_if;    //File al weg?
870
871 END_FUNCTION
```

Figuur 68, vervolg functie Write_File

Wanneer de gehele string gevuld is met data zal deze met de functie *FileWrite* terug geschreven worden naar het bestand.

In de software van de server zal weer eerste een FTP verbinding opgebouwd moeten worden. Dit gebeurd op dezelfde als bij het lezen van het bestand.

```
12:
  result := C_FTP.FTP_TRANSFTYPE(0,1);  // 1
  C_FTP.FTP_DELE(pFile:="/kitty2/program/Petri.old"); // 2

  Status := C_FTP.FTP_REN(pFrom:="/kitty2/program/Petri.log", pTo:="/kitty2/program/Petri.old");   // 3
  File.Handle := FileIO.FileOpen(path:="c:\\data\\Petri.txt", create:=0, Attribs:=0);  //4
  File.Offset :=0; // 5
  File.Length := FileIO.FileLength(handle:=file.handle); // 6
  FileIO.FileClose(handle:=File.Handle); // 7

  udLength := to_udint(File.Length); // 8
  pBuf_File := "c:\\data\\Petri.txt";// 9
  pServerFile := "/Petri.log";  // 10
  result := C_FTP.FTP_Put(pBuf_File, udLength, pServerFile); // 11
  if result > 0 then // 12
    result := FileIO.FileDelete(file:="c:\\data\\Petri.txt");
    if result < 0 then // 13
      status :=-1;
    else
      status :=1;
    end_if;
  else
    status := -1;
  end_if;
  C_FTP.FTP_CLOSE(); // 14
  ssw :=0;
```

Figuur 69, aanroep voor het terug schrijven van het bestand

In stap 1 van figuur 69 is te zien dat de transmissie bepaald moet worden met de functie *FTP_TRANSFTYPE*.
Daarna word in stap 4 de betreffende bestand geopend. Dit wordt gedaan om de lengte van het bestand te bepalen. De lengte wordt vervolgens opgeslagen in de variabele **File.Length** waarna het bestand weer gesloten wordt.
Daarna wordt de functie *FTP_PUT* aangeroepen. Hierbij wordt de lengte van het bestand, het pad naar de interne harde schijf en het pad op het netwerk van de klant meegegeven.
Wanneer het bestand succesvol verstuurd is zal het bestand met de functie *FileDelete* verwijderd worden van de interne harde schijf.
Aan het einde wordt de FTP verbinding weer gesloten.

3. Resultaten

3.1 Resultaten opdracht Functie blok library (Unitronics)

Voor deze opdracht zijn er goede resultaten neer gezet maar viel het wel tegen in vergelijking met de verwachte resultaten uit het Plan van Aanpak.
Het aantal gebruikte componenten bleek veel minder te zijn dan gedacht. Daarnaast bleek de opgebouwde software veelvoudig terug te komen bij meerdere componenten.
Complete processen konden niet gemaakt worden omdat meerder functieblokken in één niet volledig geïmporteerd konden worden.
De structuur van de blokken moest ook aangepast worden. Verwacht werd dat de programmeur vanaf de buiten kant van de blokken zijn in en outputs kon definiëren maar dit moest inwendig gedaan worden omdat de functieblokken niet gemonitord kon worden wanneer lokale scopes gebruikt zou worden.
Ook blijkt de Unilogic software, die nog maar net vrij gegeven was, niet naar behoren te functioneren. Verwacht was dat de software compleet was en goed functioneerde.

3.2 Resultaten opdracht Dataoverdracht

De resultaten voor deze opdracht waren vele malen beter dan verwacht.
Dit komt omdat pas aan de opdracht gewerkt werd in de laatste 7 weken van de afstudeerstage.
De eerste verwachting was het concept van FTP te verwerken in een Siemens PLC. Dit ging niet door omdat een module van €1730,- aangeschaft zou moeten worden.
In plaats van een Siemens PLC is er gebruik gemaakt van een Sigmatek PLC. Dit was een uitdaging omdat de applicatie software Lasal object georiënteerd geprogrammeerd wordt. Binnen mijn studie was het object georiënteerd programmeren het enige vak waar ik veel moeite mee had.
Tijdens het uitwerken van de opdracht bleek dit echter geen enkel probleem te zijn.
De opgebouwde software is door mij geanalyseerd en uitgezocht. Toen de werking voor mij duidelijk was, bleek ik instaat deze kennis over te dragen aan de collega programmeurs in een overleg sessie van twee dagen.

4. Conclusies

De conclusie omtrent de functieblok opdracht is dat de betreffende software nog niet geschikt is om een complete library op te bouwen. In deze software zitten nog teveel ‘kinderziektes’. Daarnaast zijn de mogelijkheden met de lokale scopes zeer beperkt.

De conclusie omtrent de dataoverdracht opdracht is dat voor Siemens PLC’s het wel mogelijk is maar erg duur. Dit komt omdat er minimaal 2 extra modules aangeschaft moeten worden naast de standaard S7 300 PLC.
Het concept omtrent data overdracht is voor een Sigmatek PLC uiterst geschikt. Daar en tegen wanneer basis principe uitgebreid moet worden zal de programmeur veel ervaring met Lasal moeten hebben bij het uitbreiden van de software.

5. Aanbevelingen

Voor de Unilogic software zijn de volgende aanbevelingen:

- Het gebruik van lokale scopes is te beperkt. Het zou beter zijn als de lokale scopes het zelfde functioneren als de globale. Hierbij mag de enige beperking zijn dat de lokale scopes gebruikt mogen worden binnen de betreffende functie.
- Voor het importeren van een functie zou het beter zijn dat, wanneer er meerde functies geïmplementeerd zijn binnen één functie, deze mee geïmporteerd worden. Op deze manier bestaat de mogelijkheid om gehele processen te definiëren binnen één functieblok.

Voor de Lasal software zijn de volgende aanbevelingen:

- Er is op internet is er geen informatie te vinden omtrent de functies binnen de applicatie software. Het zou voor een beginnend programmeur makkelijke zijn als er een document was waar de werking van allen verschillende functie, simpel wordt uitgelegd.

6. Literatuur

Literatuur opdracht functieblok:

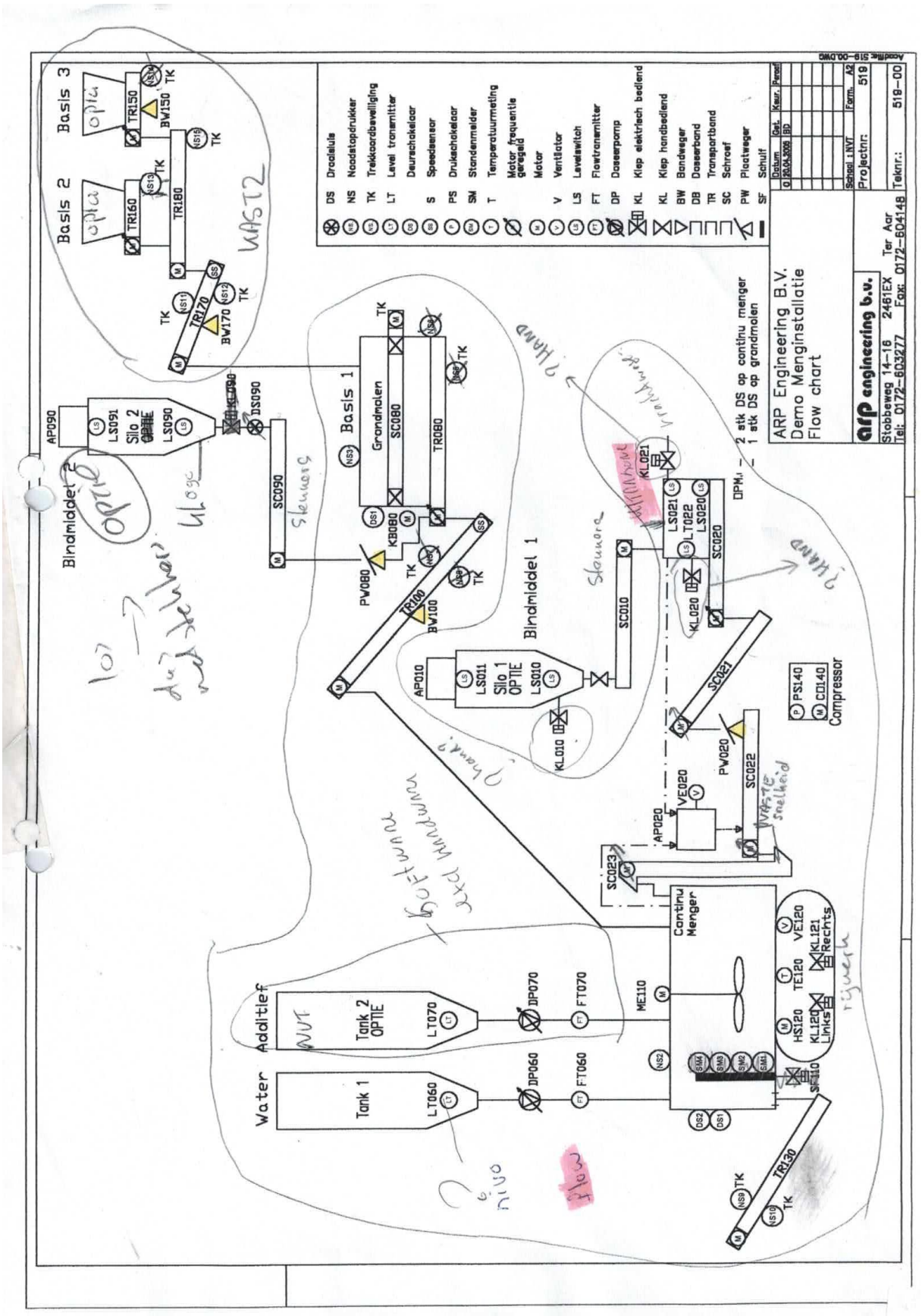
- DS8025-Transmitter-EU-EN.pdf
- 210_flow_sensor.pdf
- 235_Flow_sensor.pdf
- 00813-0100-4750.pdf
- DS8071-Standard-EU-EN.pdf
- en_datasheet_hm.pdf
- FLOW080L.pdf
- t40.6010en.pdf
- t40.6020en.pdf
- io-pt400-4k_spec_12-06.pdf
- IO_COMMON_INSTAL-GUIDE_07-13.pdf
- UID-0808THS_TECH-SPEC_08-13.pdf
- UIA-0402N_TECH-SPEC_01-14.pdf

Literatuur opdracht dataoverdracht:

- C-IPC_ATOM_Englisch.pdf
- LASAL OS.pdf
- Training Manual Lasal Class 2 ver 9.0.pdf
- 21605954_FTP_ITCP_DOKU_V11_e.pdf
- s7300_cpu314c_2_cpu315_2_cpu317_2_cpu319_3_pndp_en-US_en-US.pdf
- 81367009_FTP_S7-1200_DOKU_V1_0_en.pdf
- CP_IT_FTP_auto_e.pdf
- mn_it-cp_76.pdf
- mn_it-cp-infoprgm_76.pdf
- MN_s7-cps-ie_76.pdf
- PGH_FC-FB-S7CP_76.pdf

7. Bijlage

Bijlage A



Bijlage B

Menginstallatie

Uitgaand

- Een mengsel

Ingaand

- Water
- Additief
- Bindmiddel 1
- Bindmiddel 2
- Basis 2
- Basis 3

Mengsel -> Continue menger (1)

- Transportband 130
    - *Motor*
    - *Trekkoord 9*
    - *Trekkoord 10*
- Continue menger
    - *Deurschakelaar 1*
    - *Deurschakelaar 2*
    - *Noodstop 2*
    - *Motor ME110*
    - ***Rijwerk***
    - ***Filter 20****

Rijwerk

- *Motor HS120*
- *Temperatuurmeting 120*
- *Ventilator 120*
- *Elektronische klep 120*
- *Elektronische klep 121*

Mengsel -> Continue menger -> Water (2)

- Flow transmitter 60
- Doseerpomp 60
- Tank 1
    - *Leveltransmitter 60*

Mengsel -> Continue menger -> Additief (3)

- Flow transmitter 70
- Doseerpomp 70
- Tank 2
    - *Leveltransmitter 70*

Mengsel -> Continue menger -> Bindmiddel 1 (4)

- Schroef 23
    - *Motor frequentie geregeld*

- **Schroef 22**
    - *Motor frequentie geregeld*
    - ***Filter 20***
- Plaatweger 20
- Schroef 21
    - *Motor frequentie geregeld*
- Schroef 20
    - *Motor frequentie geregeld*
- Bunker
    - *Level switch 20*
    - *Level switch 21*
    - *Level transmitter 22*
    - *Elektronische klep 20*
    - *Elektronische klep 21*
    - ***Filter 20****
- Schroef 10
    - *Motor*
- Silo 1
    - *Elektronische klep 10*
    - *Levelswitch 10*
    - *Levelswitch 11*
    - *Filter 10*

***Schroef 22 -> Filter 20 (5)**

- Ventilator 20

Mengsel -> Continue menger -> Bindmiddel 2 (6)

- Transportband 100
    - *Motor*
    - *Speedsensor*
- Transportband 80
    - *Motor frequentie geregeld*
- Plaatweger 80
- Schroef 90
    - *Motor*
- Draaisluis 90
- Silo 2
    - *Elektronische klep*
    - *Levelswitch 90*
    - *Levelswitch 91*
    - *Filter 90*

Mengsel -> Continue menger -> Basis 1 (7)

- Transportband 100
    - *Motor*
    - *Speedsensor*
- Transportband 80
    - *Motor frequentie geregeld*
- Grondmolen
    - *Deurschakelaar 1*
    - *Noodstop 3*
- Schroef 80
    - *Motor*
- Transportband 170

    - *Motor*
    - *Trekkoord 11*
    - *Trekkoord 12*
    - *Speedsensor*
- Bandweger 170
- Transportband 180
    - *Motor*
    - *Trekkoord 15*

Mengsel -> Continue menger -> Basis 1 -> Basis 2 (8)

- Transportband 160
    - *Motor frequentie geregeld*
    - *Trekkoord 13*

Mengsel -> Continue menger -> Basis 1 -> Basis 3 (9)

- Bandweger 150
- Transportband 150
    - *Motor frequentie geregeld*

Continue menger

Input:

(1) Trekkoord 9
(1) Trekkoord 10
(1) Deurschakelaar 1
(1) Deurschakelaar 2
(1) Noodstop 2
(2) Flowtransmitter 60
(3) Flowtransmitter 70
(6) Plaatweger 80
(7) Bandweger 170
(9) Bandweger 150
(5) Plaatweger 20

Output:

(1) Motor transportband 130
(2) Doseerpomp 60
(3) Doseerpomp 70
(6) Motor transportband 100
(4) Motor freq. Schroef 23
(1) Motor menger ME110

Rijwerk

Input:

(1) Motor HS120
(1) Temperatuurmeting 120

Output:

(1) Elektronische klep 120
(1) Elektronische klep 121
(1) Ventilator 120

Water

Input:

(2) Level transmitter 60
(2) Doseerpomp 60

Output:

(2) Flowtransmitter 60

Additief

Input:

(3) Level transmitter 70
(3) Doseerpomp 70

Output:

(3) Flowtransmitter 70

Bindmiddel 1

Input:

(4) Motor freq. schroef 23
(4) Plaatweger 20
(4) Level switch 20
(4) Level switch 21
(4) Level transmitter
(4) Level switch 10
(4) Level switch 11

Output:

(4) Motor freq. schroef 22
(4) Motor freq. schroef 21
(4) Motor freq. schroef 20
(4) Elektronische klep 20
(4) Elektronische klep 21
(4) Motor schroef 10
(4) Elektronische klep 10

filter 20

Input:

(1) Motor ME110
(5) Elektronische klep 21

Output:

(6) Ventilator 20

Bindmiddel 2

Input:

(6) Motor transportband 100
(6) Bandweger 100
(6) Speedsensor
(6) Plaatweger 80
(6) Level switch 90
(6) Level switch 91

Output:

(6) Motor schroef 90
(6) Draaisluis 90
(6) Elektronische klep 90

Basis 1

Input:

(6) Motor transportband 100
(7) Deurschakelaar 1
(7) Noodstop 3
(7) Bandweger 170
(7) Speedsensor
(7) Trekkoord 11
(7) Trekkoord 12
(7) Trekkoord 15

Output:

(7) Motor freq. transportband 80
(7) Motor schroef 80
(7) Motor Transportband 170
(7) Motor Transportband 180

Basis 2

Input:

(7) Motor transportband 180
(8) Trekkoord 13

Output:

(8) Motor freq. transportband 160

Basis 3

Input:

(7) Motor transportband 180
(9) Bandweger 150

Output:

(8) Motor transportband 150

Bijlage C

Signalering storing

Input:

Tijd signalering inactief. *Integer.* (FB SS_Timer 1.Preset)
Tijd signalering actief. *Integer.* (FB SS_Timer 2.Preset)
Tijd her activatie signalering. *Integer,* (FB)
Reset signalering. *Boolean, NO contact.* (FB SS_Reset signalering)

Output:

Signalering. *Boolean, NO contact.* (FB SS_Signalering uit)

Noodstop/trekkoord

Input:

Digitale input voor contact gemaakt. *Boolean, NO contact.* (FB NS_Noodstop in)
Reset melding. *Boolean, NO contact.* (FB NS_Reset).

Output:

Noodstop actief. *Boolean NO coil.* (FB NS_Noodstop actief)

Flowtransmitter algemeen

Input:

*Puls. *Boolean, NO contact.*
*Analoog signaal. *Integer.*
*High speed counter. *Integer.*
Flow actief. *Boolean, NO contact.*
Pomp actief. *Boolean, NO contact.*
Reset cumulatieve waarde. *Boolean, NO contact.*
Reset alarm. *Boolean, NO contact.*

Output:

Algemene storing. *Boolean, NO coil.*
Liter/minuut. *Real.*
Liter/uur. *Real.*
M^3/uur. *Real.*
Cumulatief aantal liters. *Real.*
Totaal aantal liters. *Real.*
Cumulatief aantal M^3. *Real.*
Totaal aantal M^3. *Real.*
Melding geen flow.

Flowtransmitter puls

Input:

Puls. *Boolean, NO contact.* (FB FL PU_Puls in)
Flow actief. *Boolean, NO contact.* (FB FL PU_Flow actief)
Pomp actief. *Boolean, NO contact.* (FB FL PU_Pomp actief)
Reset cumulatieve waarde. *Boolean, NO contact.* (FB FL PU_Reset cumulatief)
Reset alarm. *Boolean, NO contact.* (FB FL PU_Reset alarm)
Liter per puls. *Real.* (FB FL PU_Ltr/puls)
Aantal decimalen scherm. *Integer.* (FB FL PU_Aantal decimalen)
Tijd alarm. *Real.* (FB FL PU_Tijd alarm)

Factor total waarde. *Real*, (FB FL PU_Factor total waardes)

Output:

Algemene storing. *Boolean, NO coil.* (FB SS_Algemene storing)
Liter/minuut. *Integer.* (FB FL PU_Ltr/min uit)
Liter/uur. *Integer.* (FB FL PU_Ltr/uur uit)
M^3/uur. *Integer.* (FB FL PU_M3/uur)
Cumulatief aantal liters. *Integer.* (FB FL PU_Liter (R) uit)
Totaal aantal liters. *Integer.* (FB FL PU_Liter uit)
Cumulatief aantal M^3. *Integer.* (FB FL PU_M3 (R) uit)
Totaal aantal M^3. *Integer.* (FB FL PU_M3 uit)
Melding geen flow. *Boolean, NO coil.* (FB FL PU_Alarm)

Flowtransmitter analoog

Input:

Max digitaal. *Real.* (FB FL AN_Max digitaal)
Min digitaal. *Real.* (FB FL AN_Min digitaal)
Max liter per minuut. *Real.* (FB FL AN_Max ltr/min)
Min liter per minuut. *Real.* (FB FL AN_Min ltr/min)
Digitaal in. *Integer.* (FB FL AN_Digitaal in)
Aantal decimalen scherm. *Integer.* (FB FL AN_Aantal decimalen)
Tijd omrekening. *Integer.* (FB FL AN_Timer meting.Preset)
Tijd timer alarm. *Integer.* (FB FL AN_Timer alarm.Preset)
Flow actief. *Boolean, NO contact.* (FB FL AN_Flow actief)
Pomp actief. *Boolean, NO contact.* (FB FL AN_Pomp actief)
Reset cumulatieve waarde. *Boolean, NO contact.* (FB FL AN_Reset cumulatief)
Reset alarm. *Boolean, NO contact.* (FB FL AN_Reset alarm)
Factor total waarde. *Real*, (FB FL AN_Factor total waardes)

Output:

Algemene storing. *Boolean, NO coil.* (FB SS_Algemene storing)
Liter/minuut. *Integer.* (FB FL AN_Ltr/min uit)
Liter/uur. *Integer.* (FB FL AN _Ltr/uur uit)
M^3/uur. *Integer.* (FB FL AN _M3/uur)
Cumulatief aantal liters. *Integer.* (FB FL AN _Liter (R) uit)
Totaal aantal liters. *Integer.* (FB FL AN _Liter uit)
Cumulatief aantal M^3. *Integer.* (FB FL AN _M3 (R) uit)
Totaal aantal M^3. *Integer.* (FB FL AN _M3 uit)
Melding geen flow. *Boolean, NO coil.* (FB FL AN _Alarm)

Flowtransmitter high speed counter

Input:

Max frequentie. *Real.* (FB FL FQ_Max frequentie)
Min frequentie. *Real.* (FB FL FQ_Min frequentie)
Max liter per minuut. *Real.* (FB FL FQ_Max ltr/min)
Min liter per minuut. *Real.* (FB FL FQ_Min ltr/min)
Frequentie in. *Integer.* (FB FL FQ_Frequentie in)
Aantal decimalen scherm. *Integer.* (FB FL FQ_Aantal decimalen)
Tijdsinterval high speed counter. *Integer.* (FB FL FQ_Tijdsinterval frequentie)
Tijd omrekening. *Integer.* (FB FL FQ_Timer meting.Preset)
Tijd timer alarm. *Integer.* (FB FL FQ_Timer alarm.Preset)
Flow actief. *Boolean, NO contact.* (FB FL AN_Flow actief)

Pomp actief. *Boolean, NO contact.* (FB FL AN_Pomp actief)
Reset cumulatieve waarde. *Boolean, NO contact.* (FB FL AN_Reset cumulatief)
Reset alarm. *Boolean, NO contact.* (FB FL AN_Reset alarm)
Factor total waarde. *Real*, (FB FL AN_Factor total waardes)

Output:

Algemene storing. *Boolean, NO coil.* (FB SS_Algemen storing)
Liter/minuut. *Integer.* (FB FL FQ_Ltr/min uit)
Liter/uur. *Integer.* (FB FL FQ _Ltr/uur uit)
M^3/uur. *Integer.* (FB FL FQ _M3/uur)
Cumulatief aantal liters. *Integer.* (FB FL FQ _Liter (R) uit)
Totaal aantal liters. *Integer.* (FB FL FQ _Liter uit)
Cumulatief aantal M^3. *Integer.* (FB FL FQ _M3 (R) uit)
Totaal aantal M^3. *Integer.* (FB FL FQ _M3 uit)
Melding geen flow. *Boolean, NO coil.* (FB FL FQ _Alarm)

Plaatweger algemeen

Input:

*Puls. *Bitje, NO contact.*
*Analoog signaal. *Integer.*
*High speed counter. *Integer.*
Weging aan/uit. *Bitje, NO contact.*
Transport aan/uit. *Bitje, NO contact.*
Reset cumulatieve waarde. *Bitje, NO contact.*
Reset alarm. *Bitje, NO contact.*

Output:

Alarm. *Bitje, NO coil.*
Gewicht huidig. *Real.*
Gewicht cumulatief. *Real.*
Gewicht totaal. *Real.*
Algemene storing.

Plaatweger puls

Input:

Puls. *Bitje, NO contact.* (FB PW PU_Puls in)
Weging aan/uit. *Bitje, NO contact.* (FB PW PU_Weger actief)
Transport aan/uit. *Bitje, NO contact.* (FB PW PU_Transport actief)
Reset cumulatieve waarde. *Bitje, NO contact.* (FB PW PU_Reset cumulatief)
Reset alarm. *Bitje, NO contact.* (FB PW PU_Reset alarm)
Kilogram per puls. *Real.* (FB PW PU_Kg/puls)
Aantal decimalen. *Integer.* (FB PW PU_Aantal decimalen)
Tijd alarm. *Integer.* (FB PW PU_Timer alarm.Preset)
Factor total waarde. *Real*, (FB PW PU_Factor total waardes)

Output:

Alarm. *Bitje, NO coil.* (FB PW PU_Alarm)
Aantal kilogrammen. *Integer*. (FB PW PU_Kg uit)
Kilogrammen totaal. *Integer.* (FB PW PU_Kg uit)
Kilogrammen cumulatief. *Integer.* (FB PW PU_Kg (R) uit)
Ton totaal. *Integer.* (FB PW PU_Ton totaal uit)
Ton cumulatief. *Integer.* (FB PW PU_Ton (R) uit)

Kilogrammen/minuut. *Integer*. (FB PW PU_Kg/min uit)
Kilogrammen/uur. *Integer*. (FB PW PU_Kg/uur uit)
Tonnen/uur. *Integer*. (FB PW PU_Ton/uur uit)

Plaatweger analoog

Input:

Max digitaal. *Real.* (FB PW AN_Max digitaal)
Min digitaal. *Real.* (FB PW AN_Min digitaal)
Max gewicht. *Real.* (FB PW AN_Max kg)
Min gewicht. *Real.* (FB PW AN_Min kg)
Digitaal in. *Integer.* (FB PW AN_Digitaal in)
Puls. *Bitje, NO contact.* (FB PW AN_Puls in)
Weging aan/uit. *Bitje, NO contact.* (FB PW AN_Weger actief)
Transport aan/uit. *Bitje, NO contact.* (FB PW AN_Transport actief)
Reset cumulatieve waarde. *Bitje, NO contact.* (FB PW AN_Reset cumulatief)
Reset alarm. *Bitje, NO contact.* (FB PW AN_Reset alarm)
Aantal decimalen. *Integer.* (FB PW AN_Aantal decimalen)
Tijd alarm. *Integer.* (FB PW AN_Timer alarm.Preset)
Factor total waarde. *Real*, (FB PW AN_Factor total waardes)

Output:

Alarm. *Bitje, NO coil.* (FB PW AN_Alarm)
Aantal kilogrammen. *Integer*. (FB PW AN_Kg uit)
Kilogrammen totaal. *Integer.* (FB PW AN_Kg totaal uit)
Kilogrammen cumulatief. *Integer.* (FB PW An_Kg (R) uit)
Ton totaal. *Integer.* (FB PW AN_Ton totaal uit)
Ton cumulatief. *Integer.* (FB PW AN_Ton (R) uit)
Kilogrammen/minuut. *Integer*. (FB PW AN_Kg/min uit)
Kilogrammen/uur. *Integer*. (FB PW AN_Kg/uur uit)
Tonnen/uur. *Integer*. (FB PW AN_Ton/uur uit)

Plaatweger high speed counter

Input:

Max frequentie. *Real.* (FB PW FQ_Max frequentie)
Min frequentie. *Real.* (FB PW FQ_Min frequentie)
Max gewicht. *Real.* (FB PW FQ_Max kg)
Min gewicht. *Real.* (FB PW FQ_Min kg)
Frequentie in. *Integer.* (FB PW FQ_Frequentie in)
Aantal decimalen scherm. *Integer.* (FB PW FQ_Aantal decimalen)
Tijdsinterval high speed counter. *Integer.* (FB PW FQ_Tijdsinterval frequentie)
Tijd omrekening. *Integer.* (FB PW FQ_Timer meting.Preset)
Tijd timer alarm. *Integer.* (FB PW FQ_Timer alarm.Preset)
Weging aan/uit. *Bitje, NO contact.* (FB PW FQ_Weger actief)
Transport aan/uit. *Bitje, NO contact.* (FB PW FQ_Transport actief)
Reset cumulatieve waarde. *Boolean, NO contact.* (FB PW FQ_Reset cumulatief)
Reset alarm. *Boolean, NO contact.* (FB PW FQ_Reset alarm)
Factor total waarde. *Real*, (FB PW FQ_Factor total waardes)

Output:

Alarm. *Bitje, NO coil.* (FB PW FQ_Alarm)
Aantal kilogrammen. *Integer*. (FB PW FQ_Kg uit)
Kilogrammen totaal. *Integer.* (FB PW FQ_Kg totaal uit)
Kilogrammen cumulatief. *Integer.* (FB PW FQ_Kg (R) uit)

Ton totaal. *Integer.* (FB PW FQ_Ton totaal uit)
Ton cumulatief. *Integer.* (FB PW FQ_Ton (R) uit)
Kilogrammen/minuut. *Integer*. (FB PW FQ_Kg/min uit)
Kilogrammen/uur. *Integer*. (FB PW FQ_Kg/uur uit)
Tonnen/uur. *Integer*. (FB PW FQ_Ton/uur uit))

Motor

Input:

Motor start. *Bitje, NO contact,* (FB M_Motor actief)
Thermische relais. *Bitje, NC contact*. (FB M_Thermische beveilig)
Reset alarm. *Bitje NO contact*, (FB M_Reset)

Output:

Relais inschakelen. *Bitje, NO coil,* (FB M_Motor relais)
Alarm. *Bitje, NO coil.* (FB M_Alarm)

Motor frequentie geregeld

Input:

Max digitaal. *Real.* (FB MF_Max digitaal uit)
Min digitaal. *Real.* (FB MF_Min digitaal uit)
Maximale percentage snelheid. *Real*, (FB MF_Max percentage snelheid)
Minimale percentage snelheid. *Real*, (FB MF_Min percentage snelheid)
Start frequentie regelaar. *Bitje, NO contact.*
Vrijgave frequentie regelaar. *Bitje, NO contact.*
Reverse functie. *Bitje, NO contact.*
Snelheid motor. *Integer.*
Thermisch relais. *Btije, NC contact.*

Output:

Start signaal. *Bitje, NO coil.*
Vrijgave signaal. *Bitje, NO coil.*
Reverse. *Bitje, NO coil.*
Analoog signaal. *Real.*
Thermische storing. *Bitje, NO coil.*
Algemene storing. *Bitje, NO coil.*

Motor softstarter geregeld

Input:

Start softstarter. *Bitje, NO contact.*
Vrijgave softstarter. *Bitje, NO contact.*
Reverse functie. *Bitje, NO contact.*
Thermisch relais. *Btije, NC contact.*

Output:

Start signaal. *Bitje, NO coil.*
Vrijgave signaal. *Bitje, NO coil.*
Reverse. *Bitje, NO coil.*
Analoog signaal. *Integer.*
Thermische storing. *Bitje, NO coil.*
Algemene storing. *Bitje, NO coil.*

Temperatuurmeting

Input:

Analoog. *Integer.*
Reset alarm. *Bitje, NO contact.*

Output:

Alarm. *Bitje, NO coil.*
Temperatuur. *Integer*
Algemene alarm. *Bitje, NO coil.*

Elektrische klep

Input:

Klep aan/uit. *Bitje, NO contact.*
Relais klep. *Bitje, NO contact.*

Output:

Relais klep. *Bitje, NO coil.*

Level transmitter

Input:

Analoog. *Integer.*
Max waarde. *Integer*
Min waarde. *Integer.*

Output:

Totale hoeveelheid. *Integer.*
Max

Level switch

Input:

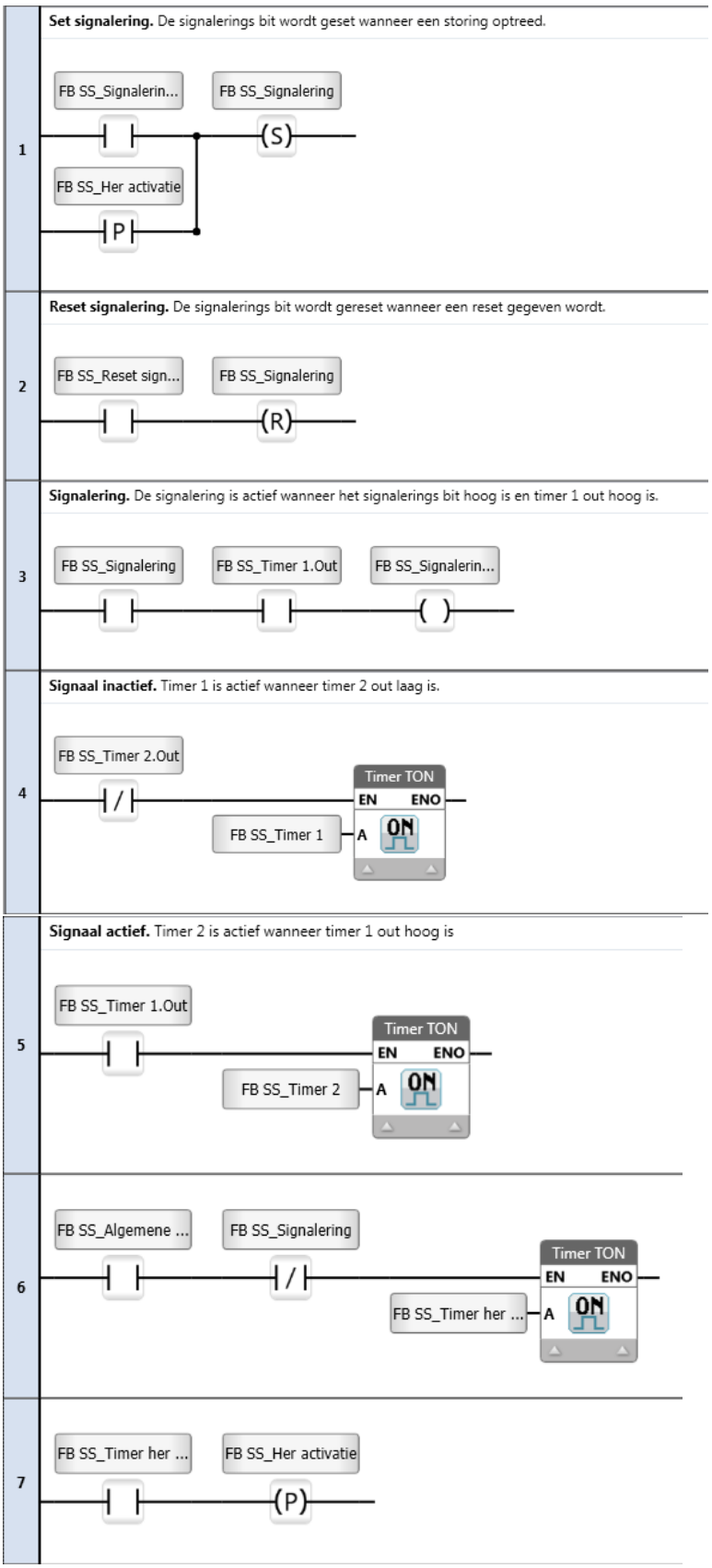
Switch aan/uit. *Bitje, NO contact.*
Setoff. *Bitje, NO contact.*

Output:

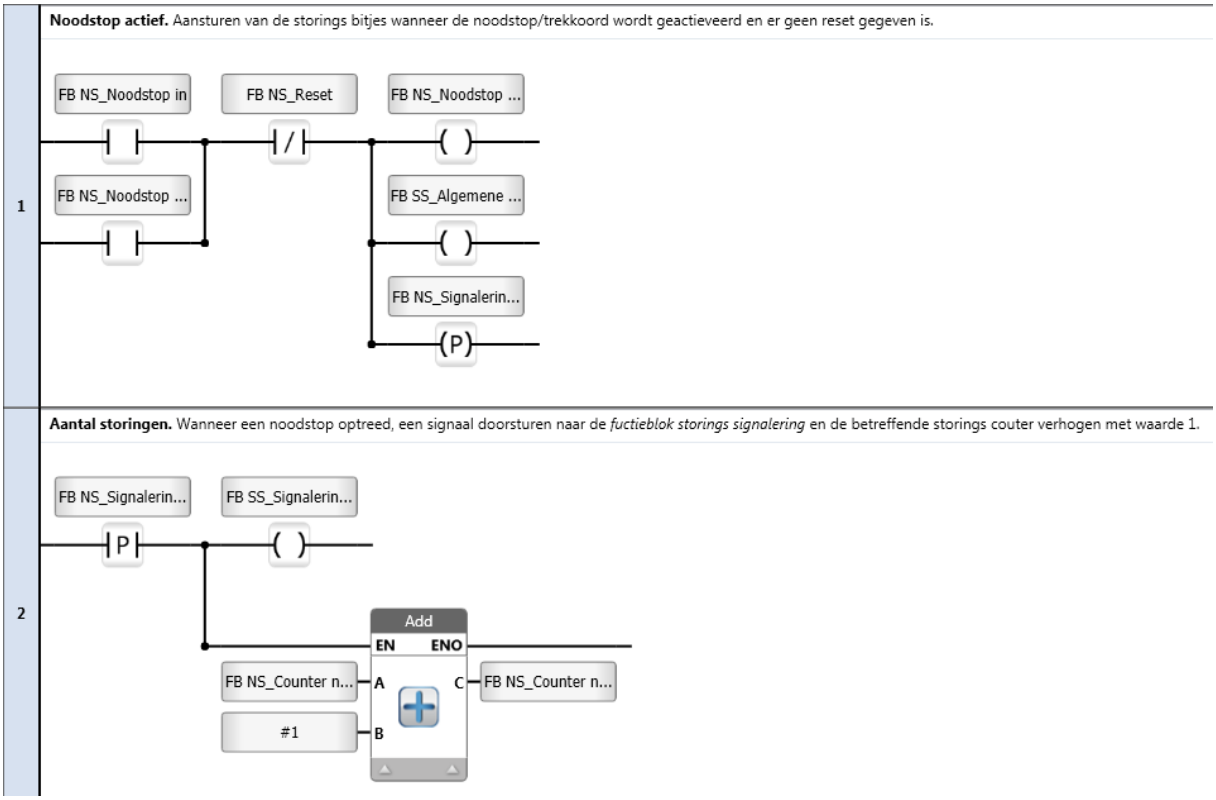
Switch aan/uit. *Bitje, NO coil.*

Bijlage D

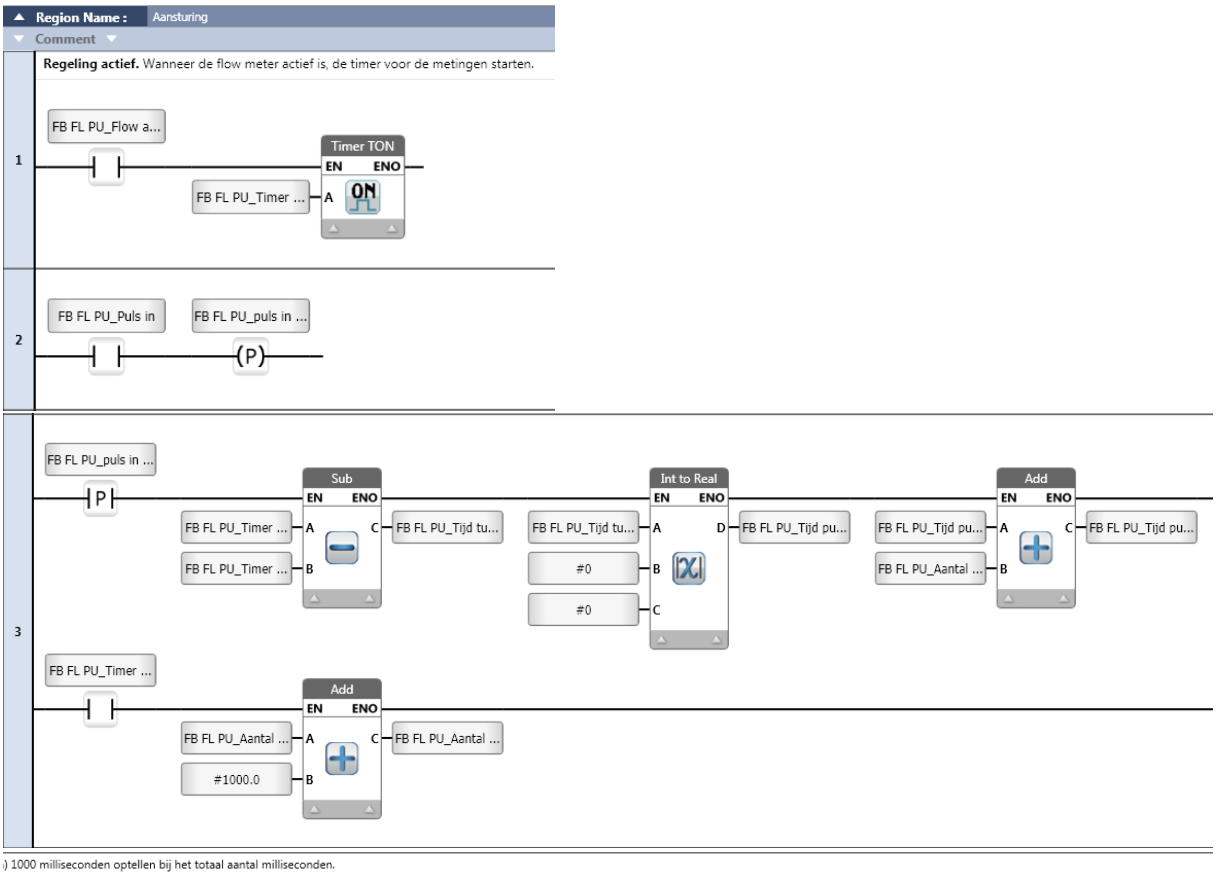
Functieblok Signalering storing



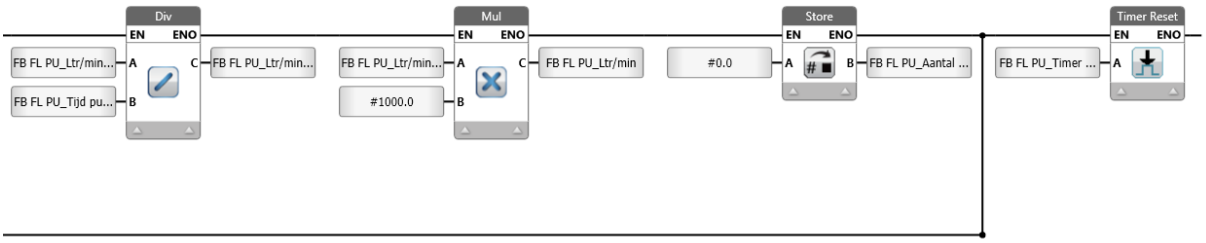
Functieblok Noodstop/trekkoord



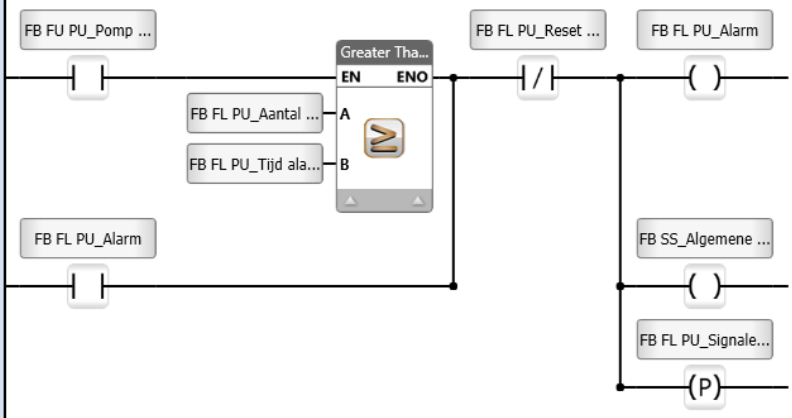
Functieblok Flowtransmitter puls



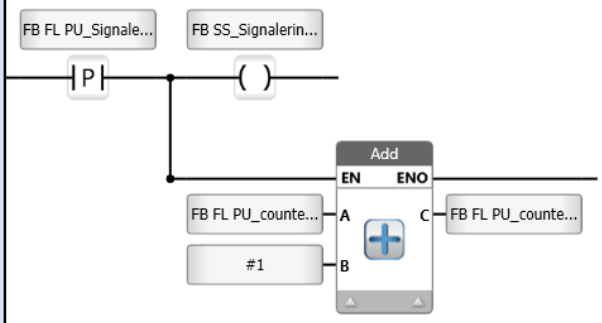
) 1000 milliseconden optellen bij het totaal aantal milliseconden.



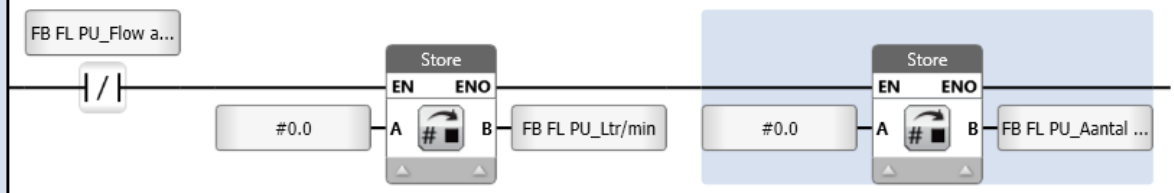
4 **Alarm.** Alarm activeren waneer de betreffende tijd dat een alarm gegeven moet worden is overschreden.



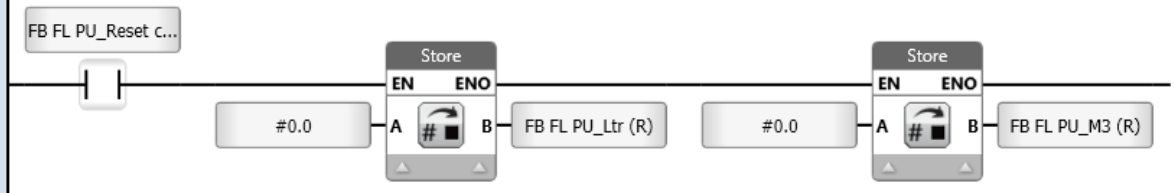
5 **Aantal storingen.** Wanneer een storing in de flow meter optreed, een signaal doorsturen naar de fuctieblok *storings signaler*

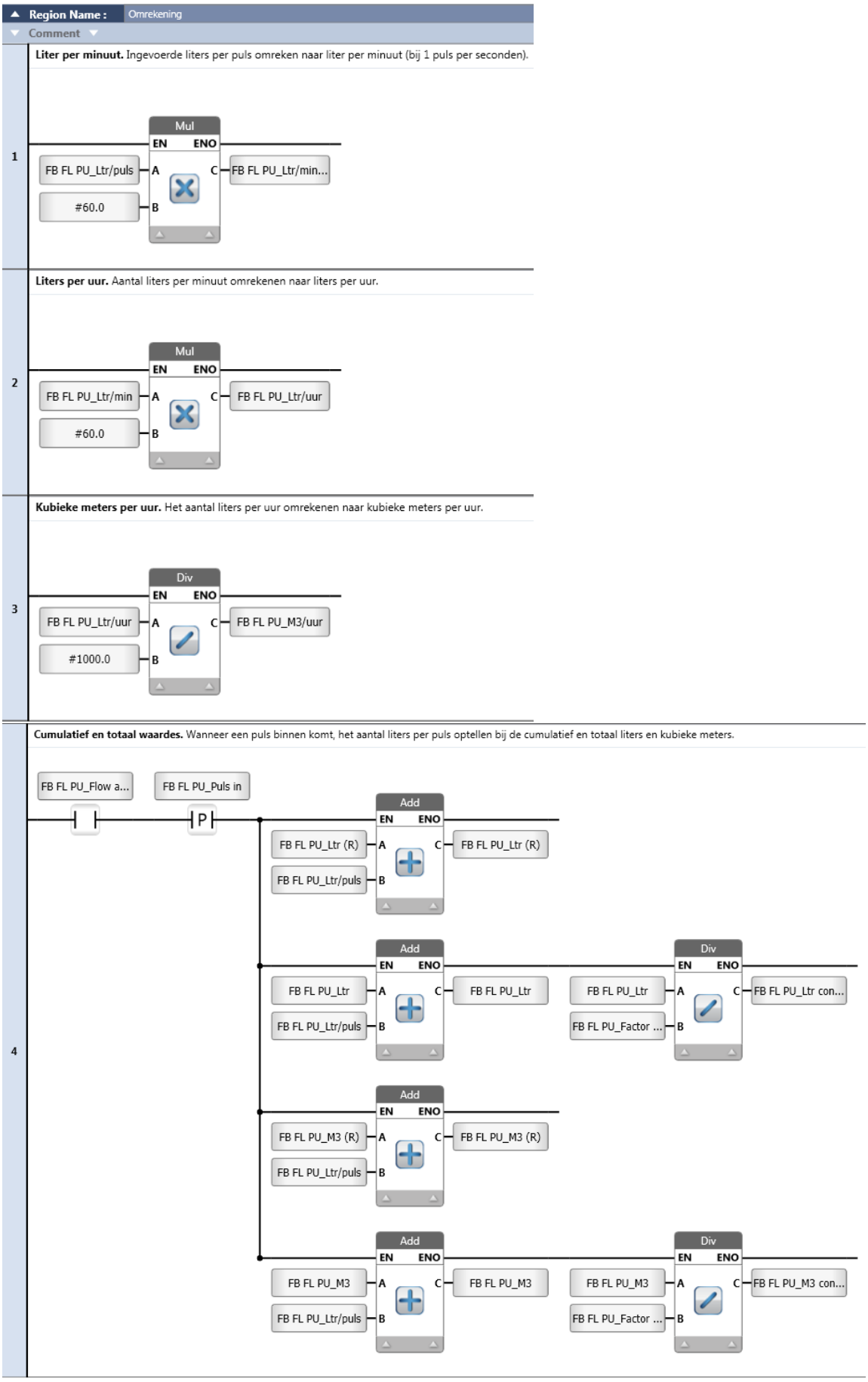


6 **Regeling inactief.** Waarde 0 aangeven wanneer er geen pulsen binnen komen.

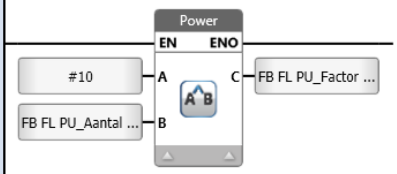


7 **Reset cumulatief.** Resetten van de cumulatieve liters en kubieke meters.

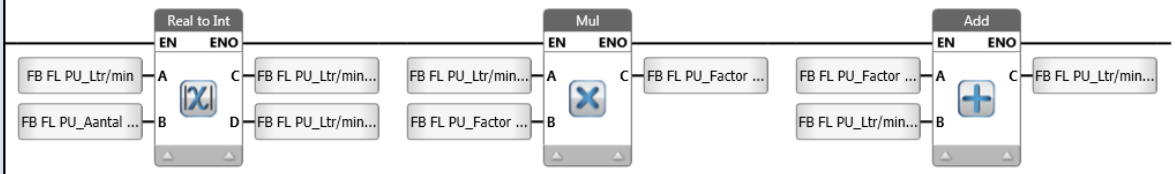



Region Name : Omrekening
Comment
1
Liter per minuut. Ingevoerde liters per puls omreken naar liter per minuut (bij 1 puls per seconden).
Mul
EN ENO
FB FL PU_Ltr/puls
A
C
FB FL PU_Ltr/min...
#60.0
B
2
Liters per uur. Aantal liters per minuut omrekenen naar liters per uur.
Mul
EN ENO
FB FL PU_Ltr/min
A
C
FB FL PU_Ltr/uur
#60.0
B
3
Kubieke meters per uur. Het aantal liters per uur omrekenen naar kubieke meters per uur.
Div
EN ENO
FB FL PU_Ltr/uur
A
C
FB FL PU_M3/uur
#1000.0
B
4
Cumulatief en totaal waardes. Wanneer een puls binnen komt, het aantal liters per puls optellen bij de cumulatief en totaal liters en kubieke meters.
FB FL PU_Flow a...
FB FL PU_Puls in
P
Add
EN ENO
FB FL PU_Ltr (R)
A
C
FB FL PU_Ltr (R)
FB FL PU_Ltr/puls
B
Add
EN ENO
FB FL PU_Ltr
A
C
FB FL PU_Ltr
FB FL PU_Ltr/puls
B
Div
EN ENO
FB FL PU_Ltr
A
C
FB FL PU_Ltr con...
FB FL PU_Factor ...
B
Add
EN ENO
FB FL PU_M3 (R)
A
C
FB FL PU_M3 (R)
FB FL PU_Ltr/puls
B
Add
EN ENO
FB FL PU_M3
A
C
FB FL PU_M3
FB FL PU_Ltr/puls
B
Div
EN ENO
FB FL PU_M3
A
C
FB FL PU_M3 con...
FB FL PU_Factor ...
B

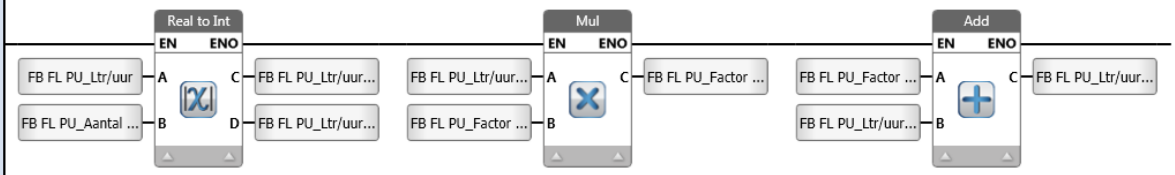
5 **Factor decimalen.** De vermenigvuldigings factor bepalen aan de hand van de gewenste decimalen achter de komma voor de uitgangs waardes



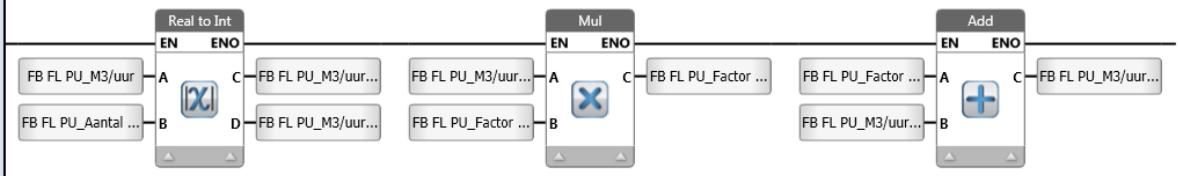
6 **Display waarde liters per minuut.** Real waarde van liters per minuut converteren naar integer waarde en omzetten naar de display waarde.



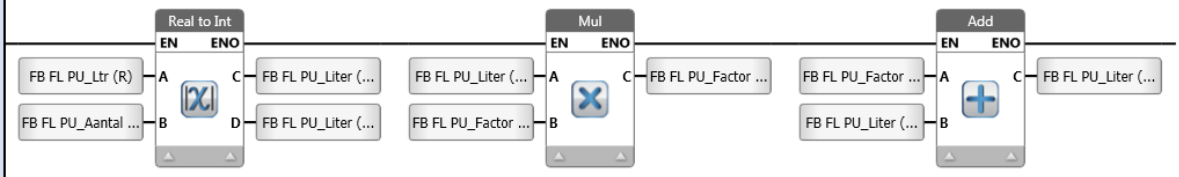
7 **Display waarde liters per uur.** Real waarde van liters per uur converteren naar integer waarde en omzetten naar de display waarde.



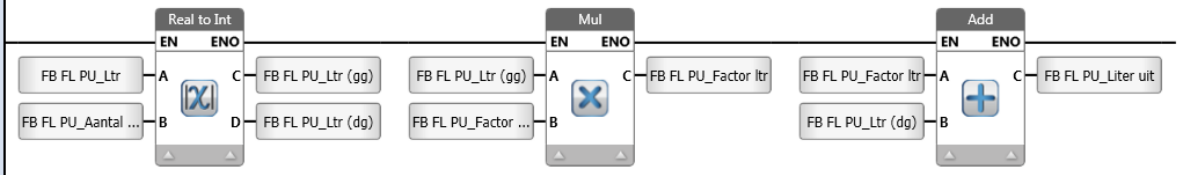
8 **Display waarde kubieke meters per uur.** Real waarde van kubieke meters per uur converteren naar integer waarde en omzetten naar de display waarde.

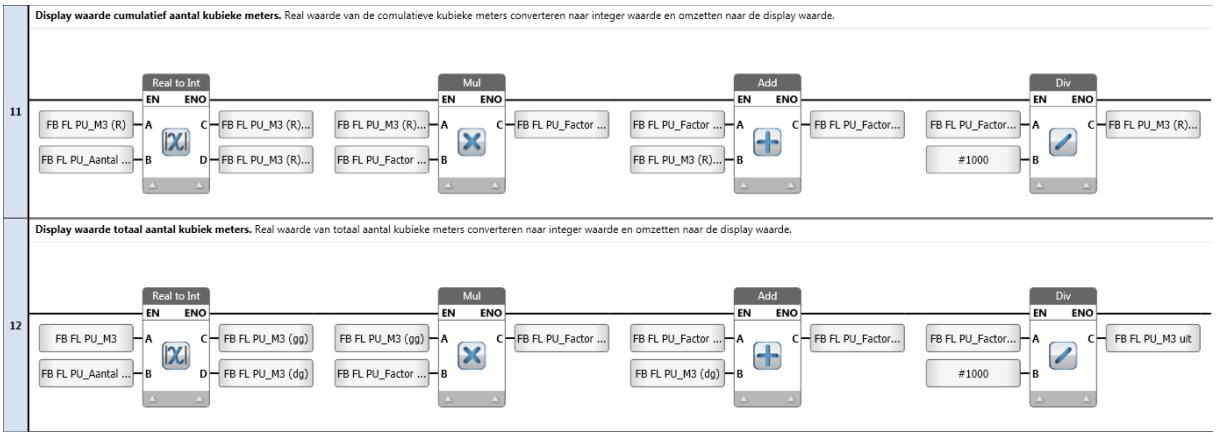


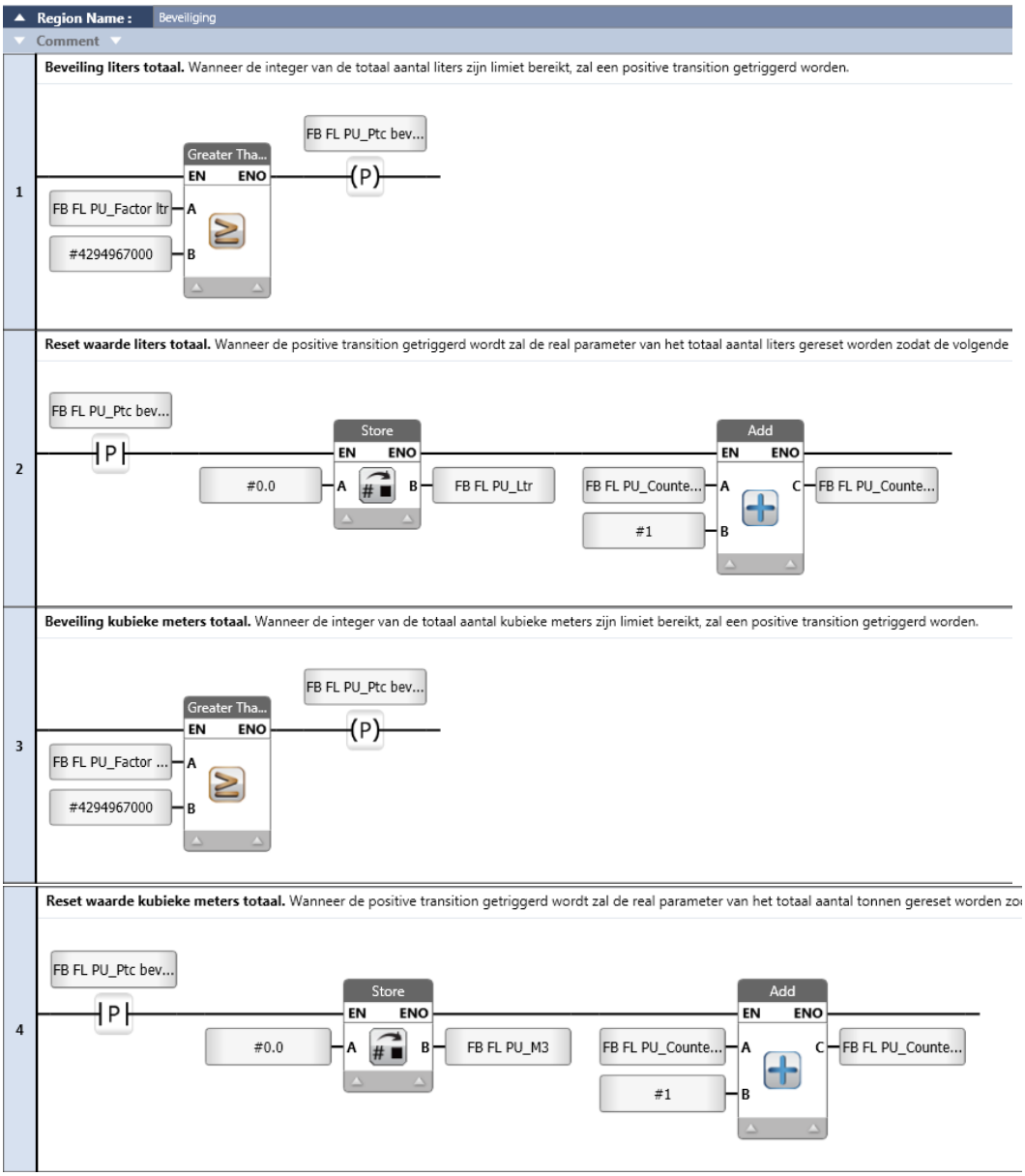
9 **Display waarde cumulatief aantal liters.** Real waarde van de cumulatieve liters converteren naar integer waarde en omzetten naar de display waarde.



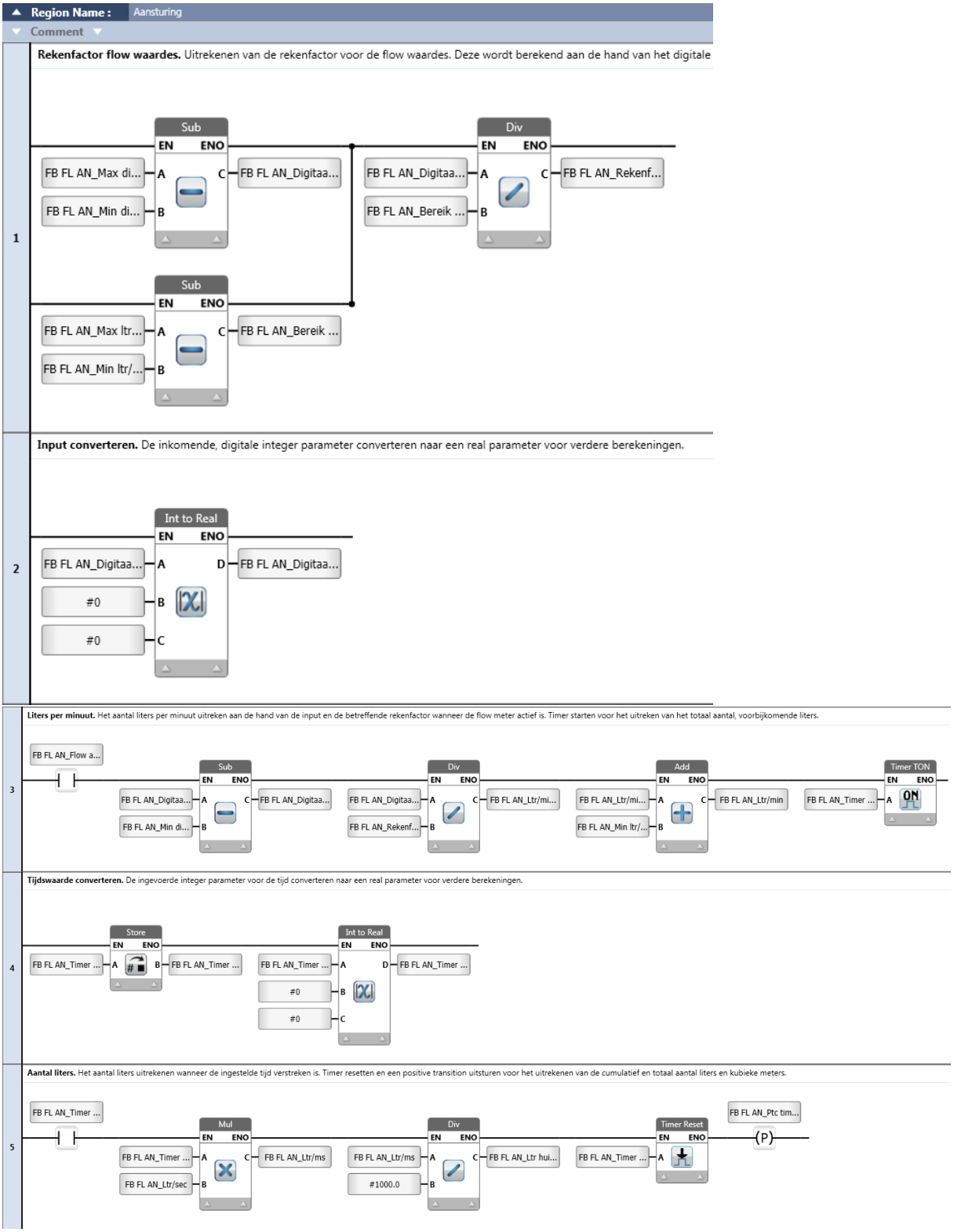
10 **Dispaly waarde totaal aantal liters.** Real waarde van de totaal aantal liters converteren naar integer waarde en omzetten naar de display waarde.



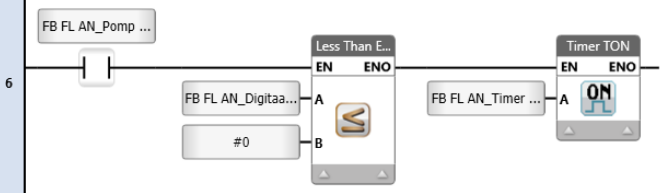

Display waarde cumulatief aantal kubieke meters. Real waarde van de comulatieve kubieke meters converteren naar integer waarde en omzetten naar de display waarde.
11
Real to Int
Mul
Add
Div
FB FL PU_M3 (R)
FB FL PU_Aantal ...
FB FL PU_M3 (R)...
FB FL PU_Factor ...
FB FL PU_Factor...
#1000
Display waarde totaal aantal kubiek meters. Real waarde van totaal aantal kubieke meters converteren naar integer waarde en omzetten naar de display waarde.
12
FB FL PU_M3
FB FL PU_Aantal ...
FB FL PU_M3 (gg)
FB FL PU_M3 (dg)
FB FL PU_M3 uit


Region Name :
Beveiliging
Comment
Beveiling liters totaal. Wanneer de integer van de totaal aantal liters zijn limiet bereikt, zal een positive transition getriggerd worden.
1
Greater Tha...
FB FL PU_Ptc bev...
FB FL PU_Factor ltr
#4294967000
Reset waarde liters totaal. Wanneer de positive transition getriggerd wordt zal de real parameter van het totaal aantal liters gereset worden zodat de volgende
2
Store
Add
#0.0
FB FL PU_Ltr
FB FL PU_Counte...
#1
Beveiling kubieke meters totaal. Wanneer de integer van de totaal aantal kubieke meters zijn limiet bereikt, zal een positive transition getriggerd worden.
3
FB FL PU_Factor ...
Reset waarde kubieke meters totaal. Wanneer de positive transition getriggerd wordt zal de real parameter van het totaal aantal tonnen gereset worden zo
4
FB FL PU_M3

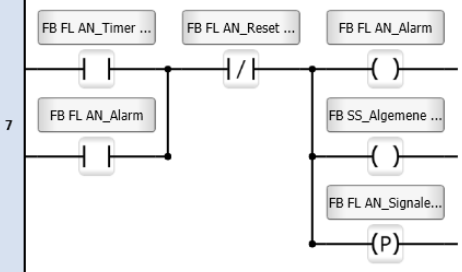
Functieblok Flowtransmitter analoog



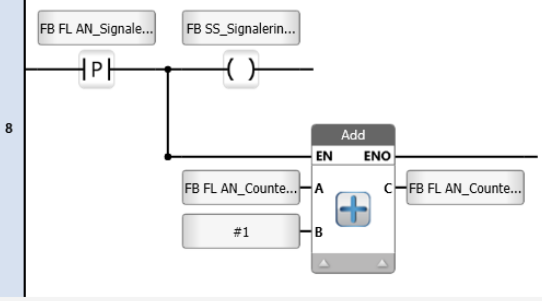
Timer alarm. Wanneer de pomp actief is en er is geen flow, timer alarm starten.



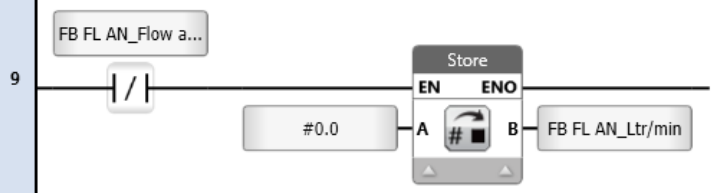
Alarm. Alarm triggeren wanneer de timer ouput "*timer alarm*" hoog is.



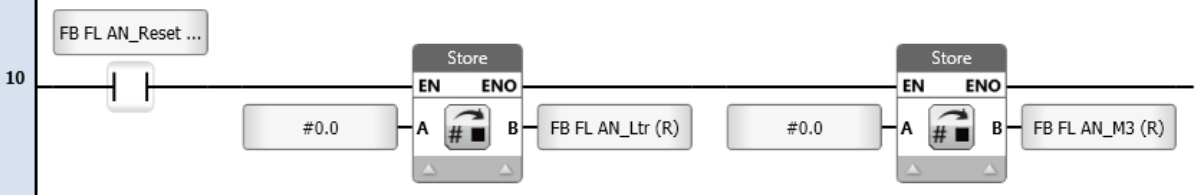
Aantal storingen. Wanneer een storing in de flow meter optreed, een signaal doorsturen naar de fuctieblok *storings signalering* en de betreffende storings couter verhogen met waarde 1.



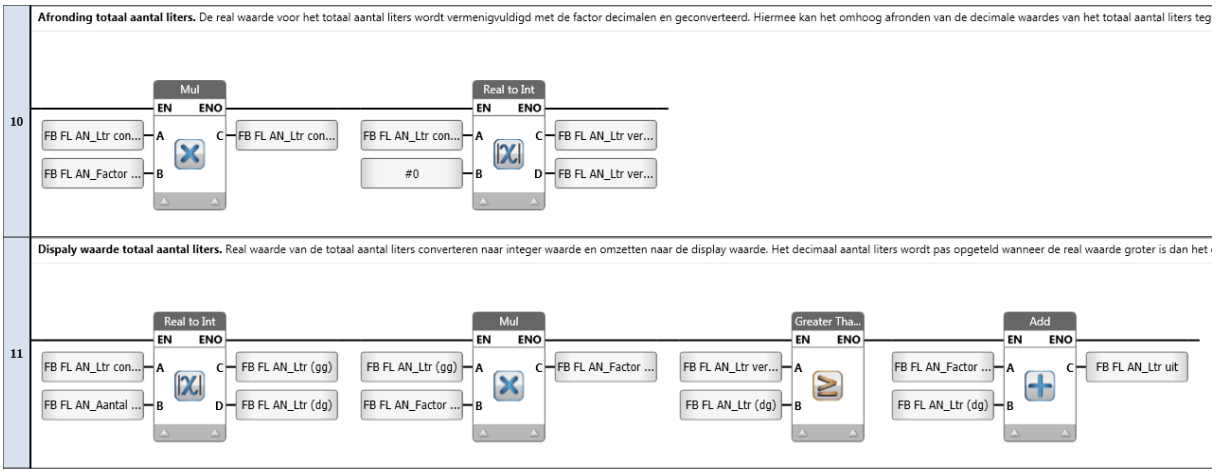
Regeling inactief. Waarde 0 weergeven op de display wanneer er geen flow binnen komt.



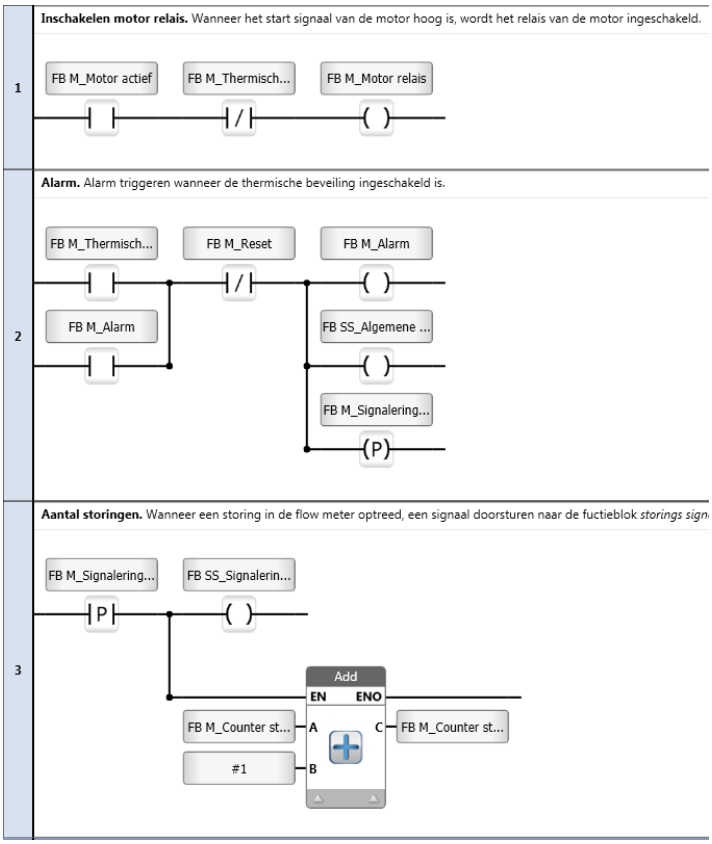
Reset cumulatieve waardes. Wanneer de reset knop ingedrukt is, de waarde 0 naar de cumulatief aantal liters en kubieke meters sturen.



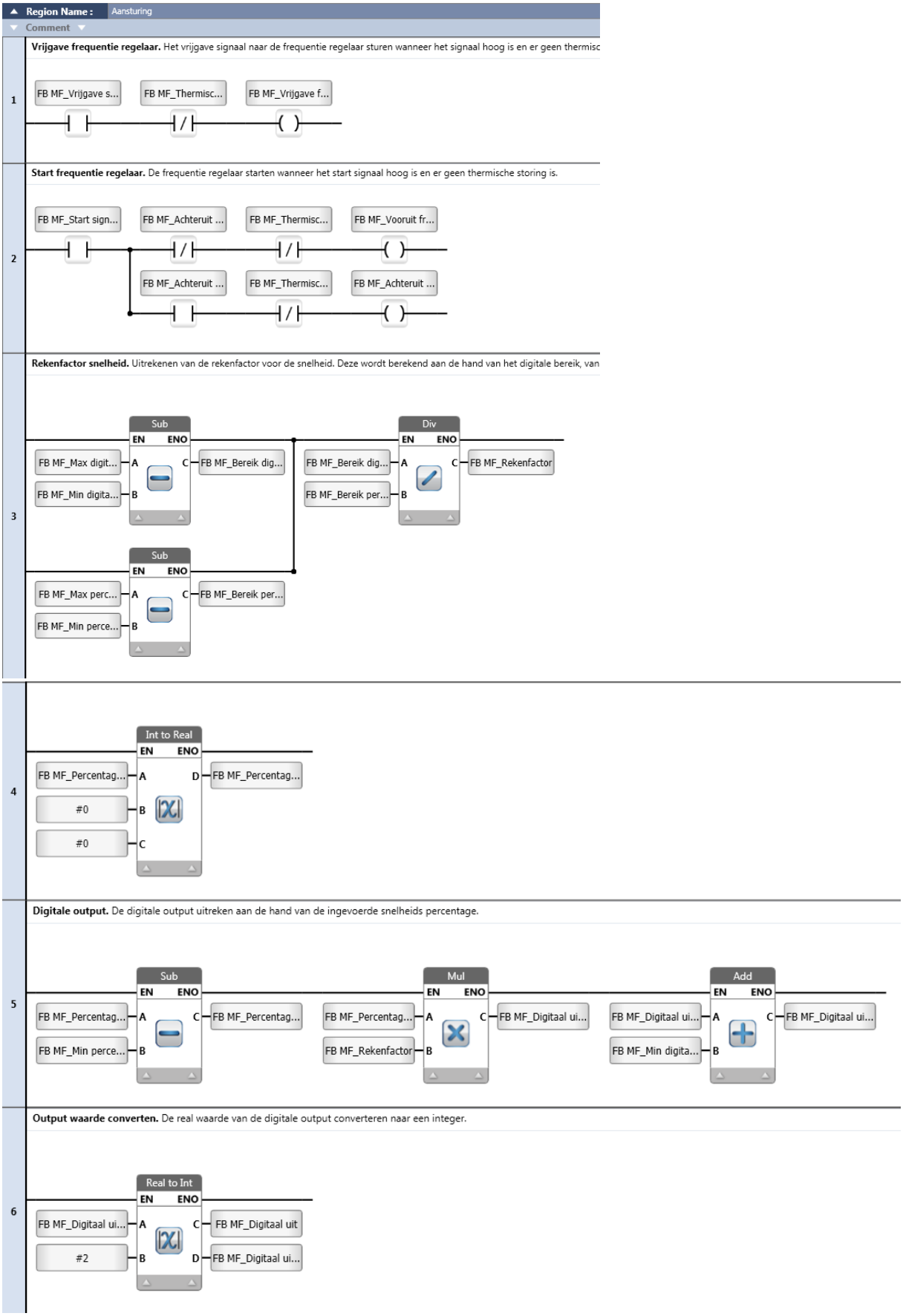
Regio omrekening verschil met de puls transmitter

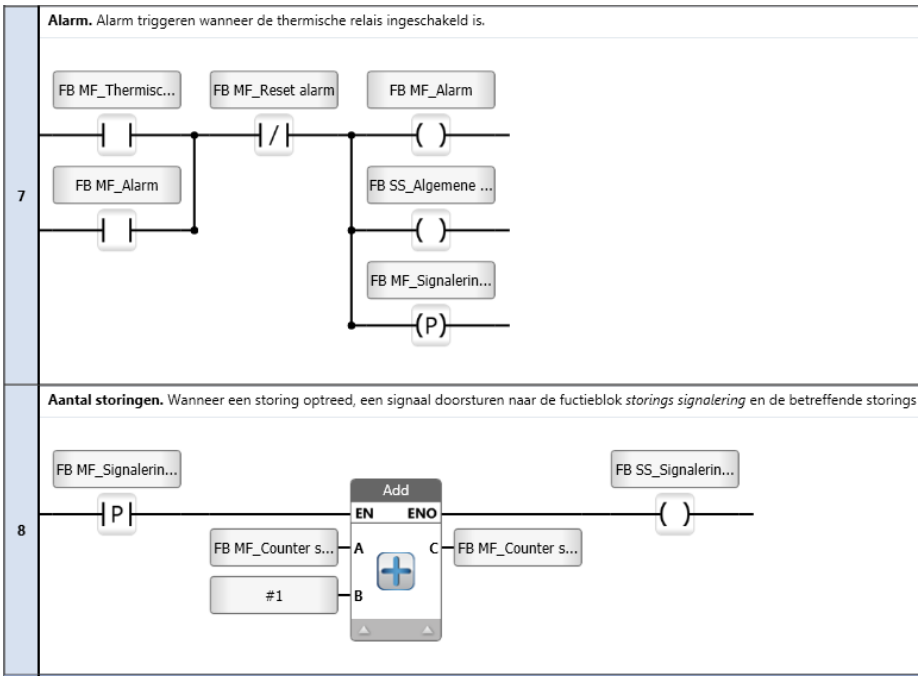


Functieblok motor

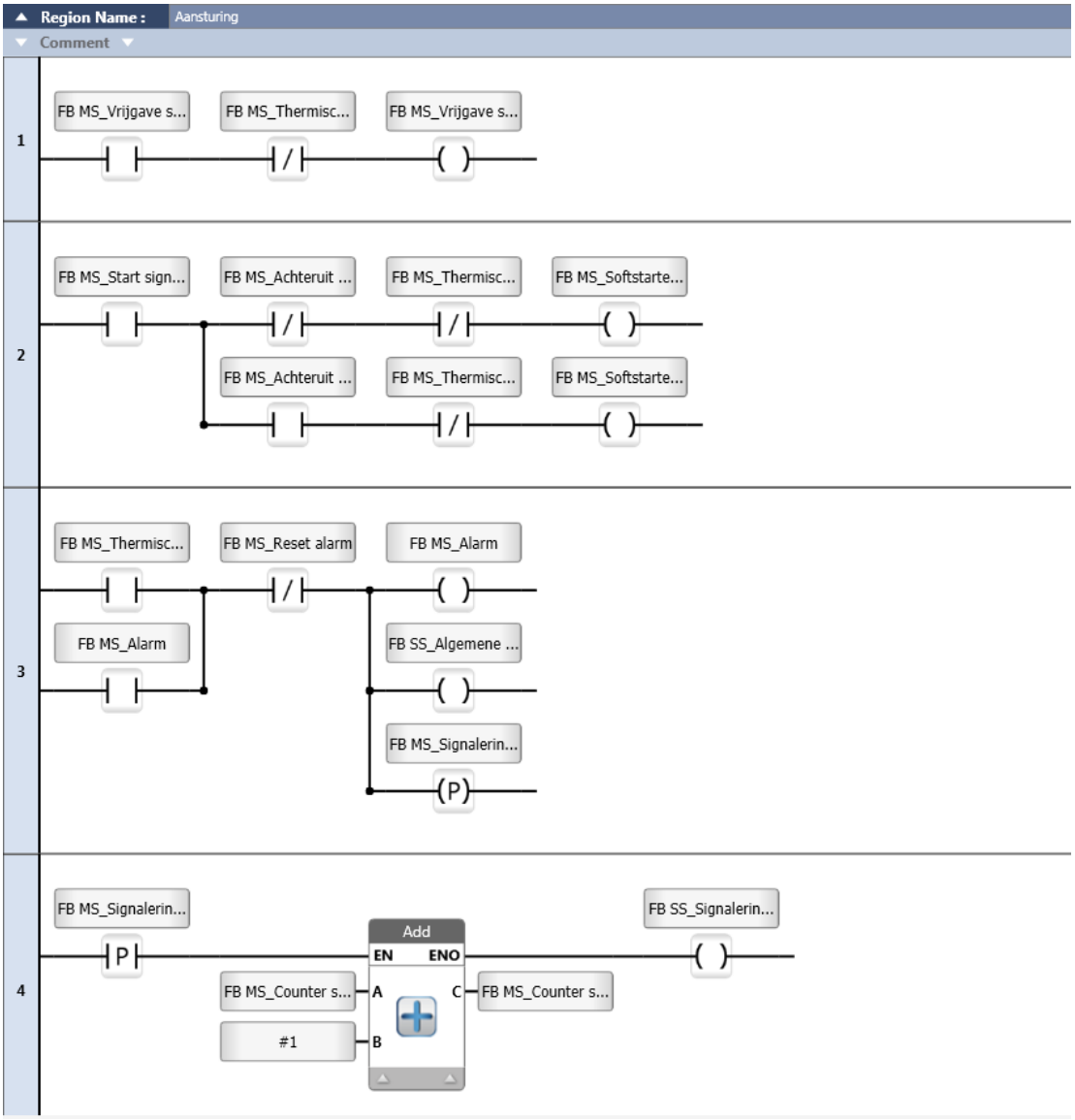


Functieblok motor frequentie

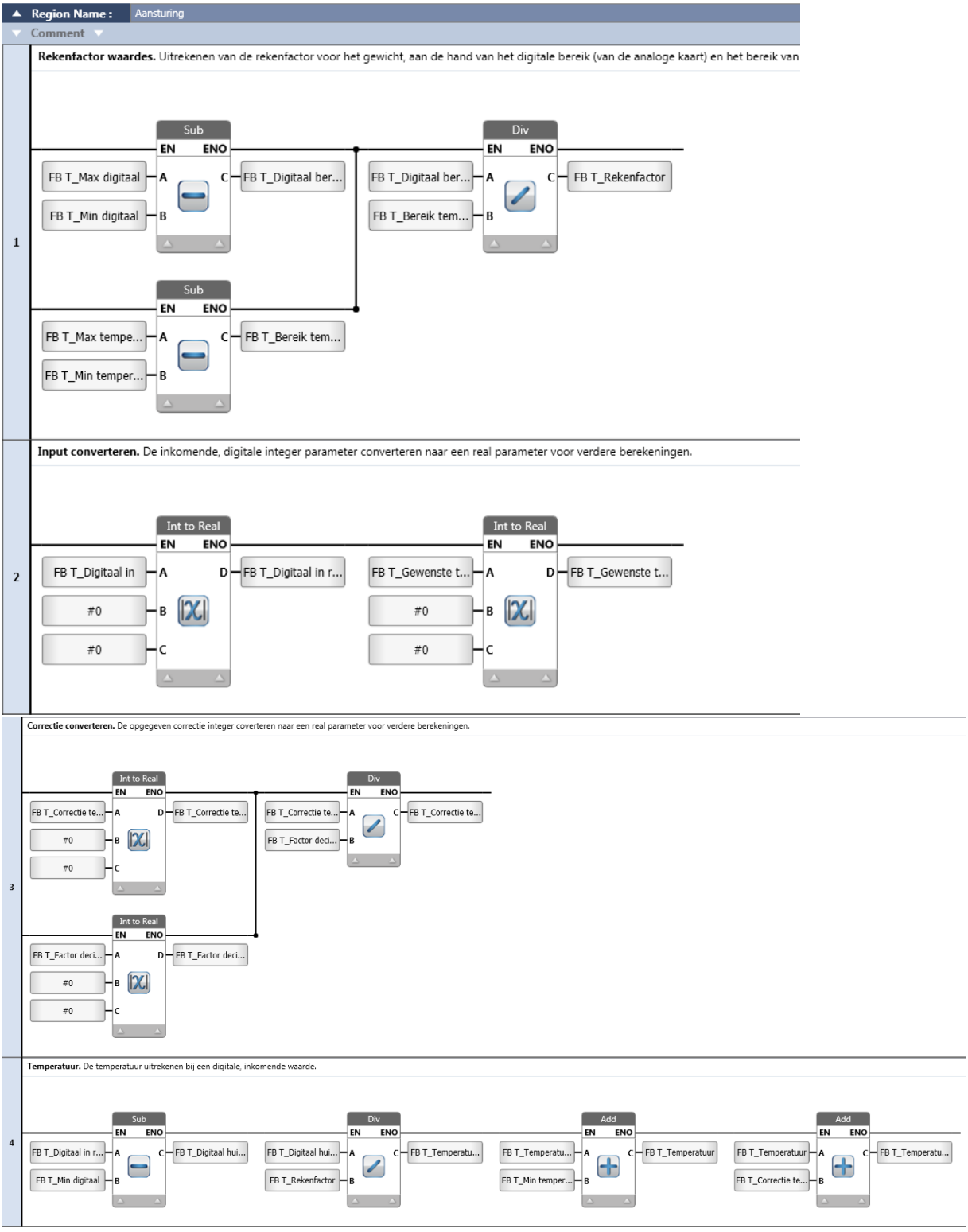



7
Alarm. Alarm triggeren wanneer de thermische relais ingeschakeld is.
FB MF_Thermisc...
FB MF_Reset alarm
FB MF_Alarm
FB MF_Alarm
FB SS_Algemene ...
FB MF_Signalerin...
(P)
8
Aantal storingen. Wanneer een storing optreed, een signaal doorsturen naar de fuctieblok storings signalering en de betreffende storings
FB MF_Signalerin...
P
Add
EN
ENO
FB MF_Counter s...
A
C
FB MF_Counter s...
#1
B
FB SS_Signalerin...

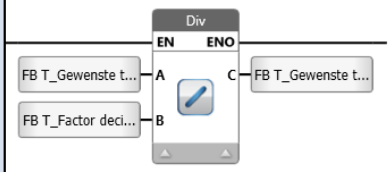
Functieblok motor softstarter



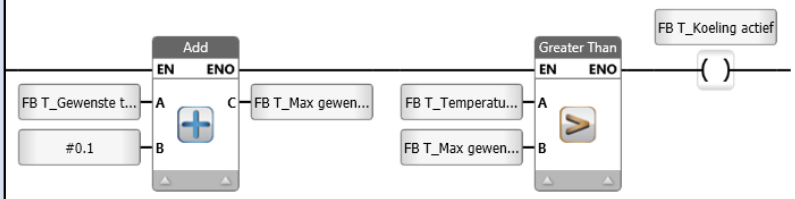
Functieblok Temperatuurmeting



5 **Gewenste waarde.** De gewenste waarde omrekenen aan de hand van de opgegeven aantal decimalen.

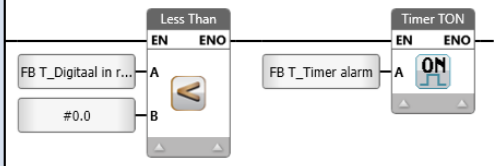


6 **Start koeling.** De koeling starten wanneer de gemeten temperatuur hoger is dan de gewenste temperatuur.

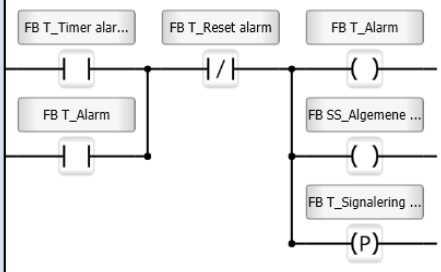


7 **Start verwarming.** De verwarming starten wanneer de gemeten temperatuur lager is dan de gewenste temperatuur.

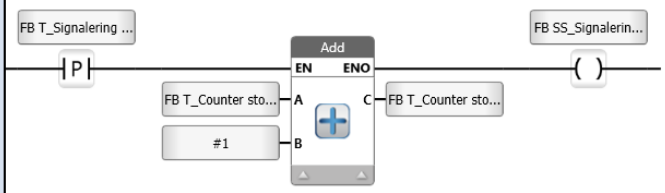
8 **Analoog controle.** Timer starten wanneer er geen analoog signaal binnen komt.

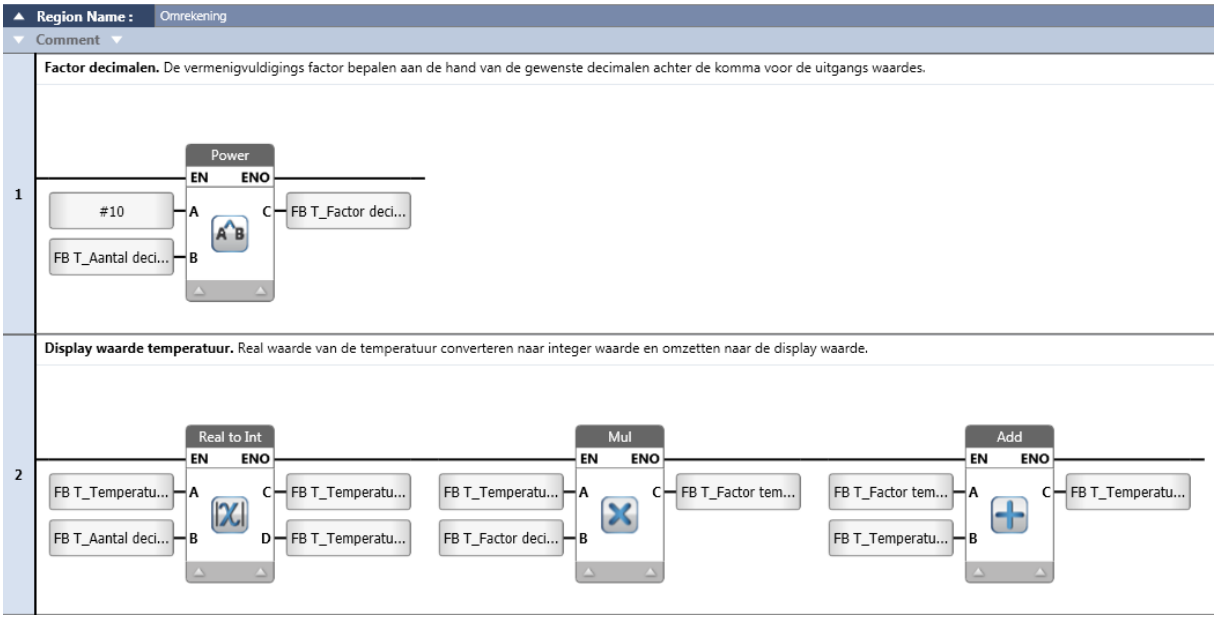


9 **Alarm.** Alarm triggeren wanneer de timer ouput "*timer alarm*" hoog is.

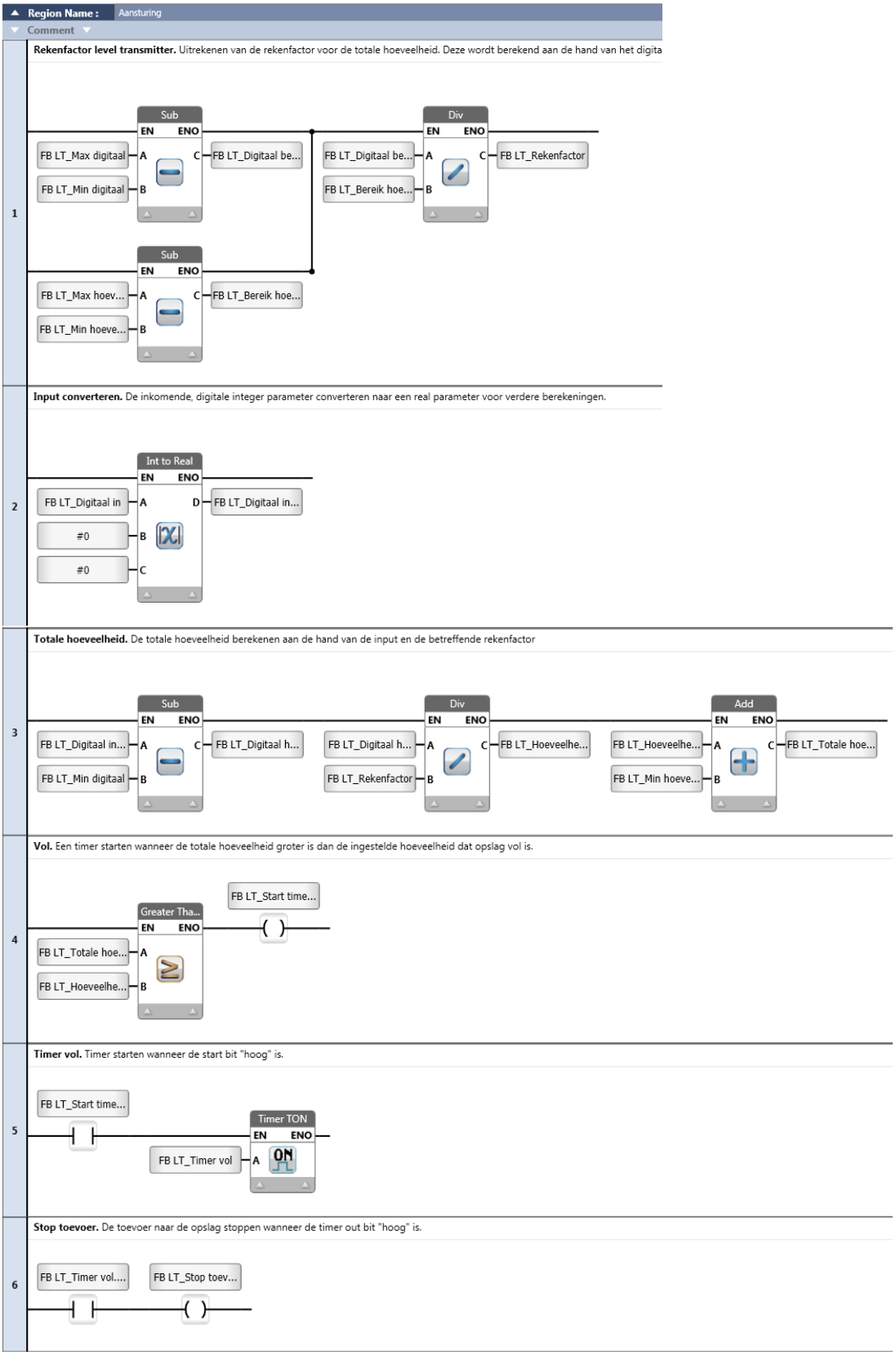


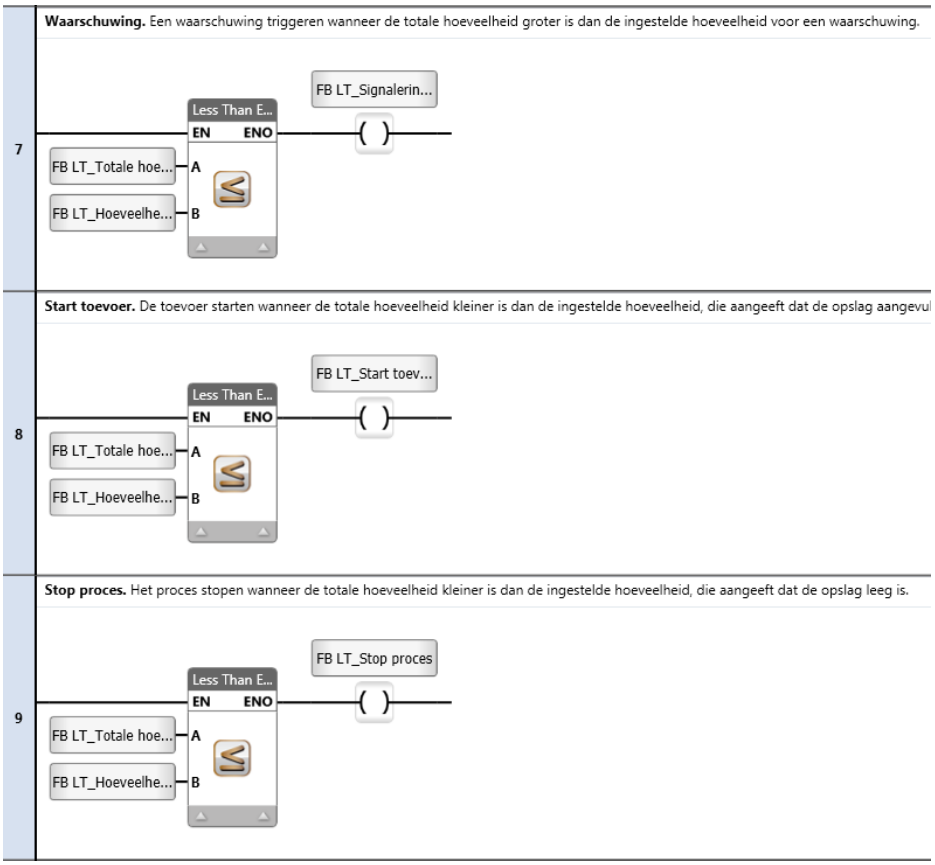
10 **Aantal storingen.** Wanneer een storing in de flow meter optreed, een signaal doorsturen naar de fuctieblok *storings signalering* en de betreffende

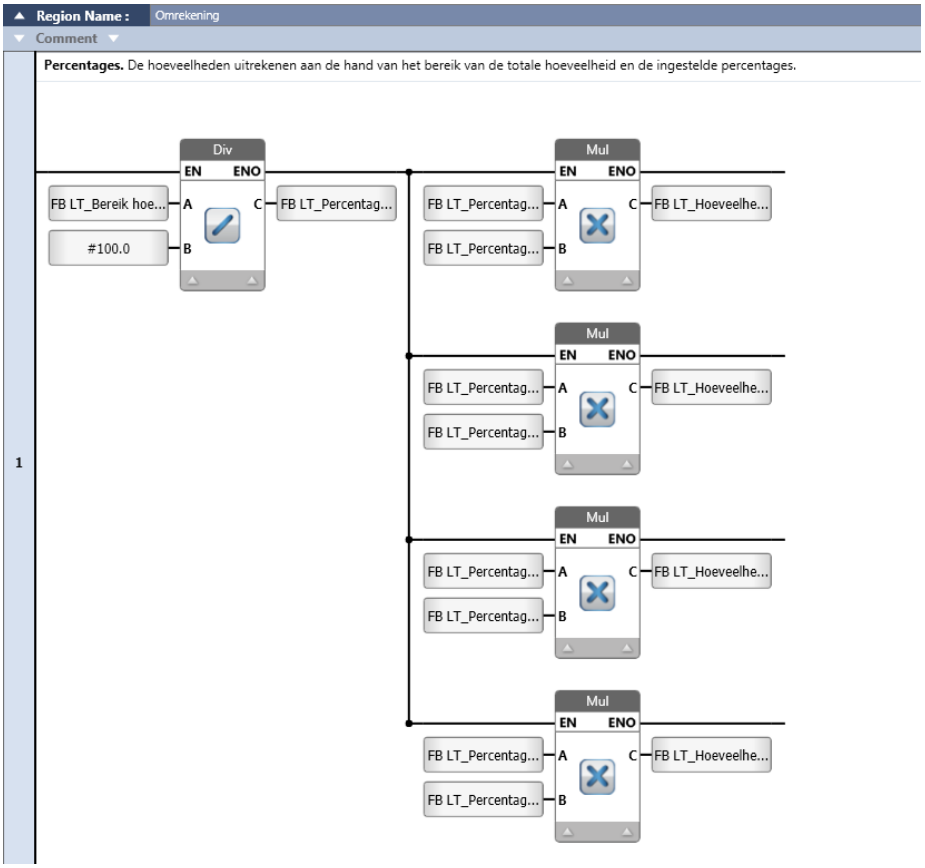



Region Name :
Omrekening
Comment
1
Factor decimalen. De vermenigvuldigings factor bepalen aan de hand van de gewenste decimalen achter de komma voor de uitgangs waardes.
Power
EN
ENO
#10
A
C
FB T_Factor deci...
FB T_Aantal deci...
B
2
Display waarde temperatuur. Real waarde van de temperatuur converteren naar integer waarde en omzetten naar de display waarde.
Real to Int
Mul
Add
FB T_Temperatu...
FB T_Aantal deci...
D
FB T_Factor tem...
FB T_Factor deci...
FB T_Temperatu...

Functieblok Level transmitter

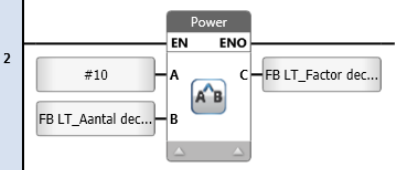



Waarschuwing. Een waarschuwing triggeren wanneer de totale hoeveelheid groter is dan de ingestelde hoeveelheid voor een waarschuwing.
7
Less Than E...
EN
ENO
FB LT_Totale hoe...
A
FB LT_Hoeveelhe...
B
FB LT_Signalerin...
Start toevoer. De toevoer starten wanneer de totale hoeveelheid kleiner is dan de ingestelde hoeveelheid, die aangeeft dat de opslag aangevul
8
Less Than E...
EN
ENO
FB LT_Totale hoe...
A
FB LT_Hoeveelhe...
B
FB LT_Start toev...
Stop proces. Het proces stopen wanneer de totale hoeveelheid kleiner is dan de ingestelde hoeveelheid, die aangeeft dat de opslag leeg is.
9
Less Than E...
EN
ENO
FB LT_Totale hoe...
A
FB LT_Hoeveelhe...
B
FB LT_Stop proces


Region Name :
Omrekening
Comment
Percentages. De hoeveelheden uitrekenen aan de hand van het bereik van de totale hoeveelheid en de ingestelde percentages.
1
Div
EN
ENO
FB LT_Bereik hoe...
A
C
FB LT_Percentag...
#100.0
B
Mul
FB LT_Percentag...
FB LT_Hoeveelhe...
Mul
FB LT_Percentag...
FB LT_Hoeveelhe...
Mul
FB LT_Percentag...
FB LT_Hoeveelhe...
Mul
FB LT_Percentag...
FB LT_Hoeveelhe...

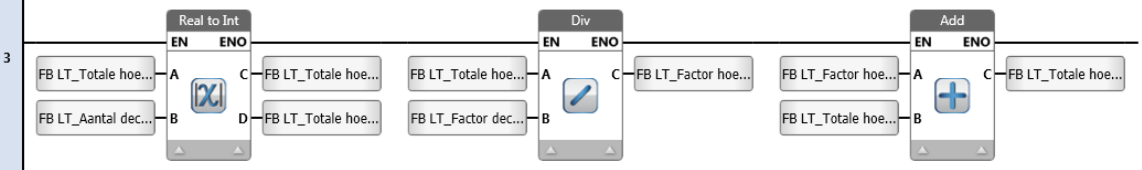
2

Factor decimalen. De vermenigvuldigings factor bepalen aan de hand van de gewenste decimalen achter de komma voor de uitgangs waardes.



3

Display waarde totale hoeveelheid. De real waarde van de totale hoeveelheid converteren naar een integer en omzetten naar een display waarde.



Bijlage E

Functies voor files

De File_IO Class wordt gebruikt om een groep van LASAL RTOS interface functies in te kapselen die gebruikt worden voor toegang tot files.

File_IO.FileOpen

Deze methode wordt aangeroepen voor het openen van een file. Je kan kiezen om een file te openen voor READ ONLY toegang met de parameter *create = 0*. Creëer een nieuwe file (ook als deze al bestaat) of definieer je eigen attributen met welke je de file wilt openen/creëren. De functie stuur de file handle of een 0 terug al een error optreed.

Input:

Path (^CHAR)	Een pointer naar de file pad string van de file die geopend moet worden.
Create (DINT)	0: open een bestaande file voor READ ONLY 1: creëer een nieuwe file ook als deze al bestaat. 2: gebruik gebruiker gedefinieerde attributen van parameter “attrib”
Attribs	Gebruiker gedefinieerde attributen om een file te openen.

Output:

Retcode (DINT)	De file ‘handle’, NIL als het niet mogelijk is de file te openen

Voorbeeld aanroep:

```
Retcode:= C_File_IO.FileOpen(“C:\\NewFile.txt”, 2, ATT_CREATE or ATT_READ_WRITE);
```

File_IO.FileClose

Deze methode wordt aangeroepen om een geopende file te sluiten.

Input:

Handle (pVoid)	De ‘handle’ van de file die gesloten moet worden.

Voorbeeld aanroep:

```
C_File_IO.FileClose(file);
```

File_IO.FileRead

Deze methode wordt aangeroepen om de data van een file te lezen. De file moet geopend zijn met de READ toegang. De functie zal de data lezen en opslaan in de buffer waarna deze het aantal gelezen bytes terug stuurt. Onthoud dat deze minder kunnen zijn dan dat ze gespecificeerd zijn in de ‘size’ parameter als het einde van de file is bereikt voordat de aangevraagde hoeveelheid data is gelezen.

Input:

Handle (DINT)	De ‘handle’ van de file die geopend is met de READ toegang.
Data (pVoid)	Een pointer naar een buffer om de file data op te slaan.
Size (UDINT)	De grote, in bytes die gelezen moet worden.

Output:

Retcode (UDINT)	Het aantal bytes die gelezen is. 0 als een error gedetecteerd is (of al aan het einde van de file). Let op dat deze waarde minder kan zijn.

Voorbeeld aanroep:
Ret0:= C_File_IO.FileRead(file, #buf[0], 10);

File_IO.FileWrite

Deze methode wordt aangeroepen wanneer data naar een file wordt geschreven. De file moet geopend zijn met de WRITE toegang. De functie zal de gespecificeerde hoeveelheid data van de buffer naar de file schrijven. De functie stuurt het aantal geschreven bytes terug of een 0 wanneer een error gedetecteerd wordt.

Input:

Handle (DINT)	De 'handle' van de file die geopend is met de WRITE toegang
Data (pVoid)	Een pointer naar een buffer om naar de file te schrijven
Size (UDINT)	De grote, in bytes om te schrijven

Output:

Retcode (UDINT)	Het aantal geschreven bytes, 0 als een error is gedetecteerd

Voorbeeld aanroep:
Ret0:= C_File_IO.FileWrite(path, #buf[0], 10);

File_IO.FileDelete

Deze methode wordt aangeroepen om een file te verwijderen. De methode krijgt een pointer naar de file pad string van de file die verwijderd moet worden. De functie stuurt een 0 terug als de file verwijderd is en <0 als er een error gedetecteerd is.

Input:

File (^CHAR)	Een pointer naar de file pad string van de file die verwijderd moet worden.

Output:

Retcode (DINT)	De file 'handle', NIL als het niet mogelijk is om de file te openen.

Voorbeeld aanroep:
Retcode := C_File_IO.FileDelete("C:\\OldFile.txt");

File_IO.FileSeek

Deze methode wordt aangeroepen om een specifieke positie te zoeken binnen een file. De functie krijgt een offset die relatief is aan de zoek mode. De "fromwhere" zoek mode wordt als volgt gegeven:
0: Zoek vanaf het begin van de file.
1: Zoek vanaf de huidige file positie.
2: Zoek vanaf het einde van de file.
De functie krijgt een nieuwe file positie terug of -37 wanneer de file positie groter is dan 32767. Alle andere, negatieve return waardes wijzen op een error.

Input:

Handle (DINT)	De 'handle' van de geopende file
Offset (UDINT)	De offset die relatief zoekt naar 'fromwhere'
Fromwhere (UDINT)	De basis zoek positie code (0, 1 of 2)

Output:

Retcode (UDINT)	De nieuwe file positie

Voorbeeld aanroep:

Ret0:= C_File_IO.FileSeek(file, 100, 0);

File_IO.FileTell
Deze methode wordt aangeroepen om de huidige file positie te geven. Een return waarde kleiner dan 0 wijst op een error.

Input:
Handle (DINT) De 'handle' van de geopende file

Output:
Retcode (DINT) De huidige file positite

Voorbeeld aanroep:
Ret0:= C_File_IO.FileTell(file);

File_IO.FileLength
Deze methode wordt aangeroepen om de huidige lengte van de file te geven. Een return waarde kleiner dan 0 wijst op een error.

Input:
Handle (DINT) De 'handle' van de geopende file

Output:
Retcode (DINT) De huidige file lengte

Voorbeeld aanroep:
Ret0:= C_File_IO.FileLength(file);

File_IO.CreateDir
Deze methode wordt aangeroepen om een nieuwe verwijzing te creëren.

Input:
Name (^CHAR) Een pointer naar de directory pad string om te creëren

Output:
Retcode (DINT) 0 als het gelukt is, < 0 wanneer een error optreed

Voorbeeld aanroep:
Ret0:= C_File_IO.CreateDir("C:\\Files");

File_IO.RemoveDir
Deze methode word aangeroepen om een verwijzing te verwijderen. De methode zal falen wanneer het de huidige verwijzing probeert te verwijderen, een non-empty directory of een root directory. De functie stuur een 0 uit als de uitvoer gelukt is en een < 0 als er een error gedetecteerd is.

Input:
Name (^CHAR) Een pointer naar de directory pad string die verwijderd moet worden.

Output:
Retcode (DINT) 0 als het gelukt is, < 0 wanneer een error optreed

Voorbeeld aanroep:
Ret0:= C_File_IO.RemoveDir(“C:\\Files”);

File_IO.FindFirst

Deze methode wordt aangeroepen om de verwijzing van een file te zoeken. De functie krijgt een file string. Als de string drive en/of directory informatie bevat dan kan deze geen standaard gedefinieerde karakters bevatten. In andere gevallen kunnen de * en ? karakters gebruikt worden. De functie zoekt voor de eerste directory entry die overeenkomen met de specificaties. Alleen de files die overeenkomen met een (niet)-meetellende attribuut mask worden gebruikt in de zoek actie. Deze ‘handle’ kan gebruikt worden in de volgende aanroepen tot de File.FindNext () aangeroepen wordt. Let wel dat deze ‘handle’ gesloten moet worden door de File_IO.FindClose () wanneer deze klaar is. Als er geen overeenkomende directory entry gevonden is zal de functie return waarde < 0 terug sturen.

Input:

Path (^CHAR)	Een pointer naar de directory pad en file string waarnaar gezocht moet worden (standaard gedefinieerde karakters als * en ? worden ondersteund)
Finfo (^_DDE_INFO)	Een pointer naar een Directory Entry Information structuur waar de teruggegeven file informatie in opgeslagen moet worden
Att_inkl (USINT)	Een mask van een file attribuut die meegerekend moet worden in de zoek actie
Att_exkl (USINT)	Een mask van een file attribuut die niet meegerekend moet worden in de zoek actie

Output:

Retcode (DINT)	De ‘handle’ van de eerste file die overeenkomt met de zoek criteria, < 0 als er geen overeenkomst gevonden is.

Voorbeeld aanroep:
Ret0:= C_File_IO.FindFirst(“*.txt”, #finfo, 0x00, 0x00);

File_IO.FindNext

Deze methode aangeroepen om een zoek actie voort te zetten. De functie krijgt de ‘handle’ van de vorige aanroep File_IO.FindFirst (). De functie zal volgende overeenkomende file vinden. Wanneer deze gevonden is, wordt de file informatie opgeslagen en de waarde 0 terug gestuurd. Een negatieve waarde wordt uitgestuurd wanneer er geen overeenkomsten meer gevonden zijn.

Input:

Handel (pVoid)	Een file ‘handle’ die teruggegeven wordt door de aanroep File_IO.FindFirst ()
Finfo (^_DDE_INFO)	Een pointer naar een Directory Entry Information structuur waar de teruggegeven file informatie in opgeslagen moet worden

Output:

Retcode (DINT)	0 als een overeenkomende directory entry gevonden is, < 0 wanneer er geen overeenkomsten file meer gevonden zijn.

Voorbeeld aanroep:
Ret0:= C_File_IO.FindNext(file, #finfo);

File_IO.FindClose

Deze methode wordt aangeroepen om een file zoek actie te sluiten. De functie krijgt een file ‘handle’ van de vorige aanroep File_IO.FindFirst ().

Input:

Handle (pVoid) Een file ‘handle’ teruggegeven door de functie File_IO.FindFirst ()

Output:
Retcode (DINT) 0 wanneer de ‘handle’ gesloten is, < 0 wanneer een error gedetecteerd is

Voorbeeld aanroep:
Ret0:= C_File_IO.FindClose(file);

File_IO.CloseAllOpenFiles
Deze methode wordt aangeroepen om alle geopende files te sluiten. Deze functie niet aangeroepen wanneer andere files geopend zijn door de opererende systeem of door ander threads.

Voorbeeld aanroep:
Ret0:= C_File_IO.CloseAllOpenFiles();

File_IO.Rename
Deze methode wordt aangeroepen wanneer de name van de file gewijzigd moet worden.

Input:
Oldname (^CHAR) Een pointer naar de huidige file name string
Newname (^CHAR) Een pointer naar de nieuwe file name string

Output:
Retcode (DINT) 0 als de file name gewijzigd is, < 0 wanneer een error gedetecteerd wordt

Voorbeeld aanroep:
Retcode:= C_File_IO.Rename(“OldName.txt”, ”NewName.txt”);

Functies voor FTP

FTP_CONNECT (udIP, pUser, pPassword)
Creëert een FTP connectie.

Input:

udIP (UDINT)	IP-adres van de FTP server
pUser (^CHAR)	Pointer naar de username voor server-login
pPassword (^CHAR)	Pointer naar de password voor server-login

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

0	Connectie ok
-1	Kan geen connectie maken met de HOST
-2	Server was geannuleerd
-3	Verkeerde password
-4	Verkeerde username
-10	Er is al een verbinding

FTP_EXT_CONNECT (udIP, uPort, pUser, pPassword, pCallbackFct)
Creëert een FTP connectie me geavanceerde parameter. De gebruiker gedefinieerde callback-functie zal reageren op elke serverresponse met een input waarde.

Input:

udIP (UDINT)	IP-adres van de FTP server
uPort (UINT)	FTP poort
pUser (^CHAR)	Pointer naar de username voor server-login
pPassword (^CHAR)	Pointer naar de password voor server-login
pCallbackFct (pVoid)	Pointer naar de user-callback-functie

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

0	Connectie ok
-1	Kan geen connectie maken met de HOST
-2	Server was geannuleerd
-3	Verkeerde password
-4	Verkeerde username
-10	Er is al een verbinding

FTP_CWD (pDirectory)
Functie verandert de verwijzing naar de server.

Input:

pDirectory (^CHAR)	Pointer naar de directoryname

Output:
dRetVal (DINT) Returnwaarde
Mogelijke return waardes:
0 Directory veranderd
-1 Directory bestaat niet of geen rechtvaardiging
-10 Geen connectie

FTP_PWD (pBuf)
De functie stuur de actuele verwijzing naar de server terug.

Input:
pBuf (^CHAR) Pointer naar de buffer waar de directortname naar toe wordt gekopieerd

Output:
dRetVal (DINT) Returnwaarde

Mogelijke return waardes:
0 Return waarde is correct
-1 Opdracht gefaald
-10 Geen connectie

FTB_MKDIR (pDirectory)
De functie creëert een verwijzing op de server.

Input:
pDirectory (^CHAR) Pointer naar de directoryname die gecreëerd moet worden.

Output:
dRetVal (DINT) Returnwaarde

Mogelijke return waardes:
0 Directory gecreëerd
-1 Directoryname niet mogelijk of geen rechtvaardiging
-10 Geen connectie

FTP_RMDIR (pDirectory)
De functie verwijderd een verwijzing op de server.

Input:
pDirectory (^CHAR) Pointer naar de directoryname die gecreëerd moet worden.

Output:
dRetVal (DINT) Returnwaarde

Mogelijke return waardes:
0 Directory gecreëerd

-1 Directoryname niet mogelijk of geen rechtvaardiging
-10 Geen connectie

FTP_DELE (pFile)
De functie verwijderd een file op de server.

Input:
pFile (^CHAR) Pointer naar file name, welke gedeletet moet worden op de server.

Output:
dRetVal (DINT) Return waarde

Mogelijke return waardes:
0 File verwijderd
-1 Filename niet mogelijk of geen rechtvaardiging
-10 Geen connectie

FTB_CLOSE ( )
De functie sluit de verbinding naar de FTP-server.

Output:
dRetVal (DINT) Return waarde

Mogelijke return waardes:
0 Verbinding gesloten
-10 Er is geen verbinding

FTP_LIST (pPath, pBuf_File, udLength)
Net als de DOS-command ‘dir’ stop de functie alle files in een buffer of een file. Alleen files en geen directorys.

Input:
pPath (^CHAR) Pointer naar de directoryname
pBuf_File (^CHAR) Pointer naar de buffer of filename
udLength (UDINT) Lengte van de buffer of maximale levenslengte

Output:
dRetVal (DINT) Return waarde

Mogelijke return waardes:
0 Opdracht succesvol
-1 Opdracht afgewezen of geen aanvraag van de server
-4 Niet mogelijk een verbinding te openen
-6 Transmissie onderbroken door de server
-7 Transmissie error
-10 Geen connectie

FTP_NLIST (pPath, pBuf_File, udLength)
Net als de DOS-command 'dir' stop de functie alle files en directorys in een buffer of een file.

Input:

pPath (^CHAR)	Pointer naar de directoryname
pBuf_File (^CHAR)	Pointer naar de buffer of filename
udLength (UDINT)	Lengte van de buffer of maximale levenslengte

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

0	Opdracht succesvol
-1	Opdracht afgewezen of geen aanvraag van de server
-4	Niet mogelijk een verbinding te openen
-6	Transmissie onderbroken door de server
-7	Transmissie error
-10	Geen connectie

FTP_PUT (pBuf_File, udLength, pServerFile)
De functie neemt data van de buffer of file en creëert een file met de gegeven naam op de FTP-server.

Input:

pBuf_File (^CHAR)	Pointer naar de buffer of filename
udLength (UDINT)	Lengte van de databuffer of maximale levenslengte
pServerFile (^CHAR)	Pointer naar de filename (wordt gecreëerd op de FTP-server)

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

>=0	Aantal bytes (overgedragen aan de server)
-1	Opdracht afgewezen of geen aanvraag van de server
-4	Niet mogelijk een verbinding te openen
-5	FTP-class heeft de transmissie geannuleerd
-6	Transmissie onderbroken door de server
-7	Transmissie error
-10	Geen connectie

FTP_APPEND (pBuf_File, udLength, pServerFile)
De functie neemt data van de buffer of file en voegt deze data toe aan een bestaande file met de gegeven naam op de FTP-server.

Input:

pBuf_File (^CHAR)	Pointer naar de buffer of filename
udLength (UDINT)	Lengte van de databuffer of maximale levenslengte

pServerFile (^CHAR)	Pointer naar de filename (wordt toegevoegd naar file op de FTP-server)

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

>=0	Aantal bytes (overgedragen aan de server)
-1	Opdracht afgewezen of geen aanvraag van de server
-4	Niet mogelijk een verbinding te openen
-5	FTP-class heeft de transmissie geannuleerd
-6	Transmissie onderbroken door de server
-7	Transmissie error
-10	Geen connectie

FTP_GET (pBuf_File, udLength, pServerFile)
De functie leest de data van de file en schrijft deze data naar een file of buffer.

Input:

pBuf_File (^CHAR)	Pointer naar de buffer of filename
udLength (UDINT)	Lengte van de databuffer of maximale levenslengte
pServerFile (^CHAR)	Pointer naar de filename (wordt gelezen op de FTP-server)

Output:

dRetVal (DINT)	Return waarde

Mogelijke return waardes:

>=0	Aantal bytes (overgedragen aan de server)
-1	Opdracht afgewezen of geen aanvraag van de server
-4	Niet mogelijk een verbinding te openen
-5	FTP-class heeft de transmissie geannuleerd
-6	Transmissie onderbroken door de server
-7	Transmissie error
-10	Geen connectie

FTP_TRANSFTYPE (bASCII, bFile)
Transmissie in ascii of binair, buffer of file.

Input:

bASCII (BOOL)	1 = ASCII transmissie (default) 0 = Binary transmissie
bFile (BOOL)	1 = File is buffer 0 = RAM is buffer (default)

Output:

dRetVal (DINT)	Return waarde

Mogelijk return waardes:

0	Opdracht succesvol
-1	Opdracht afgewezen of geen aanvraag van de server

-10 Geen connectie

FTP_SET_OPT (pValue, option)

Input:

pValue (pVoid)	Waarde
Option (UDINT)	Set option

Output:

Rc (DINT)	0 = ok, -1 = error (ptr = NIL, ongeldige option)