Scriptie

Hardware Simulator

Een stap dichter richting totale “plant” simulatie.

Opdrachtgever:	**Imtech Vonk** **Modem 30** **7741 MJ Coevorden**
Plaats:	Coevorden
Datum:	4 Januari 2011
Auteur:	Mark Haanstra
Studentnummer:	1531233
Studierichting:	Industriële Automatisering
Hogeschool:	Hogeschool Utrecht
Bedrijfsbegeleider:	J. Töllner
1ste docent begeleider:	J. Schouten
2de docent begeleider:	E. van Akkeren

Afstudeerproject:

Uitbreiding hardware simulator

Scriptie

Opdrachtgever:	Imtech Vonk Modem 30 7741 MJ, Coevorden
Bedrijfsbegeleider:	J. Töllner
Plaats:	Coevorden
Datum:	4 Januari 2011
Auteur:	Mark Haanstra
Studentnummer:	1531233
Afstudeerperiode:	30 augustus 2010 t/m 28 januari 2011
School:	Hogeschool Utrecht Oudenoord 700 3513 EX, Utrecht
Studierichting:	Industriële Automatisering
Inleverdatum:	4 januari 2011
1ste docent begeleider:	J. Schouten
2de docent begeleider:	E. van Akkeren
Versie:	1

Voorwoord

Dit scriptie is geschreven ter verdediging van mijn afstudeerstage, die ik gedurende 20 weken heb mogen uitvoeren bij Imtech Vonk te Coevorden. Deze afstudeerstage was een afronding voor mijn opleiding Industriële Automatisering (IA) aan de Hogeschool Utrecht.

Tijdens het afstuderen heb ik veel geleerd over het programmeren in LabVIEW, het proces van ontwerp strategie naar een werkend product.
Dit afstudeerverslag is bedoeld voor alle geïnteresseerden binnen Imtech Vonk die meer te willen weten komen over de hardware simulator en hoe de programma structuur in elkaar zit.

Hierbij wil ik in de eerste plaats mijn afstudeerbegeleider dhr. J. Töllner bedanken voor zijn inzet en begeleiding tijdens mijn afstudeerperiode. Ook wil ik via deze weg het bedrijf Imtech Vonk bedanken voor het ter beschikking stellen van de afstudeerplek en alle mensen bedanken die mij tijdens de afstudeerperiode hebben geholpen met hun advies en kennis.

Verder gaat mijn dank uit naar dhr. J Schouten voor begeleiding van mijn afstuderen vanuit de Hogeschool Utrecht.

Ten slotte richt ik mijn dankwoord tot mijn ouders, broertje en mijn vriendin, die mij steeds hebben gesteund gedurende mijn studieperiode.

Mark Haanstra

Coevorden, 4 Januari 2011

Samenvatting

Deze scriptie is tot stand gekomen omdat binnen Imtech Vonk de behoefte bestond om de bestaande hardware simulator uit te breiden. In deze scriptie staat de weg beschreven vanaf de probleem stelling tot aan het op te leveren product met daartussen de gemaakte keuzes.

De hardware simulator wordt gebruikt voor het uitvoerig testen van de besturingskasten die Imtech Vonk ontwerpt en bouwt. De hardware simulator is enkele jaren geleden ontwikkeld met een minimum aan functionaliteit en gebruiksvriendelijkheid.
De hardware simulator simuleert digitale ingangen, digitale uitgangen, analoge ingangen en analoge uitgangen van de te testen besturingskast.

De probleemstelling is dat er minimaal kan worden gesimuleerd, bepaalde simulaties niet mogelijk zijn zoals een MODBUS simulatie en het instellen van de hardware simulator is vrij complex.

Het doel van de opdracht is dan ook om de functionaliteit en de gebruiksvriendelijkheid te verhogen. Daarmee zijn de volgende subopdrachten gepaard: het ontwerpen van MODBUS simulaties, uitbreiden simulatiemogelijkheden, het toevoegen van een OPC koppeling en gebruiksvriendelijkheid van de IO configuratie maken.

Hoe de programma's moeten werken, welke stappen ze moeten ondernemen, welke simulatie mogelijkheden er moeten komen zijn in overleg met dhr. Jürgen Töllner gerealiseerd en onderbouwt met de theorie (als dit van toepassing is).

Hoofdstuk 1 gaat de inleiding van de scriptie. Hierin wordt de structuur van Imtech Vonk beschreven, de aanleiding van de opdracht en algemene informatie.

Hoofdstuk 2 gaat over de structuur van de opdacht, hoe de opdracht is opgebouwd.

Hoofdstuk 3 is een inleidend verhaal over LabVIEW. Hierin staat beschreven wat LabVIEW inhoudt, hoe het programma is opgebouwd en hoe bepaalde functies er in LabVIEW uitzien.

In hoofdstuk 4 wordt de MODBUS simulatie besproken. Hierin staat van begin tot het eind beschreven hoe de MODBUS simulatie tot stand is gekomen.

In hoofdstuk 5 wordt de User-friendly IO configuratie besproken. Hierin staat van begin tot het eind beschreven hoe de User-friendly IO configuratie tot stand is gekomen.

In hoofdstuk 6 wordt de uitbreiding van de simulatie mogelijkheden besproken. Hierin staat van begin tot het eind beschreven hoe de uitbreiding van de simulatie mogelijkheden tot stand zijn gekomen.

Inhoudsopgave

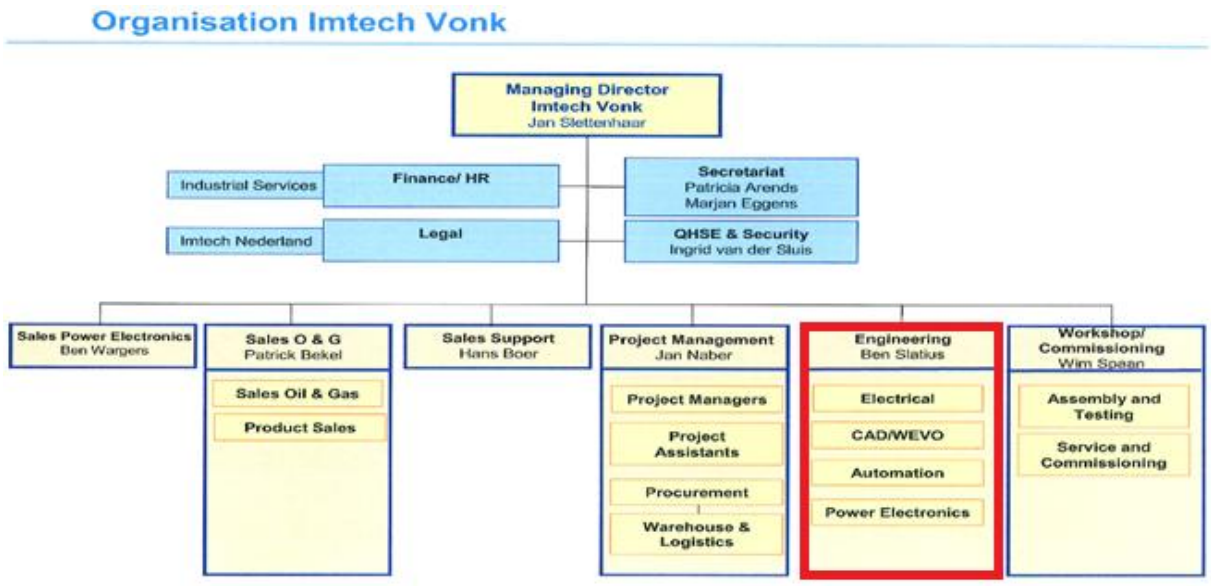
1. Inleiding

1.1 Het afstudeerbedrijf

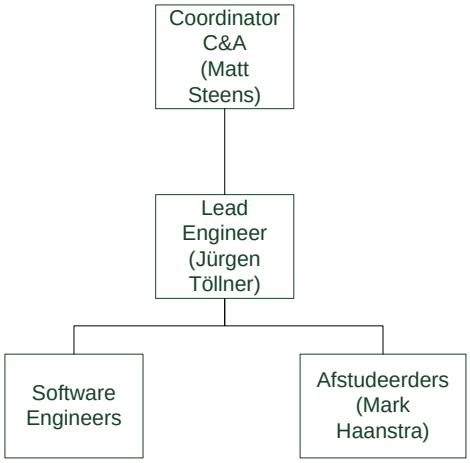
Imtech N.V. is een Europese technische dienstverlener op het gebied van elektrotechniek, ICT en werktuigbouw. Deze disciplines worden gebundeld om integrale en multidisciplinaire totaal oplossingen te bieden. Imtech levert hierbij consultancy, design engineering, implementatie, onderhoud, service en projectmanagement gedurende de complete levenscyclus van de technologie en bedrijfsprocessen bij klanten. Het bedrijf telt ongeveer 23.000 medewerkers en heeft meer dan 415 vestigingen verdeeld over meer dan 20 landen wereldwijd.

Onderdeel van de Imtech N.V. is Imtech Vonk, te Coevorden. Imtech Vonk is een engineering en contracting bedrijf en als zodanig actief in de markten: olie & gas, vermogenselektronica en energietechniek. Ze zijn veelal opererend in Europa, Verre Oosten, Midden-Oosten en Afrika. Imtech Vonk lever totaal projecten op basis van ontwerp tot inbedrijfstelling en after-sales service. De projecten omvatten levering van onder andere: vermogens elektronica, veiligheidssystemen, proces bewaking en besturing (PLC/ SCADA of DCS), containers met diverse elektrische installaties, elektriciteitsopwekking, distributie & omzetting en houtgestookte warmtekrachtcentrales.

Ik val onder de afdeling Control & Automation. Deze afdeling houdt zich bezig met het automatiseren van industriële processen.
De projecten van de afdeling Control & Automation lopen ver uiteen van Houtgestookte WKK's tot Energie centrales van ziekenhuizen. De afdeling is te zien in het figuur 1.1 onder afdeling Enginering, Automation (het rode kader). In figuur 1.1 is het organogram van Imtech Vonk te zien. In figuur 1.2 is het organogram van de afdeling Control & Automation te zien.



figuur 1.1. Organogram Imtech Vonk.



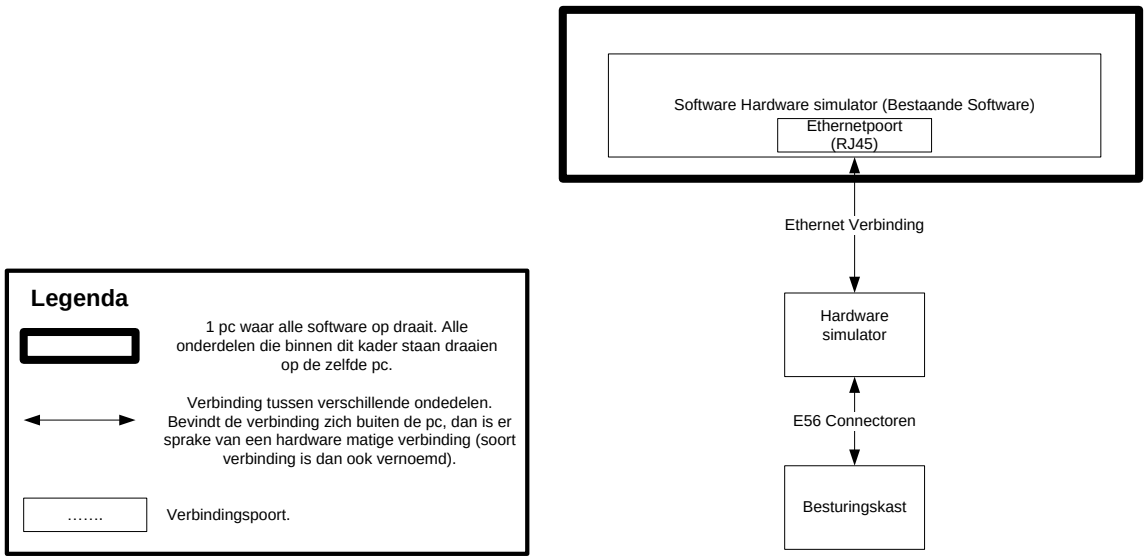
figuur 1.2. Organogram afdeling Control & Automation.

1.2 De aanleiding

Imtech Vonk test haar besturingskasten met een hardware simulator. Deze hardware simulator simuleert het te besturen proces. Dit houdt in dat alle IO's, digitale ingangen, digitale uitgangen, analoge ingangen, analoge uitgangen van de besturingskast worden aangesloten op de hardware simulator. Deze zal de IO's dan aansturen/ uitlezen. Per product dat door Imtech Vonk wordt gemaakt wordt samen met de klant getest.
Deze testen worden FAT (Factory Acceptions Test) testen genoemd. Deze FAT testen worden uitgevoerd om de besturingskasten uitvoerig te kunnen testen waardoor problemen kunnen worden opgespoord en kunnen worden verholpen.

Vanuit Imtech Vonk is er de behoefte om de bestaande hardware simulator uit te breiden.
Het uitbreiden van de hardware simulator moet het mogelijk maken om de besturingskasten beter en uitvoeriger te kunnen testen. Hieruit stroomt voort dat er een betere FAT test kan worden afgenomen (omdat de kast al eens een keer goed is getest).Hierdoor ontstaat er een betere indruk richting de klant met oog op het functioneel testen van de besturingskasten.

De huidige situatie is schematisch weergegeven in figuur 1.3.



figuur 1.3. Huidige situatie.

1.3 De probleemstelling

De hardware simulator is enkele jaren geleden ontwikkeld door een student van de opleiding Industriële Automatisering van de Hogeschool Utrecht. In deze ontwikkeling zijn de hardware componenten van de hardware simulator onderzocht en uitgezocht en is de hardware simulator ontwikkeld. Qua software is het hoognodige ontwikkeld om met de hardware simulator te kunnen werken. Nu is het gewenst om een aantal tekortkomingen op te lossen en er moeten nieuwe functionaliteiten aan toe worden gevoegd. Het uitbreiden van de hardware simulator moet het mogelijk maken om een betere indruk over te brengen naar de klant met oog op het functioneel testen van de besturingskasten, intern zorgt het ervoor dat de functionaliteit van de testen hoger wordt en daar mee de efficiëntie omhoog gaat.

De huidige situatie van de hardware simulator is:

- 192 digitale ingangen,
- 256 digitale uitgangen (potentiaal vrije contacten),
- 96 analoge uitgangen (4mA t/m. 20mA passief),
- 16 analoge ingangen (4mA t/m. 20mA passief).

De problemen en tekortkomingen van de hardware simulator zijn:

- Configuratie van IO is vrij complex en kan alleen door een ervaren engineer worden uitgevoerd,
- Huidige 4mA t/m. 20mA loops alleen passief,
- Niet makkelijk om NAMUR schakelaars te simuleren,
- Huidige IO te traag voor snelle proces simulatie.

De gewenste uitbreiding is van de hardware simulator zijn:

- MODBUS RTU/ASCII simulatie (RS232/RS485),
- MODBUS RTU/ASCII simulatie over TCP (Ethernet),
- High speed IO functionaliteit toevoegen,
- “User-friendly” IO configuratie sofware (Mogelijkheid om IO gegevens direct van EPLAN in te lezen),
- Uitbreiding van simulatie mogelijkheden,
- Mogelijkheid om actieve 4mA t/m 20mA signalen uit te lezen.

Eventuele uitbreidingen van de hardware simulator is:

- Foundation fieldbus simulatie.

Om aan al deze punten te kunnen vol doen is er meer tijd nodig dan het half jaar dat staat voor het afstuderen.
Om tot een realistische opdracht te komen, wat in een half jaar kan worden volbracht, zijn de tekortkommingen/ gewenste uitbreidingen doorgenomen met de stagebegeleider, dhr. J. Töllner.
In overleg met de stagebegeleider zijn er punten gekozen, waarbij hij de hoogste prioriteit had om uit te voeren.

De opdracht die daar uit volgt is beschreven in het volgende hoofdstuk, 1.4 Doelstelling.

1.4 Doelstelling

De doelstelling van het project is het uitbreiden van de functionaliteit en het uitbreiden van de gebruiksvriendelijkheid van de hardware simulator.

Hierdoor worden de simulatie mogelijkheden beter, waardoor de besturingskasten beter getest kunnen worden.
Door de gebruiksvriendelijkheid te vergroten kunnen niet alleen ervaren engineers maar ook monteurs gebruik maken van de hardware simulator.

Het doel van de opdracht is onderverdeeld in de onderdelen “Gebruiksvriendelijkheid” en “Uitbreiden functionaliteit”.
Per onderdeel zijn er subonderdelen waaruit de opdracht bestaat. De globale eisen, randvoorwaarden en prioriteiten zijn per subonderdeel van de opdracht beschreven.

Het belang voor het bedrijf is dat alle punten van de opdracht goed worden afgesloten.
Dit betekend dat er weer een goede stap is gezet in de richting van het uiteindelijke doel, het volledig kunnen simuleren van een proces en hierdoor de regel/ besturingskasten zo goed mogelijk kunnen te testen.

Alle software programma’s worden geschreven in het programma LabVIEW.

Er is gekozen om de software te schrijven in LabVIEW om de volgende redenen:

- Binnen Imtech Vonk zijn er LabVIEW licenties aanwezig,
- De bestaande software is al geschreven in LabVIEW,
- LabVIEW is breed inzetbaar in de besturingstechniek, kan verschillende protocollen aan.

De subonderdelen zijn als volgt beschreven:

1.4.1 Gebruiksvriendelijkheid

"User-friendly"IO configuratie software.

Globale eisen/ randvoorwaarden:

- Een IO lijst van EPLAN kunnen inlezen,
- Aan de hand van de IO lijst van Eplan de IO's van de simulator kunnen configureren,
- Aan de hand van de geconfigureerde IO's een tekstfile kunnen genereren waarin staat hoe de Hardwaresimulator moet worden aangesloten op de besturingskast.

1.4.2 Uitbreiden functionaliteit

MODBUS RTU/ ASCII simulatie (RS232/ RS485)

Globale eisen/ randvoorwaarden:

- Zowel Master en Slave,
- De Master en Slave moeten over de functiecodes 1, 3, 15 en 16 beschikken,
- De Master en Slave moeten zowel het RTU als het ASCII protocol aankunnen.

MODBUS TCP simulatie (ETHERNET)

Globale eisen/ randvoorwaarden:

- Er moet een Master en een Slave worden gemaakt,
- De Master en Slave moeten over de functiecodes 1, 3, 15 en 16 beschikken.

De Master en Slave moeten met het TCP protocol werken.

- *OPC koppeling voor alle IO*

Globale eisen/ randvoorwaarden:

- De OPC koppeling wordt door LabVIEW beheert,
- In de OPC koppeling moeten alle IO waarden worden opgeslagen
- Verschillende computers moeten gebruik kunnen maken van de OPC.

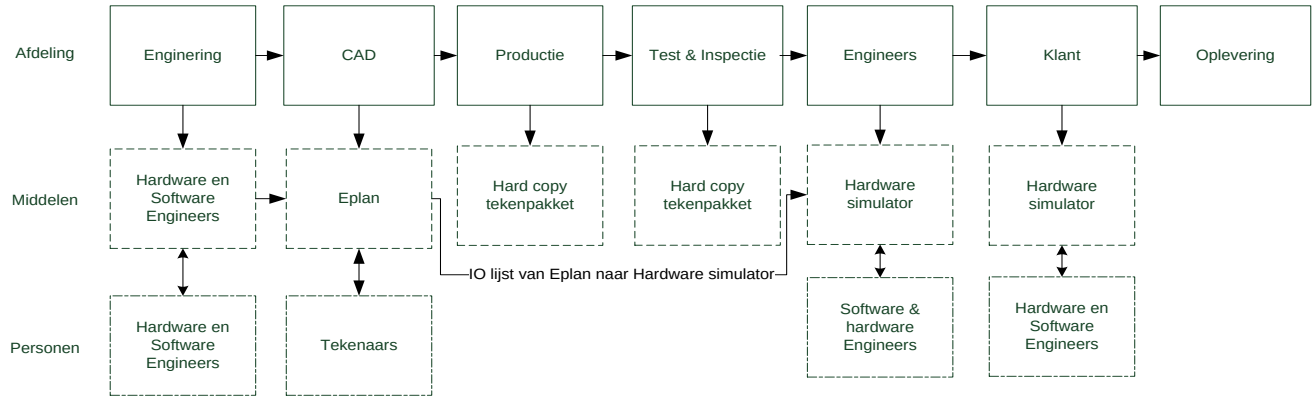
Uitbreiding van simulatie mogelijkheden

Globale eisen/ randvoorwaarden:

- Het kunnen simuleren van kleppen, motoren, level transmittors, temperatuurgestuurde kleppen.
- Het kunnen simuleren van complexe processen.

1.5 Stakeholders

Verschillende afdelingen binnen het bedrijf kunnen invloed hebben op het project. In figuur 1.4 is een schematische weergave te zien van de afdelingen die voorkomen in het proces. Hierin is ook te zien welke afdelingen gebruik maken van de hardware simulator.



figuur 1.4. Stakeholders.

1.6 Opbouw van het document (leeswijzer)

De opbouw van de scriptie is als volgt.

In hoofdstuk 2 wordt de pc waarde software opstaat behandelt. Hierin is te lezen hoe de programma's onderling met elkaar communiceren wat LabVIEW inhoudt en waarom er voor LaBVIEW is gekozen.

In hoofdstuk 3 wordt de MODBUS simulatie behandeld. Er wordt begonnen met de theorie, hoe het MODBUS protocol is opgebouwd, en wat de verschillen zijn tussen de verschillende protocollen. Daarna wordt beschreven hoe MODBUS simulatie is opgebouwd. Er wordt afgesloten met een conclusie.

In hoofdstuk 4 wordt de "User-friendly"IO configuratie behandeld. Hierin wordt beschreven waaraan de "User-friendly" interface moet voldoen, hoe de "User-friendly" IO configuratie is opgebouwd. Hoe de interface tot stand is gekomen.

In hoofdstuk 5 wordt de "Uitbreiding simulatie mogelijkheden" behandeld.

In hoofdstuk 6 wordt de OPC koppeling behandeld.

In dit verslag worden verschillende talstelsels gebruikt. Om aan te geven welk talstelsel er gebruikt wordt staat bij elk getal een subscript met een cijfer. Hieronder staan welke talstelsels er zijn en welk subscript cijfer deze hebben.

- Binaire talstelsel$_2$,
- Decimale talstelsel$_{10}$,
- Hexadecimale talstelsel$_{16}$.

Mocht er achter een getal geen subscript staan, dan wordt er gebruik gemaakt van het decimale talstelsel.

Bronvermelding: bronvermelding wordt gedaan aan de hand van een superscript met blokhaken [].
Staat er een cijfer tussen de blokhaken, dan gaat het om een boek. Staat er een letter tussen de blokhaken, dan gaat het om een internetsite of een digitaal document. Deze bronnen zijn te vinden in de literatuurlijst.

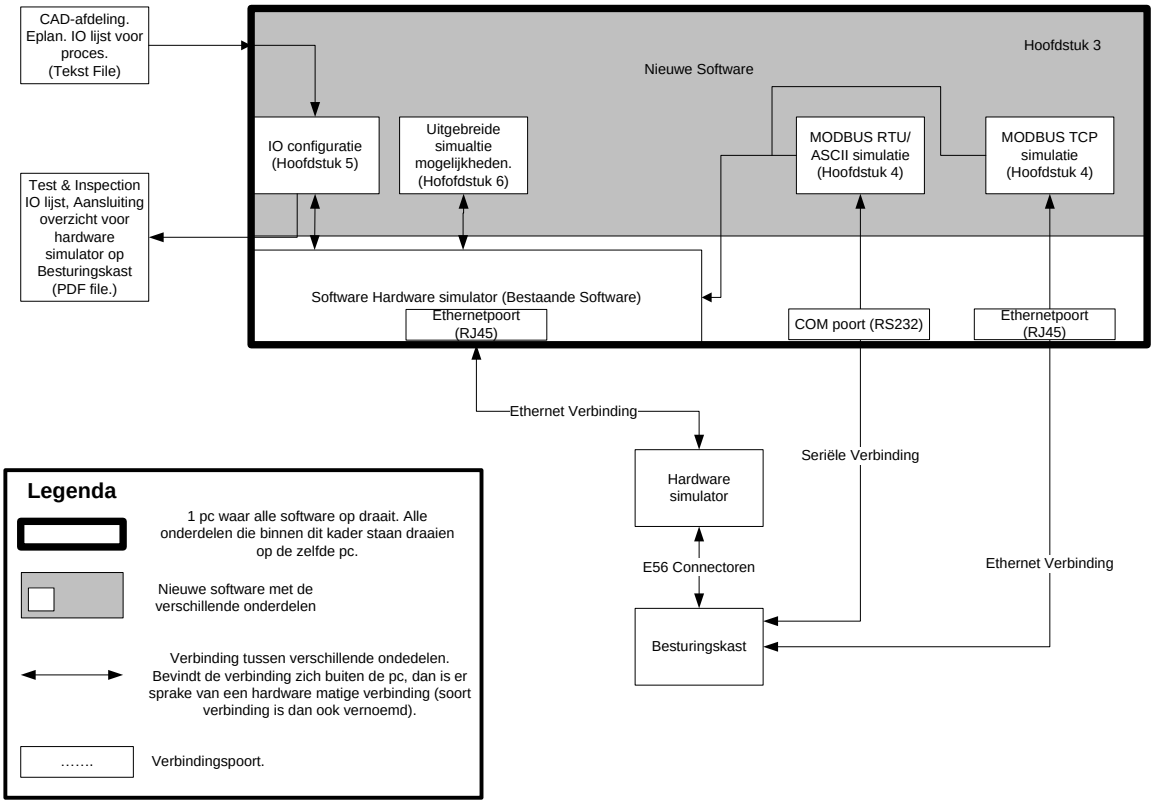
2. De structuur van de opdracht

Zoals de huidige situatie beschrijft bestaat de situatie van de hardware simulator uit de volgende onderdelen: een pc waarop de software draait en de uiteindelijke te testen besturingskast.

De opdracht bestaat uit 5 subopdrachten namelijk:

- MODBUS RTU/ASCII simulatie over seriële poort,
- MODBUS RTU/ASCII simulatie over TCP,
- IO Configuratie,
- Uitbreiding simulatie mogelijkheden.

Deze subonderdelen zijn allemaal software toepassingen en komen allemaal op de bestaande pc te draaien. Dit heeft geresulteerd tot figuur 2.1.
Figuur 2.1 geeft een schematische weergave weer van de opdracht en hoe de software onderling met elkaar communiceert of communiceert met de hardware simulator/ besturingskast. Per onderdeel in de schematische weergave staat een hoofdstuk nummer. Dit hoofdstuk nummer verwijst naar het hoofdstuk waarin het onderdeel uitvoering wordt beschreven.



figuur 2.1. Schematische weergave project opdracht.

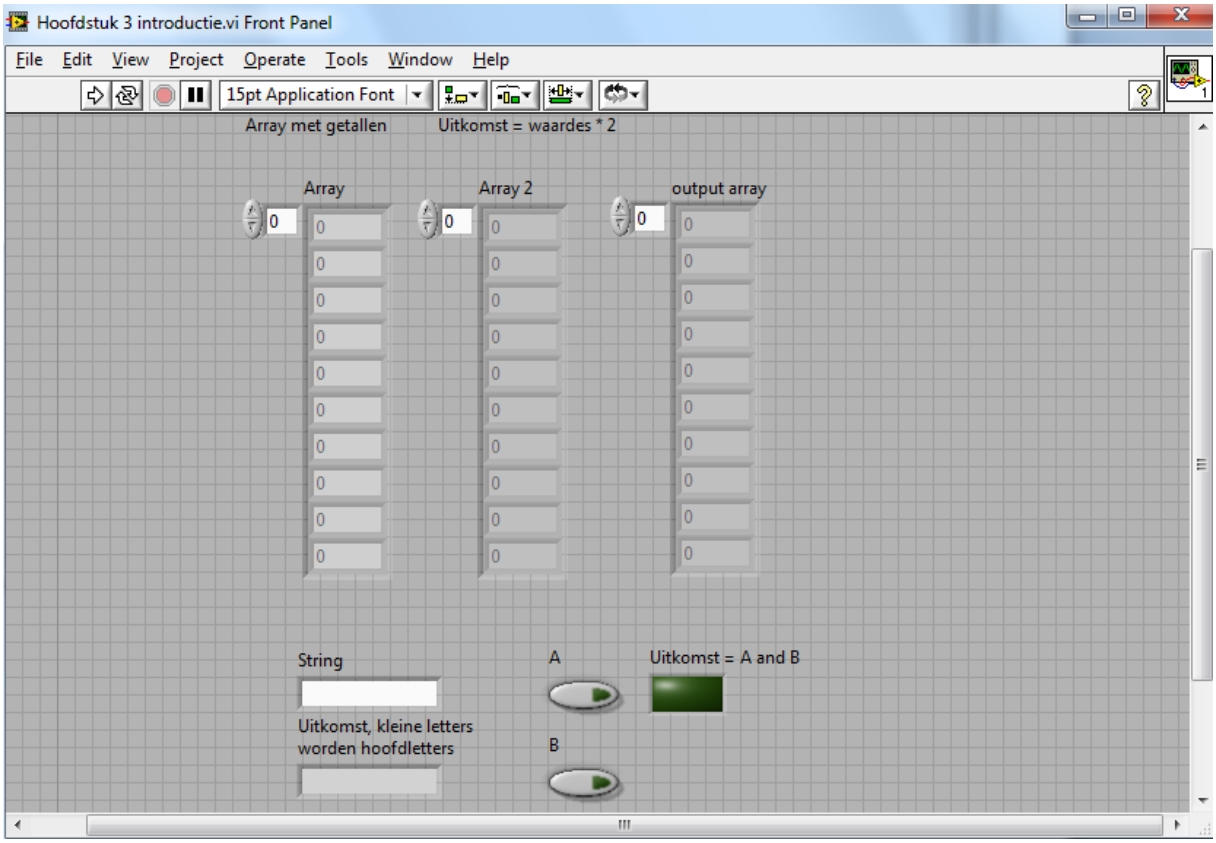
3. PC van de Hardware simulator

De Hardware simulator wordt via TCP aangestuurd door een pc. Op deze pc draait LabVIEW.

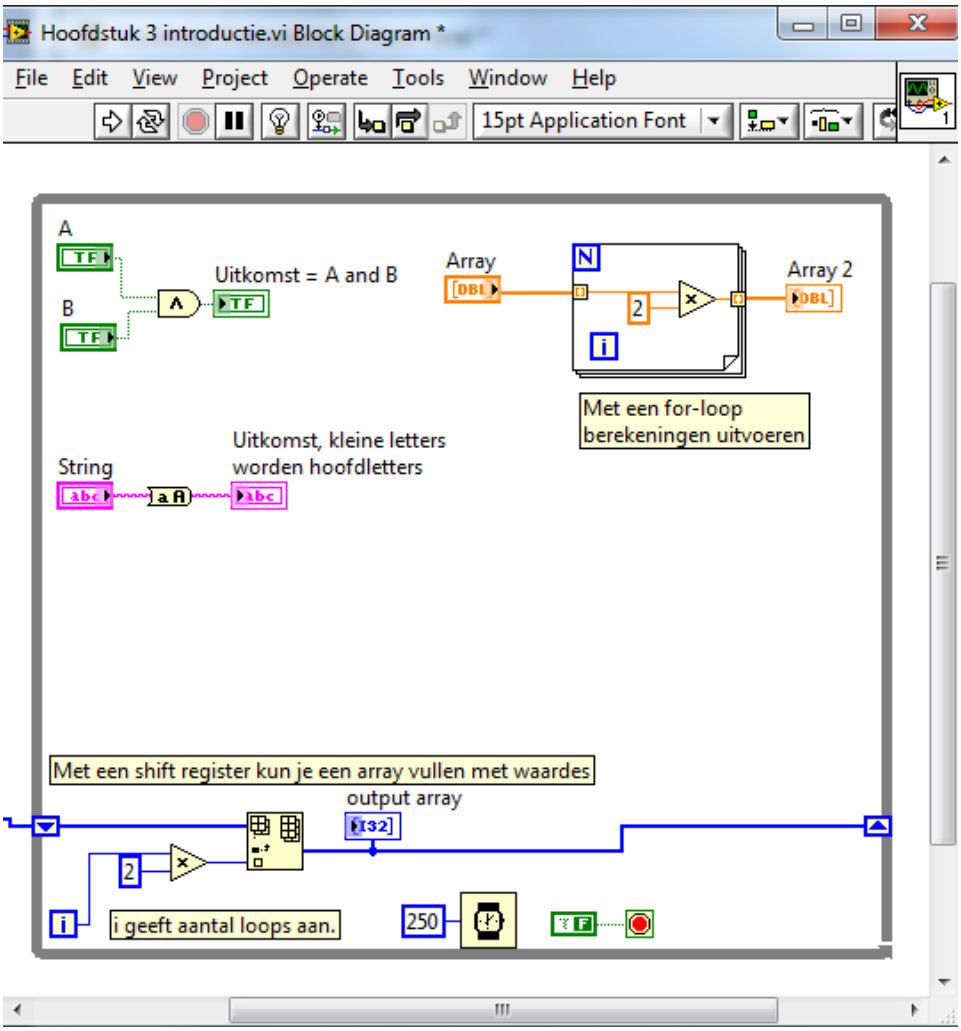
LabVIEW (Laboratory Virtual Instrumentation Engineering Wordbench) is een grafische programmeeromgeving, ontwikkeld door National Instruments. De grafische omgeving is voornamelijk geschikt voor de besturingstechniek en dan met name, data-acquisitie en het communiceren met meet instrumenten.
De LabVIEW programmeertaal omvat de zelfde taalconstructies en datastructuren als traditionele programmeertalen waardoor LabVIEW gebruikt kan worden voor brede software toepassingen.
LabVIEW heeft overeenkomsten met C++, Digitale Techniek en de wiskunde.
LabVIEW kan zowel onder Windows, Linux als MacOS X worden gebruikt.

Een LabVIEW applicatie bestaat uit 2 verschillende vensters,

- een VI (Virtueel Instrument) hierin worden de bedieningsorganen en indicatoren geplaatst zoals knoppen, lampen, grafieken,
  Figuur 3.1 laat een LabVIEW VI zien.
- een Block Diagram, hierin staat de broncode,
  Figuur 3.2 laat een LabVIEW Block Diagram zien.



Figuur 3.1. Een LabVIEW VI.



Figuur 3.2. LabVIEW Blok Diagram.

Een LabVIEW programma wordt volledig grafisch gemaakt. Functies en subroutines worden aangegeven als blokken. Deze bolken zijn onderling verbonden d.m.v. gekleurde lijnen. De kleur van de lijn geeft aan om wat voor een datatype het gaat.

Oranje geeft aan dat het om een real data type gaat.
Blauw geeft aan dat het om een integer data type gaat.
Roze geeft aan dat het om een string data type gaat.
Groen geeft aan dat het om een booleaans type gaat.

De dikte van een lijn varieert.
Een dunne lijn is een data verbinding tussen verschillende blokken.
Een dikke lijn stelt een eendimensionale array voor.
Een dubbel dikke lijn stelt een multidimensionale array voor.
For loops, While loops, Case statements worden met rechthoeken aangegeven. Alles wat binnen deze rechthoeken is geplaatst, wordt binnen deze rechthoeken uitgevoerd.

De verbindingslijnen die de functieblokken met elkaar verbinden dienen voor twee zaken. Als eerste zorgen ze voor de gegevensoverdracht tussen de functieblokken, ten tweede zorgen de lijnen voor de volgorde waarin de functieblokken worden uitgevoerd.
De volgorde waarin de functieblokken zijn geplaatst is de volgorde waarin de functieblokken worden uitgevoerd. Een vertakking van de verbindingslijnen zorgt er voor dat meerdere functies tegelijk kunnen worden uitgevoerd. Dit wordt ook wel multi-treading genoemd.

Zoals te zien is in figuur 2.1 komen alle software programma's op één pc te staan. Momenteel staat op de pc de bestaande software van de hardware simulator, aan deze software wordt door de afstudeerder niks veranderd. Mocht er veranderingen noodzakelijk zijn in de bestaande software dan worden deze aanpassingen gedaan door de afstudeerbegeleider. De nieuwe software programma's worden op de pc "geïnstalleerd" en zullen onderling communiceren met de bestaande software.

4. MODBUS simulatie

Er kan zich een situatie voordoen dat er bij een automatiseringsproject een onderdeel bevindt die wordt aangestuurd door het MODBUS protocol.
Om deze verbinding te kunnen testen zal er een hardware component moeten worden besteld waarmee de verbinding kan worden getest. Dit is een omslachtige manier van testen en kost veel tijd en geld.

Om efficiënter en goedkoper een MODBUS verbinding te kunnen testen moet er een simulatie programma worden gemaakt die de functies van het hardware component kan over nemen. De simulatie programma vervangt dus het hardware component.

Om een goed simulatie programma te kunnen schrijven moet er eerst goed worden ingelezen in de theorie van het MODBUS protocol.
Zodra het MODBUS protocol bekend is, kan het programma worden uitgedacht. Hierbij moet goed worden nagegaan wat het programma moet gaan doen en vooral wat voor stappen het programma moet gaan doorlopen.

4.1 Theorie MODBUS

MODBUS is een seriële communicatie protocol ontworpen door het bedrijf Modicon in 1979.
Het protocol werd ontworpen om te kunnen communiceren met de plc's van het bedrijf.

In de loop der jaren is het MODUBS protocol uit gegroeid tot een standaard protocol voor de industrie en het meest gebruikte protocol om verschillende hardware componenten met elkaar te verbinden.

De reden voor het veelvuldig gebruik van het MODBUS protocol is:

- Het is een open en gratis protocol,
- Makkelijk implementatie in industrie,
- Het verzend alleen data, geen extra informatie er omheen.

Het MODBUS protocol werkt volgens het Master Slave principe. Dit houdt in dat de Master een bericht verstuurd over de bus naar een bepaalde Slave bijvoorbeeld Slave 1.
Alle Slave's ontvangen dit bericht en alleen de Slave met het adres 1 mag op het bericht reageren.
Slave's kunnen alleen reageren op een bericht dat van de Master afkomstig is en als het bericht voor hun bestemd is.
Slave's kunnen onderling geen berichten versturen.

MODBUS wordt vaak gebruikt voor opvragen van waarden van bijvoorbeeld regelkleppen, frequentie regelaars maar ook voor het aansturen van kleppen, motoren en pompen.

4.1.1 Protocol versies

Het MODBUS protocol werkt op de fysieke lagen Seriële poort, Ethernet en andere netwerken dat het Internet Protocol ondersteunen.

Het MODBUS protocol kent verschillende versies namelijk:

- *MODBUS RTU:* RTU (Remote Terminal Unit) Dit protocol wordt gebruikt over een seriële verbinding en maakt gebruik van een binaire code. Het RTU bericht bestaat uit commando's/ data gevolgd door een CRC check, ook wel Cyclic Rendundancy Check. Deze CRC check is er voor om te na te gaan of het hele bericht goed is over gekomen.
  Het MODBUS RTU protocol is het meest gebruikte MODBUS protocol. Het MODBUS RTU bericht moet aan één gesloten worden verstuurd.
- *MODBUS ASCII:* ASCII (American Standard Code for Information Interchance) Dit protocol maakt gebruik van seriële verbinding en werkt met ASCII karakters. Het ASCII protocol formaat bestaat uit een startbit teken (:), daarna het data bericht met een longitudinal rendunancy check (lrc) gevolgd door de karakters CR/LF.
- *MODBUS TCP/IP:* Dit is een MODBUS protocol dat werkt op TCP/ IP netwerken. Dit protocol maakt geen gebruik van een CRC/lrc berekening omdat de lagen hoger in het OSI model dit voor hun rekening nemen.
- *MODBUS over TCP/IP:* Dit is een MODBUS RTU bericht dat wordt verzonden over het TCP/ IP netwerk. Hierbij wordt wel een CRC berekening gemaakt omdat dat het een RTU bericht is.
- *MODBUS PLUS (MODBUS+ of MU+):* Dit is een uitgebreide versie van MODBUS, ontwikkeld en in eigendom van Schneider Electric. Om dit protocol te kunnen uitvoeren is er een speciale co-processor nodig die een snelle HDLC token rotatie kan verwerken. MODBUS Plus maakt gebruik van een 1 Mbit/s twisted pair verbinding en heeft op elke knooppunt in het netwerk een geïsoleerde transformator. Deze transformator zorgt voor transistion/edge triggering in plaats van voltage/ level triggering.
  Speciale interfaces zijn nodig voor om MODBUS Plus te kunnen koppelen aan een pc. Hiervoor zijn de speciale kaarten ISA (SA85), PCI en PCMCIA gemaakt.

De MODBUS protocollen die voor dit project worden gebruikt zijn:

- MODBUS RTU,
- MODBUS ASCII,
- MODBUS RTU versturen over TCP.

Op deze protocollen wordt daarom dieper in gegaan. De uitwerking van de MODBUS protocollen zijn beschreven in bijlage B MODBUS.

Omdat de functiecodes een belangrijk onderdeel van de opdracht zijn, zijn deze uitgewerkt op de volgende pagina.

Uitwerking functiecodes

De functiecodes die worden gebruikt voor de MODBUS simulatie zijn functiecodes 1, 3, 15 en 16. Functie code 5 en 6 worden niet gebruikt, omdat met de functiecodes 15 en 16 ook single registers en coils kunnen worden weg geschreven.

Om een beeld te krijgen hoe een RTU MODBUS bericht is opgebouwd wordt hieronder een paar voorbeelden uitgewerkt.

Functiecode 1:

De Master wil de status van de coils 20 t/m 56 weten van slave 17.
Het slave adres is dan 17, in hexadecimaal 11_{16}.
De functiecode is 1_{16}.
Het start adres wordt dan 20-1 = 19, het talstelsel begint vanaf 0. Dit wordt in hexadecimaal 13_{16}.
Het aantal coils van 20 t/m 56 is dan 37, in hexadecimaal 25_{16}.
De CRC wordt berekend over al deze waarden en wordt dan $0E84_{16}$.

Het bericht wat de Master verstuurd wordt dan:

11 01 0013 0025 $0E84_{16}$.

Een response van de slave kan zijn:

11 01 05 CD6BB20E1B $45E6_{16}$ waarbij:

11_{16}: is het slaven adres in decimaal 17.
01_{16} : is de functie code 1.
05_{16}: is het aantal data bits (37 coils / 8 databits is 4.6, er zijn dus 5 bytes nodig).
CD_{16}: Coils 27 - 20 (1100 1101).
$6B_{16}$: Coils 35 - 28 (0110 1011).
$B2_{16}$: Coils 43 - 36 (1011 0010).
$0E_{16}$: Coils 51 - 44 (0000 1110).
$1B_{16}$: 3 spaceholders en de Coils 56 t/m 52 (0001 1011).
45E6: De CRC check.

Functiecode 3

De Master wil de status van de registers 40108 t/m 40110 weten van slave 17.
Het slave adres is dan 17, in hexadecimaal 11_{16}.
De functiecode is 3_{16}.
Het start adres wordt dan 40108 – 40110 = 107, in hexadecimaal is dat $6B_{16}$.
Het aantal coils van 40108 t/m 40110 is dan 3, in hexadecimaal 3_{16}.
De CRC wordt berekend over al deze waarden en wordt dan 7687_{16}.

Het bericht wat de Master verstuurd wordt dan:

11 03 006B 0003 7687_{16}.

Een response van de slave kan zijn:

11 03 06 AE41 5652 4340 $49AD_{16}$ waarbij:

11_{16}: is het slaven adres in decimaal 17.
03_{16} : is de functie code 3.
06_{16}: is het aantal data bits (3 registers van elk 2 bytes groot, dus 6 bytes).
$AE41_{16}$: De waarde of Register 40108.
5652_{16}: De waarde of Register 40109.
4340_{16}: De waarde of Register 40110.
$0E_{16}$: Coils 51 t/m 44 (0000 1110).
$1B_{16}$: 3 spaceholders en de Coils 56 t/m 52 (0001 1011).
45E6: De CRC check.

Functiecode 15

De Master wil data wegschrijven naar de Coils 20 t/m 29 van Slave 17
De functiecode is 15, in hexadecimaal $0F_{16}$.
Het start adres wordt dan 20-1 = 19, het talstelsel begint vanaf 0. Dit wordt in hexadecimaal 13_{16}.
Het aantal coils van 20 t/m 29 is dan 10, in hexadecimaal A_{16}.
Het aantal databytes wordt dan 10 Coils /8 = 1.25, is 2 bytes. 2_{16}.
De Coils 27-20 ($1100\ 1101_2$) wordt dan CD_{16}.
De Coils 29-28 met de rest gevuld met space holders ($0000\ 0001_2$) wordt dan 01_{16}.
De CRC wordt berekend over al deze waarden en wordt dan $BF0B_{16}$.

Het bericht wat de Master verstuurd wordt dan:

11 0F 0013000A 02 $CD01_{16}$.

Een response van de slave kan zijn:

11 0F 0013 00A 2699_{16} waarbij:

11_{16}: is het slaven adres in decimaal 17.
$0F_{16}$: is de functie code 15.
0013_{16}: Het data adres van de eerste Coil.
02_{16}: is het aantal databytes (10 coils / 8 databits is 1.25, er zijn dus 2 bytes nodig).
2699: De CRC check.

Functiecode 16

De Master wil data wegschrijven naar de registers 40002 & 40003 van Slave 17
De functiecode is 16, in hexadecimaal 10_{16}.
Het start adres wordt dan 40002-1 = 1_{16}.
Het aantal registers is 2_{16}.
Het aantal databytes wordt dan 2 registers van elk 2 bites, 4_{16}.
De waarde voor register 40002 kan zijn $000A_{16}$.
De waarde voor register 40003 kan zijn 0102_{16}.
De CRC wordt berekend over al deze waarden en wordt dan $C6F0_{16}$.

Het bericht wat de Master verstuurd wordt dan:

11 10 0001 0002 04 000A 0102 $C6F0_{16}$.

Een response van de slave kan zijn:

11 10 0001 0002 1298_{16} waarbij:

11_{16}: is het slaven adres in decimaal 17.
10_{16} : is de functie code 16.
0001_{16}: Het adres van het eerste register.
0002_{16}: Het aantal registers dat wordt weg geschreven.
1298: De CRC check.

4.2 Ontwerp MODBUS simulatie

Aan de hand van de theorie kunnen de MODBUS simulaties worden gemaakt.
Om tot een goed software programma te komen moet eerst worden uitgedacht/ ontworpen wat het programma precies moet doen, welke stappen het programma moet doorlopen, welke actie daar mee gepaard gaan.
Om goed gestructureerd te werk gaan met het ontwerpen van een programma wordt er ontworpen via een flowchart. Dit houdt in dat er op papier via een activiteiten diagram een globale werking van het programma wordt gemaakt. In deze activiteiten diagram staan de stappen die het programma doorloopt, welke stappen eventueel parallel lopen en welke acties bij elke stap worden uitgevoerd.

Per simulatie onderdeel wordt bekeken wat de simulatie moet gaan doen, aan welke eisen het simulatie programma moet voldoen en welke stappen de simulatie moet doorlopen.
De simulaties worden in de volgende 4 subparagrafen behandelt.

4.2.1 MODBUS RTU/ ASCII Master simulatie

De MODBUS Master simulatie moet aan de volgende eisen doen:

- Om een simulatie te krijgen die breed toepasbaar is het wenselijk om de functiecodes 1, 3, 15 en 16 in de software te integreren. Er zijn voor deze functiecodes gekozen omdat deze het meest worden toegepast binnen Imtech Vonk.
- Ook is het wenselijk om zowel het RTU protocol als het ASCII protocol te kunnen gebruiken.

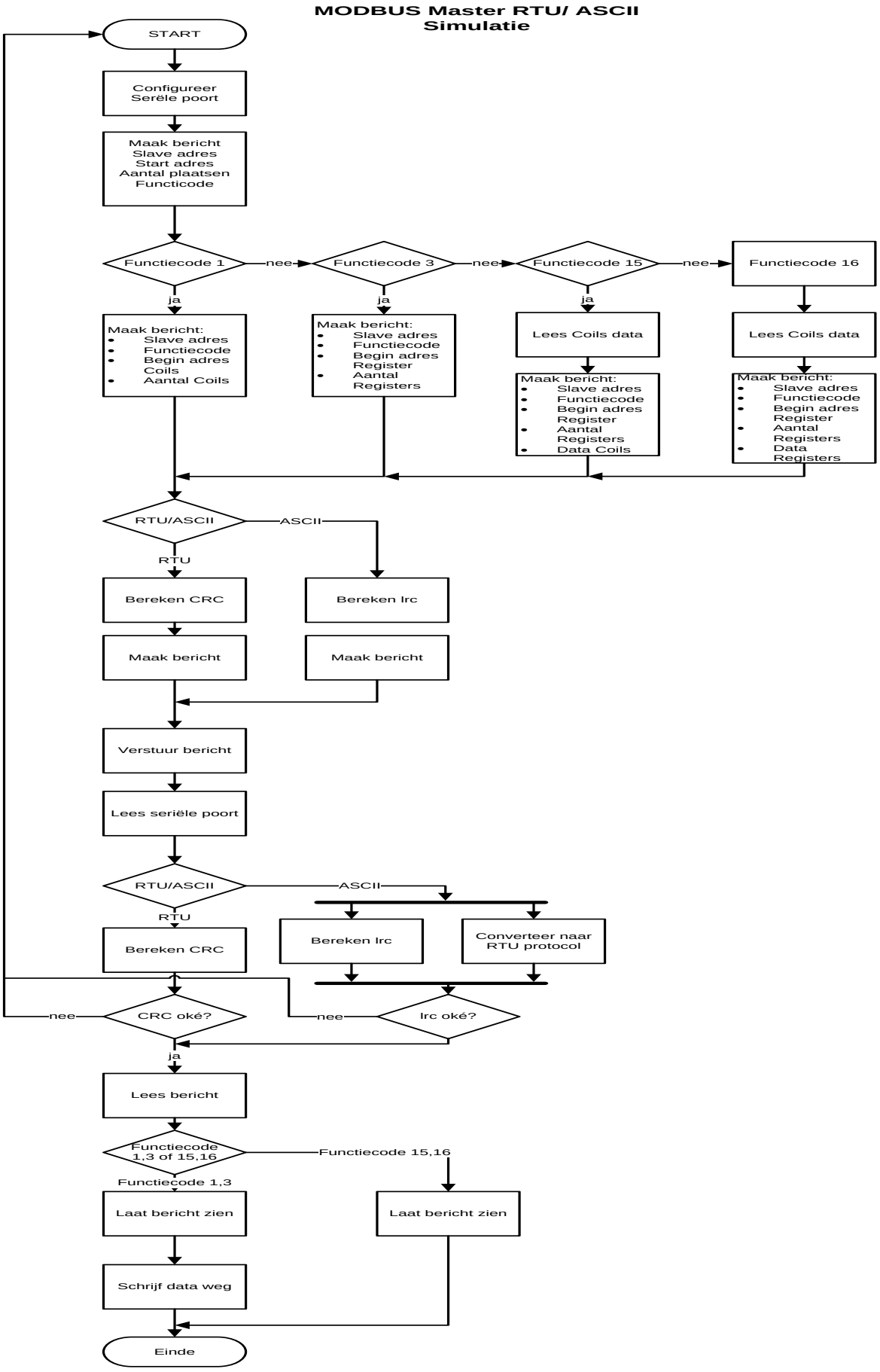
De flowchart van de MODBUS Master Simulatie is te zien in figuur 4.1.

4.2.2 MODBUS RTU/ ASCII Slave Simulatie

De MODBUS Slave simulatie moet aan de volgende eisen doen:

- Om een simulatie te krijgen die breed toepasbaar is het wenselijk om de functiecodes 1, 3, 15 en 16 in de software te integreren. Er zijn voor deze functiecodes gekozen omdat deze het meest worden toegepast binnen Imtech Vonk.
- Ook is het wenselijk om zowel het RTU protocol als het ASCII protocol te kunnen gebruiken.

De flowchart is te zien in figuur 4.2.



figuur 4.1. Flowchart MODBUS Master Simulatie.

figuur 4.2. flowchart MODBUS Slave.

4.2.3 MODBUS RTU over TCP Master

De MODBUS RTU over TCP Master verschilt niet veel van MODBUS RTU/ ASCII Master.

De verschillen zijn:

- MODBUS RTU over TCP werkt alleen met het RTU protocol,
- Het bericht wordt verstuurd via een Ethernet poort over het TCP protocol.

Het programma zal dan ook niet veel verschillen van de andere Master programma. De functieblokken: converteren naar ASCII, ASCII naar RTU, en de lrc berekening zullen ontbreken omdat deze niet nodig is. Omdat de verschillen tussen RTU/ASCII en RTU over TCP zeer klein zijn zal de flowchart van MODBUS RTU over TCP te zien zijn in de bijlage.

De flowchart van de MODBUS RTU over TCP Master is te zien de bijlage B.

4.2.4 MODBUS RTU over TCP Slave

De MODBUS Slave simulatie moet aan de volgende eisen doen:

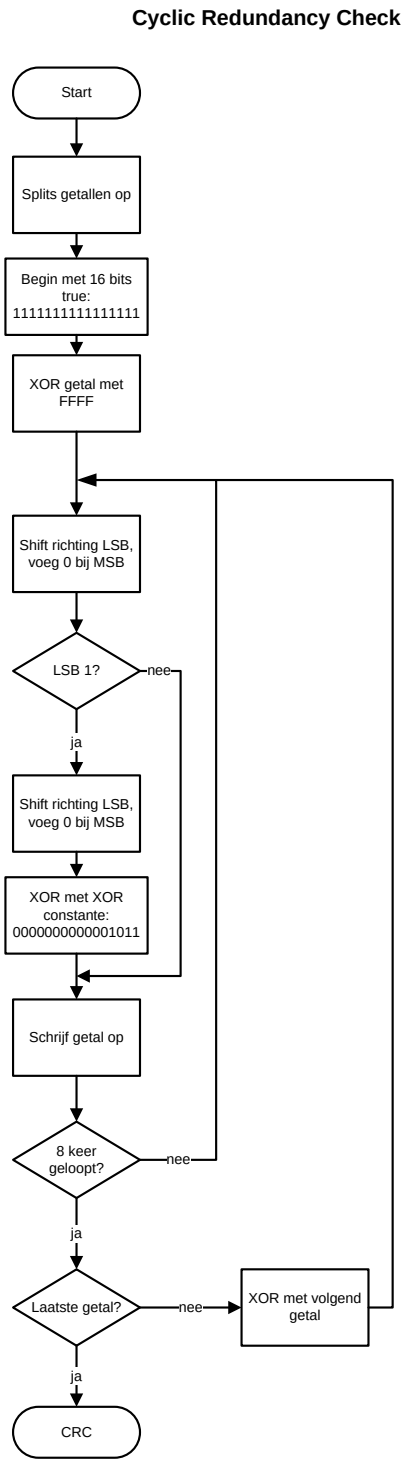
- Om een simulatie te krijgen die breed toepasbaar is het wenselijk om de functiecodes 1, 3, 15 en 16 in de software te integreren.
- De Slave moet communiceren via een Ethernet poort over het TCP protocol.

Het programma wordt ontworpen via een flowchart. Aan de hand van een flowchart wordt het programma geschreven. Onder het schrijven kan het voorkomen dat bepaalde functies nog moeten worden gemaakt of worden aangepast. Omdat de verschillen tussen RTU/ASCII en RTU over TCP zeer klein zijn zal de flowchart van MODBUS RTU over TCP te zien zijn in de bijlage.
De flowchart is te zien in de bijlage B.

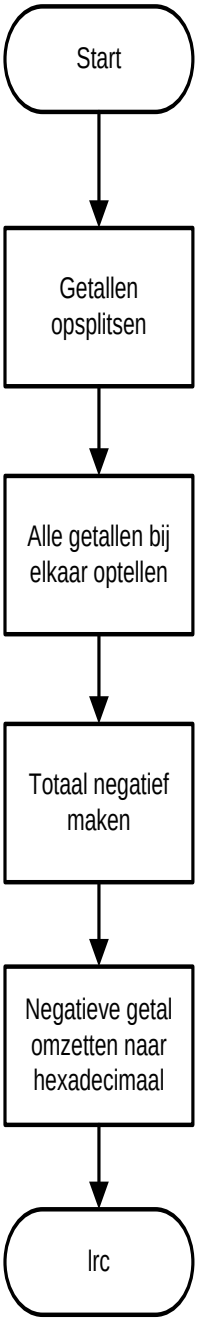
Belangrijke functies die vaak terugkomen in de simulaties zijn:

- CRC check,
- Lrc check.

Daarom is van deze functies een Koninklijke weg gemaakt.
In figuur 4.3 is de flowchart van de CRC te zien.
In figuur 4.4 is de flowchart van de lrc te zien.



figuur 4.3. Flowchart CRC.



Figuur 4.4. Flowchart lrc.

Een flowchart is een globale benadering voor het schrijven voor een software programma. Vanuit de flowchart zal het software programma worden geschreven. Onder het schrijven van het software programma zullen er nog aanpassingen worden gedaan of bepaalde functies zullen nog verder moeten worden uitgedacht.

Vanuit de flowchart zijn de software simulaties gemaakt. In hoofdstuk 4.3 worden deze software simulaties behandeld.

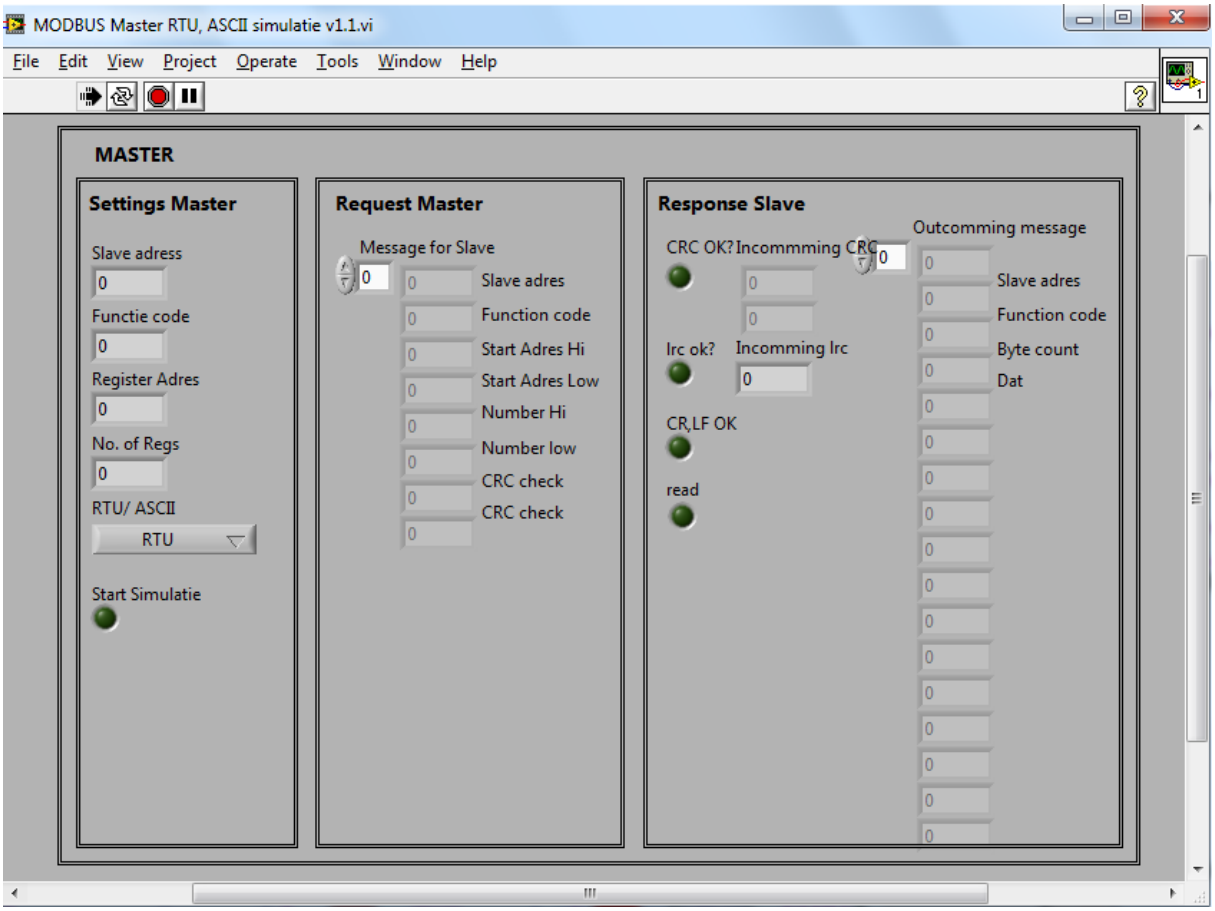
4.3 Uitwerking MODBUS simulaties

In deze paragraaf worden de uitwerkingen van de simulaties behandeld. Per simulatie wordt de VI behandeld, waarom er voor een bepaalde interface is gekozen en wat er per interface te zien is.

Zoals in figuur 2.1 te zien is, zullen de software toepassingen op de zelfde pc draaien als de bestaande software. De MODBUS simulatie zal door de bestaande software op worden geroepen en ook weer afgesloten. Om dit te kunnen realiseren wordt de MODBUS simulatie aangestuurde door globale variabelen. Globale variabelen houdt in dat een bepaald software programma data wegschrijft in variabelen en dat een ander software deze data weer kan gebruiken.

4.3.1 MODBUS RTU/ASCII Master

Doordat de MODBUS Masters simulatie op de achtergrond zal draaien, zullen niet alle gegevens worden weergegeven. Er is daarom gekozen voor een minimale weergave van gegevens. Alleen de hoognodige gegevens zullen te zien zijn op de VI. Het VI van de MODBUS simulatie is in figuur 4.5 te zien.

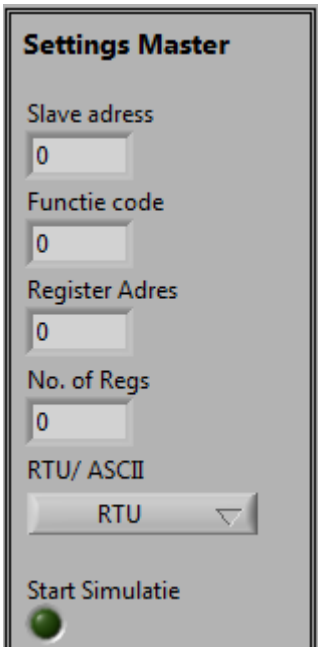


Figuur 4.5. VI MODBUS Master VI..

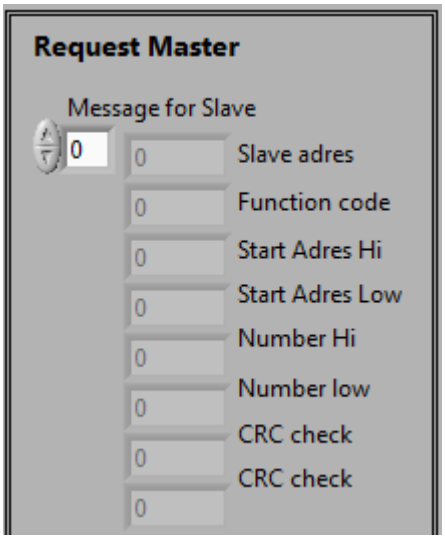
In figuur 4.6 is de eerste kolom van MODBUS Master VI. Er is gekozen om het Slave adres te laten zien, de functiecode, register adres, aantal registers, welk protocol er wordt gebruikt en of de simulatie is gestart.
Er is voor deze gegevens gekozen om de volgende redenen:

- Dit zijn de belangrijkste gegevens om de laten zien, hier draait de MODBUS simulatie om,
- Mocht zich problemen voordoen met oog op een niet werkende simulatie, dan is het handig om deze gegevens paraat te hebben voor het zoeken naar storingen.

In figuur 4.7 is de tweede kolom van de MODBUS Master VI te zien. Er is gekozen om het bericht naar de Slave weer te geven. Dit omdat er dan in één oog opzicht te zien is welk bericht er naar de slave wordt gestuurd.



figuur 4.6. Status MODBUS Master simulatie.



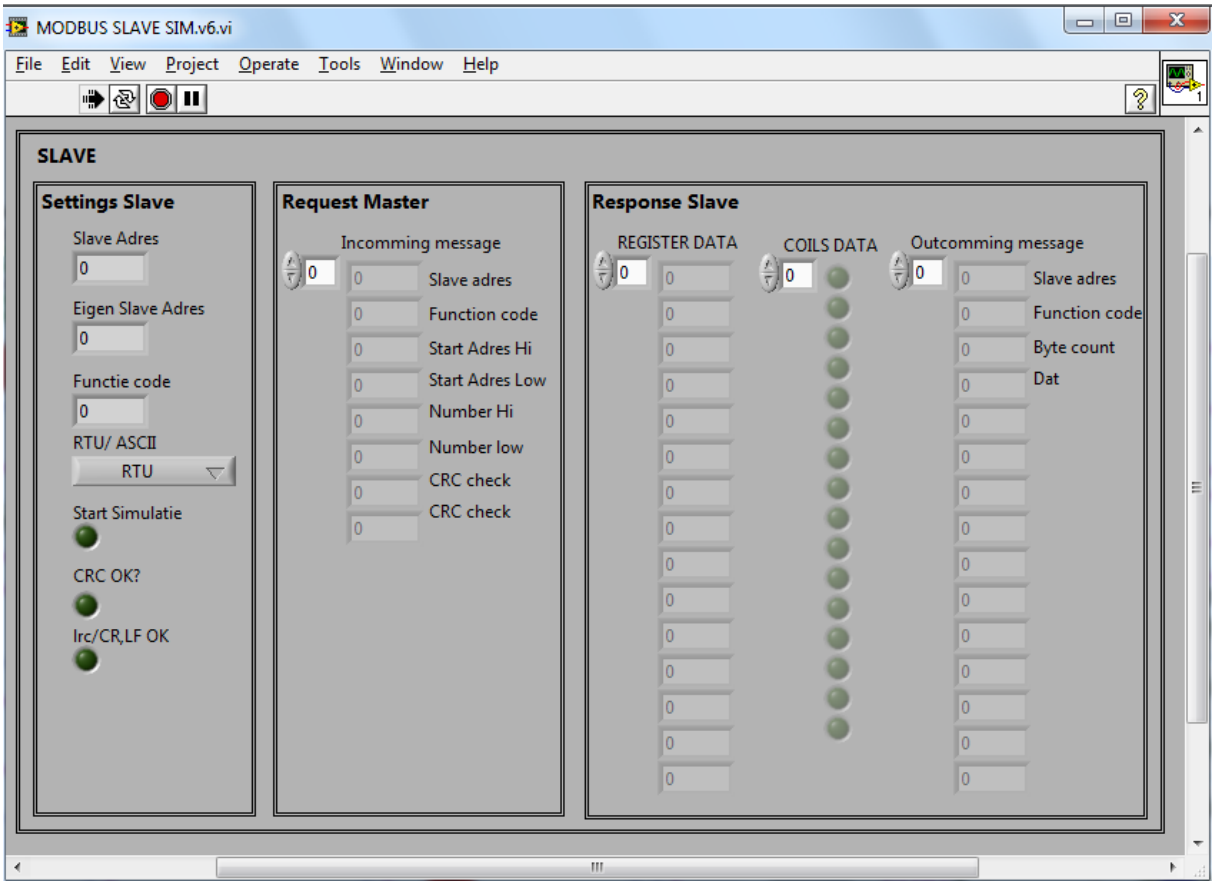
figuur 4.7. Bericht voor Slave van MODBUS Master simulatie.

In figuur 4.8 is de derde kolom van de MODBUS Master VI te zien. Er is er voor gekozen het inkomende bericht te laten zien, dit is essentieel voor de simulatie.
maar ook de inkomende CRC en lrc zijn belangrijk. Mocht het programma niet werken, dan zijn dit de eerste onderdelen waarnaar wordt gekeken.

figuur 4.8. Respons Slave.

4.3.2 MODBUS RTU/ ASCII Slave

Doordat de MODBUS Slave simulatie op de achtergrond zal draaien, zullen niet alle gegevens worden weergegeven. Het VI van de MODBUS simulatie is in figuur 4.9 te zien.

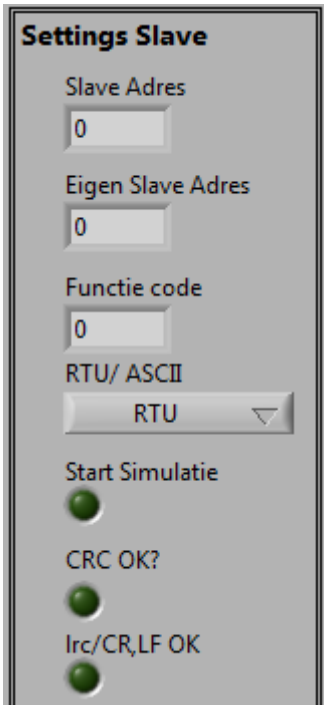


figuur 4.9. VI MODBUS RTU/ ASCII Slave

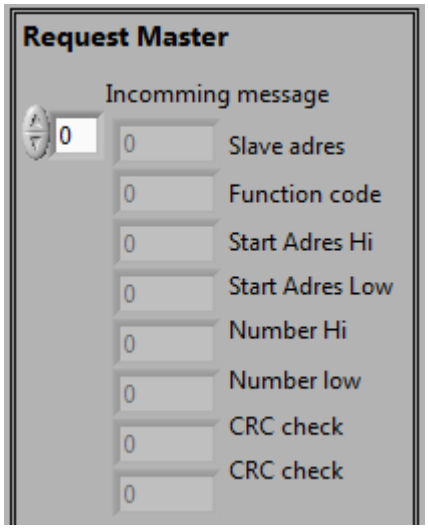
In figuur 4.10 is de eerste kolom te zien van de Slave. Er is gekozen om het binnenkomende Slave adres te laten zien, het eigen Slave adres te laten zien, welke functiecode er wordt gebruikt, welk protocol er wordt gebruikt, of de simulatie is gestart en of de CRC of lrc goed zijn.
Er is gekozen voor deze lay-out om de volgende redenen:

- Dit zijn de belangrijkste gegevens om de laten zien,
- Voor storing zoeken is het belang om deze gegevens in één oogopslag te kunnen zien.

In figuur 4.11 is de tweede kolom van de Slave te zien. In deze kolom staat het inkomend bericht van de Master. Er is gekozen voor het inkomende bericht zodat men kan zien wat er aan de Slave wordt gevraagd.

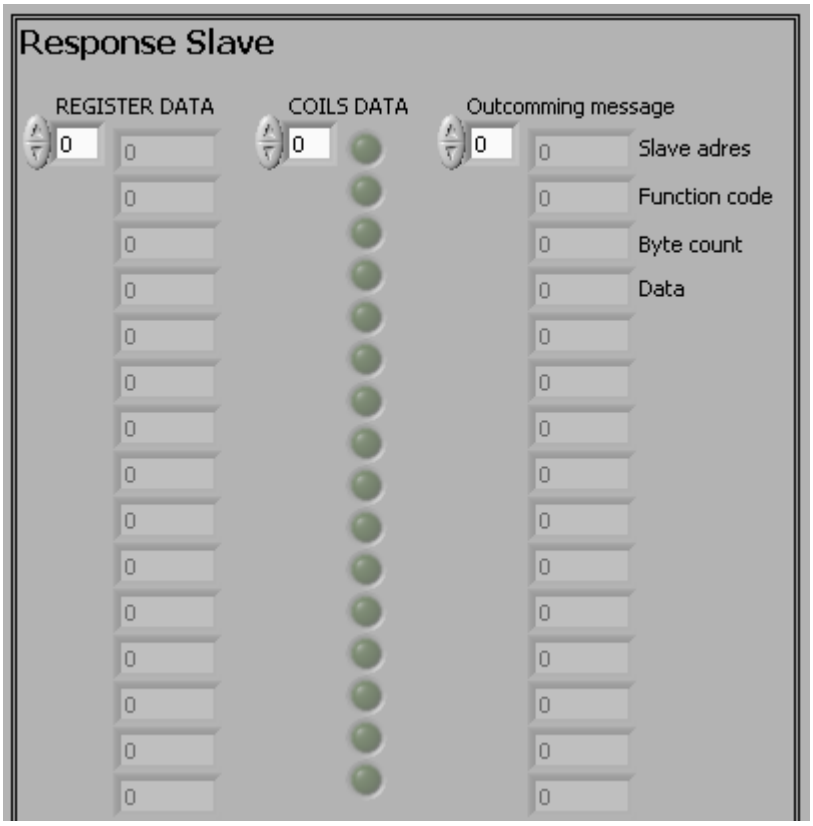


figuur 4.10. Slave settings.



figuur 4.11. Inkomen bericht Master.

De laatste kolom is te zien in figuur 4.12. In deze kolom wordt de data weergegeven die de Slave heeft. Dit kunnen coils data of register data zijn. Tevens is het uitgaande bericht te zien met de betekenis van elk getal erbij. Het is van belang dat men weet wat de data van de Slave is en welk bericht er naar de Master gaat. Daarom is er gekozen voor deze lay-out.



figuur 4.12. Uitgaand bericht Slave.

4.3.3 MODBUS RTU over TCP Master

De MODBUS RTU over TCP Master ziet nagenoeg het zelfde uit als de MODBUS RTU/ ASCII over RS232. Er zijn een paar verschillen tussen de MODBUS RTU over TCP Master en MODBUS RTU/ ASCII over RS232.

- Bij de MODBUS TCP wordt alleen gebruik gemaakt van het RTU protocol,
- Er is bij MODBUS TCP geen Slave adres nodig omdat er gebruik wordt gemaakt van IP adres.

Omdat de verschillen klein zijn en de VI het zelfde eruit zien worden de VI's niet behandeld.

4.3.5 MODBUS RTU over TCP Slave

De MODBUS RTU over TCP Slave ziet nagenoeg het zelfde uit als de MODBUS RTU/ ASCII over RS232. Er zijn een paar verschillen tussen de MODBUS RTU over TCP Slave en MODBUS RTU/ ASCII over RS232.

- Bij de MODBUS TCP wordt alleen gebruik gemaakt van het RTU protocol,
- Er is bij MODBUS TCP geen Slave adres nodig omdat er gebruik wordt gemaakt van IP adres.

Omdat de verschillen klein zijn en de VI het zelfde eruit zien worden de VI's niet behandeld.

4.3.6 Beschrijving broncode simulaties

Het volledige programma is te zien in de bijlagen.
Nu wordt er beperkt tot specifieke onderdelen van het programma.

Functie blokken.

Door het hele programma komen zogenaamde functieblokken voor. Deze functie blokken zijn op zichzelf staande VI's de een bepaalde functie vervullen.
De reden voor het kiezen van een functie blok kan verschillende achtergronden hebben, namelijk:

- Een berekening of een stuk code komt meerdere keren voor in de software. Deze code kan dan steeds weer op nieuw worden gemaakt, maar het is overzichtelijker om deze code in een functieblok te plaatsen en deze meerdere keren aanroepen.
- Ruimte besparing. Om het programma overzichtelijk en leesbaar te houden is het wenselijk om ervoor te zorgen dat de broncode in 1 scherm (lees beeldscherm computer) weer te geven. Om dit te realiseren kunnen bepaalde onderdelen in functieblokken worden geplaatst.

Omdat zo goed als alle functieblokken zowel in de: Master als in de Slave, in RTU/ ASCII protocol als in de RTU over TCP protocol voorkomen, worden alle functieblokken behandeld.
Per functieblok wordt:

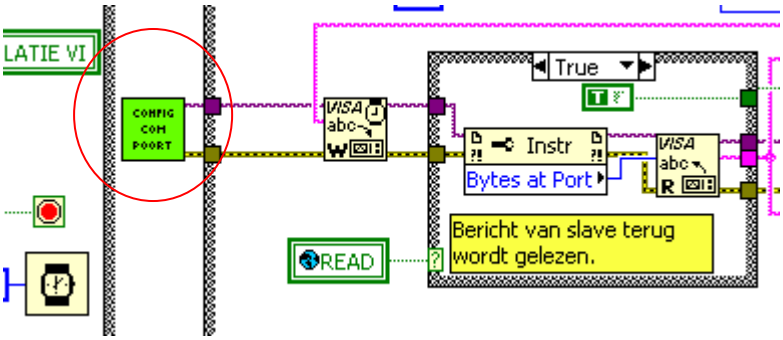
- de titel genoemd,
- globale uitleg beschreven en waarom er voor een functieblok is gekozen,
- het figuur waar de functieblok in staat getoond (wordt met een rode cirkel aangetond),
- werking van de functieblok beschreven,
- broncode vertoond.

Configuratie com-poort.

In figuur 3.11 is de functieblok "CONFIG COM POORT" te zien. Dit functieblok zorgt ervoor dat de seriële com-poort wordt geconfigureerd. In dit blok worden de volgende instellingen ingesteld:

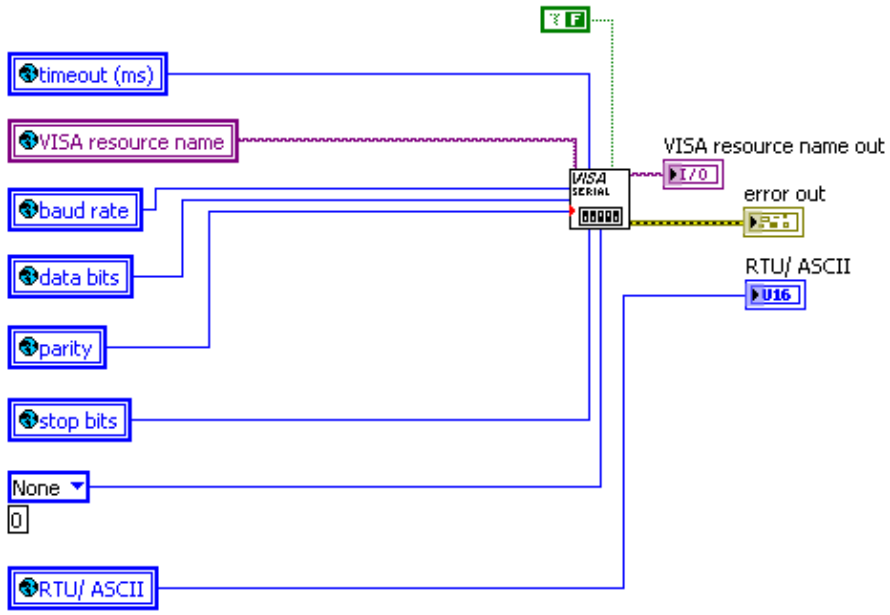
- Timeout (ms),
- Visa resource name (com-poort),
- Baud rate,
- Data bits (RTU 8, ASCII 7),
- Paraity,
- Stop bits,
- RTU of ASCII.

De reden om de com-poort configuratie in een functieblok om te zetten is omdat de com-poort instellingen voor zowel de Master als de Slave moet worden gedaan en omdat dit ruimte bespaart.
In figuur 4.13 is de code te zien die anders in de Block diagram van de MODBUS Master simulatie moest worden geplaatst.



figuur. 4.13 Com-poort configuratie.

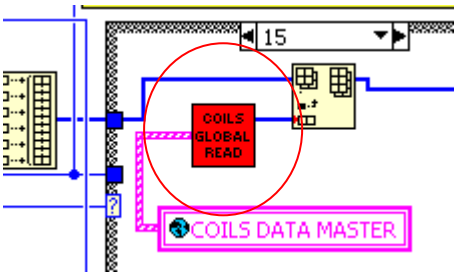
De werking is als volgt, de com-poort settings worden door de bestaande software in globale variabelen geladen. Deze globale variabelen worden ingelezen in de functieblok. Met deze com-poort settings word dan de com-poort geconfigureerd. De broncode is te zien in figuur 4.14.



figuur 4.14. Code voor configuratie com-poort.

Coils data uitlezen.

In figuur 4.15 is de functieblok COILS GLOBAL READ te zien. Deze functieblok wordt gebruikt bij functiecode 15 (write Coils). Deze functieblok leest de globale variabele COILS DATA MASTER uit. In deze globale variabel staan een lijst met Tag-code, adres, en waarde. Deze lijst zal worden geschreven naar de slave, afhankelijke van het start adres en aantal plekken. Er is gekozen voor deze functieblok voor de ruimte besparing die het oplevert.



figuur 4.15. Coils Global Read.

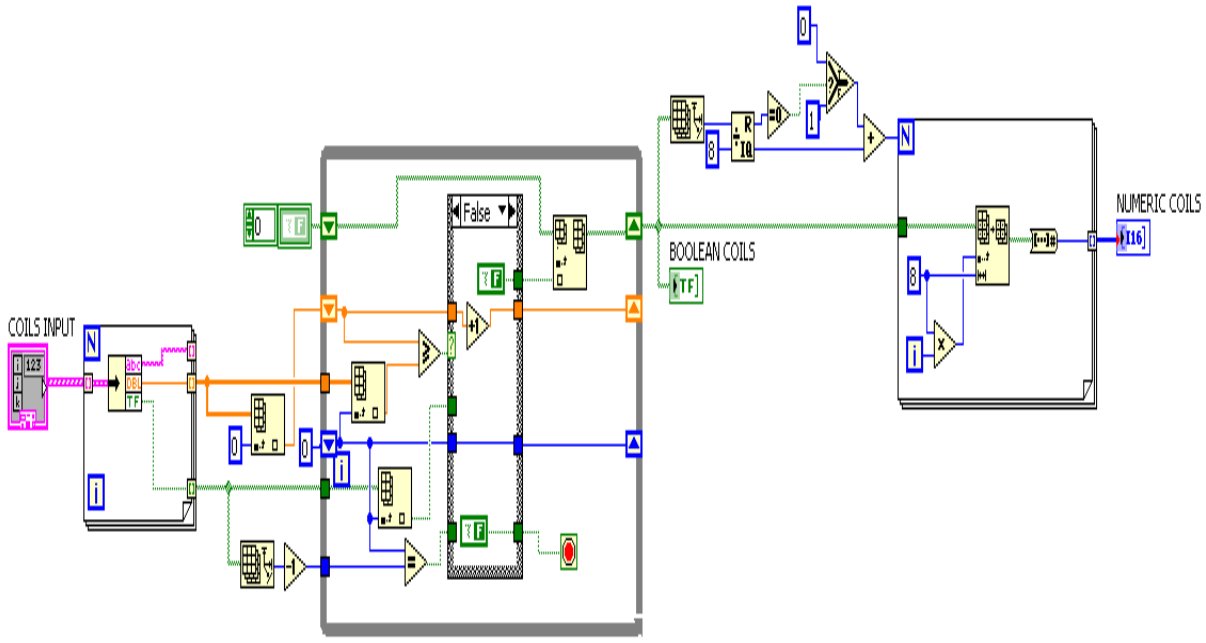
De waarden kunnen als volgt eruit zien, zie tabel 4.1.

Tag-code	Adres	waarde
11-GBS-2150	30001	TRUE
11-GBS-2151C	30002	FALSE
11-GBS-21530	30005	TRUE
11-XZA-2022	30010	TRUE

tabel 4.1. Coils data lijst.

De adressen lopen op van 30001 t/m 30002, dan 30005 en als laatste 30010. Zoals te zien is zijn de adressen 30003, 30004, 30006, 30007, 30008 en 30009 niet ingevuld. Er is ervoor gekozen om de ontbrekende adressen op te vullen met FALSE zodat er een aaneen lopende lijst van data ontstaat.

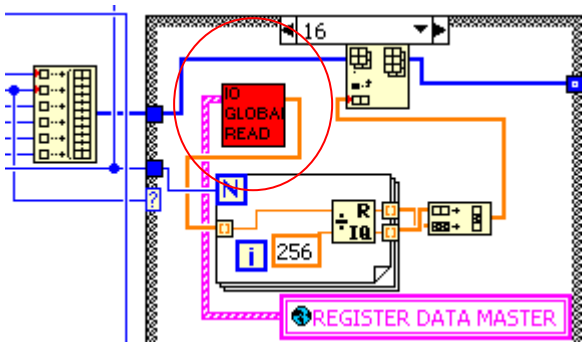
De functie COILS DATA MASTER zorgt hiervoor. In figuur 4.16 is de broncode van de functieblok te zien. De data wordt geladen. In de tabel wordt gekeken naar de adressen, alle adressen met de waardes worden geladen.
Mocht er een adres niet aanwezig zijn, dan wordt deze plek opgevuld de waarde FALSE.



figuur 4.16. Broncode COILS DATA MASTER.

IO Global Read.

In figuur 4.17 is de IO GLOBAL READ te zien. Deze functie blok heeft functie als de COILS DATA MASTER, het uitlezen van data. Het verschil is alleen dat de IO GLOBAL READ de registers die naar de Slave moet worden geschreven moet uitlezen.



figuur 4.17. IO GLOBAL READ

Deze functieblok wordt gebruikt bij functiecode 16 (write Registers). Deze functieblok leest de globale variabele REGISTER DATA MASTER uit. In deze globale variabel staan een lijst met Tag-code, adres, en waarde. Deze lijst zal worden geschreven naar de Slave, afhankelijke van het start adres en aantal plekken.
De functieblok IO GLOBAL READ is gemaakt voor ruimte besparing en omdat het een ingewikkeld stuk code is die beter in een aparte functieblok kan worden geplaatst.

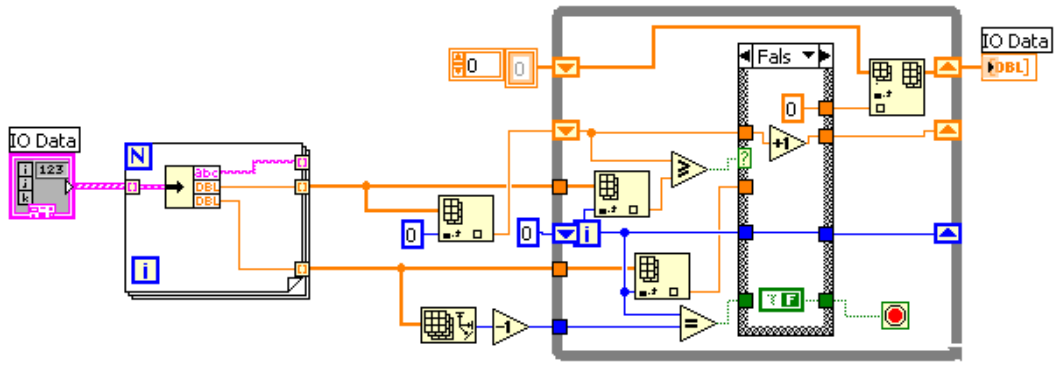
De waarden kunnen als volgt eruit zien, zie tabel 4.2.

Tag-code	Adres	waarde
11-SI-2078	40001	15
11-SI-2079	40002	200
11-PDIA-2166	40005	38
11-PDIA-2176	40010	85

tabel 3.2. Register data.

De adressen lopen op van 40001 t/m 40002, dan 40005 en als laatste 40010. Zoals te zien is zijn de adressen 40003, 40004, 40006, 40007, 40008 en 40009 niet ingevuld. Er is ervoor gekozen om de ontbrekende adressen op te vullen met 0 zodat er een aaneen lopende lijst van data ontstaat.
De lijst met alle waarden kan dan worden verstuurd naar de Slave.

In figuur 4.18 is de broncode van de functieblok te zien. De data wordt geladen. In de tabel wordt gekeken naar de adressen, alle adressen met de waardes worden geladen.
Mocht er een adres niet aanwezig zijn, dan wordt deze plek opgevuld de waarde 0.



figuur 4.18. Broncode IO GLOBAL READ.

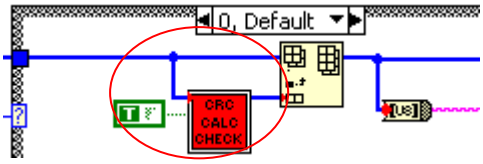
Elk bericht heeft een redundancy check. Aan de hand van deze check kan worden gecontroleerd of het bericht compleet is. Voor het RTU en ASCII protocol zijn verschillende redundancy check.
Het RTU protocol heeft de CRC check en de ASCII protocol heeft de lrc check.

CRC Calc

Bij het RTU protocol moet een CRC worden berekend of gecontroleerd. Deze CRC wordt berekend aan de hand van het te verzenden bericht en wordt na berekening aan het einde van het bericht toe gevoegd.

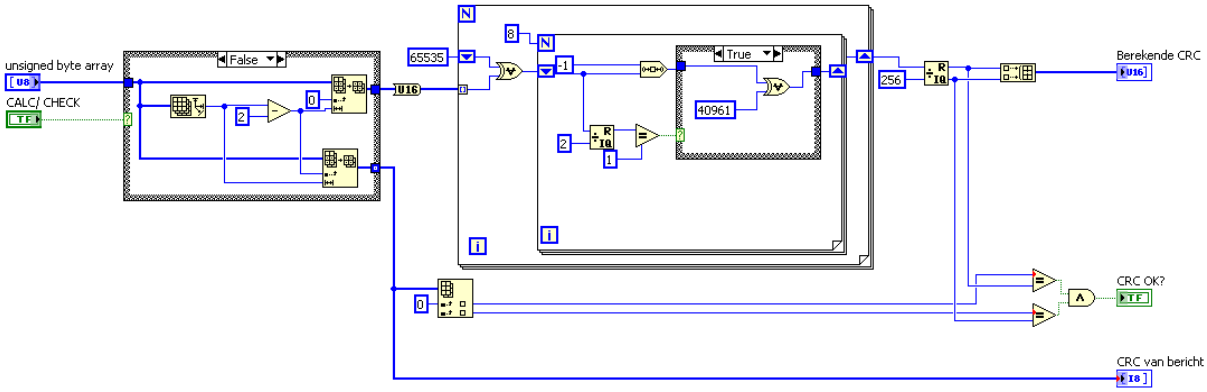
Omdat deze berekening vaak terug komt, zowel in het versturen van het bericht als in het controleren van het inkomende bericht is deze berekening in een aparte functieblok gemaakt.

In figuur 4.19 is de CRC Calc Check functieblok te zien.



figuur 4.19. CRC Calc/ check.

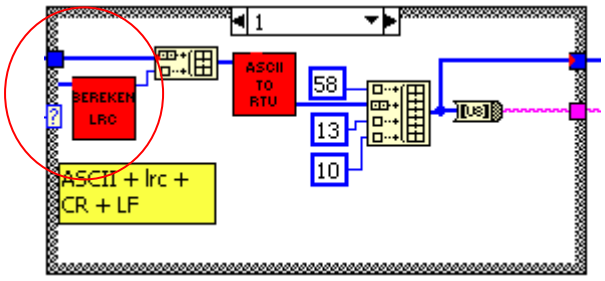
De broncode is te zien in figuur 4.20. Aan de hand van de TRUE/ FALSE variabele wordt de calculatie of check functie gebruikt. TRUE staat voor de calculatie, FALSE staat voor de check functie.



figuur 4.20. CRC calculatie/ check.

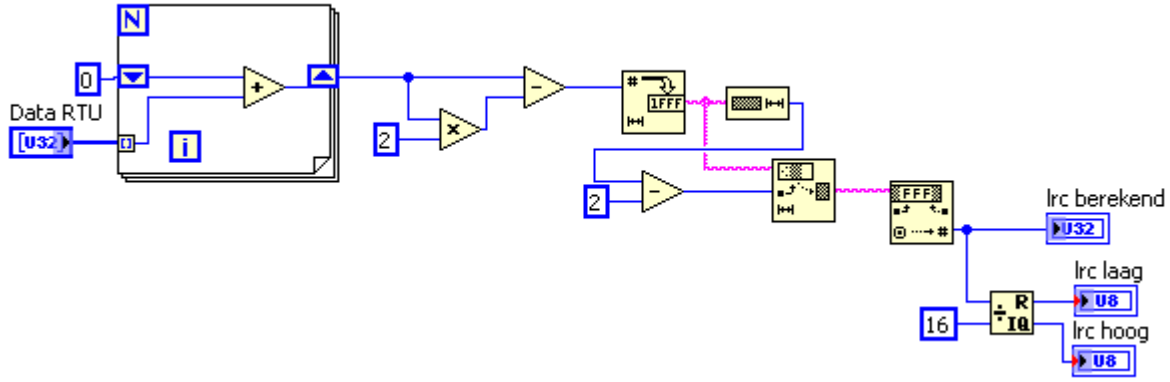
Lrc check.

In figuur 4.21 is de lrc berekening te zien. Deze lrc check wordt gebruikt bij het ASCII protocol en is ervoor om te controleren of een bericht foutloos is ontvangen. Er is gekozen voor een lrc functieblok omdat deze berekening vaker voor komt, namelijk bij uitgaand en ingaand bericht.



figuur 4.21. Lrc check.

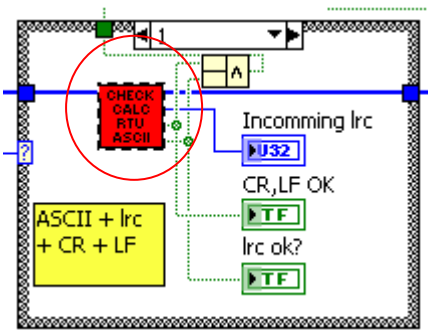
De broncode is te zien in figuur 4.22.



figuur 4.22. Broncode lrc berekening.

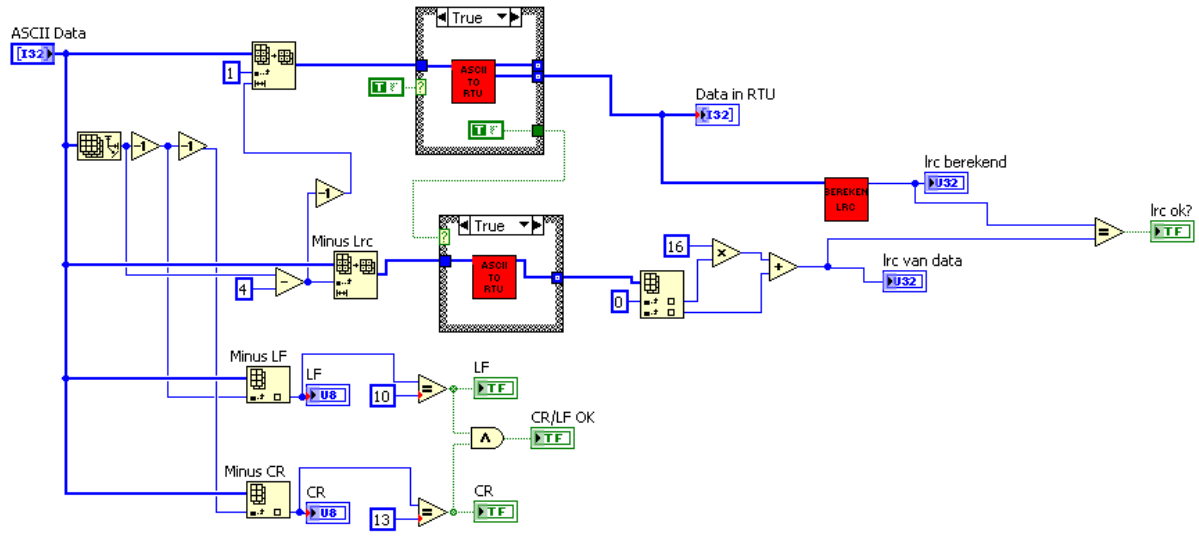
Check Calc RTU ASCII.

Als er een inkomend ASCII bericht is, dan moet er worden gekeken of de CR, LF aanwezig zijn en of de lrc goed is. Dit wordt gedaan door de functieblok CECK CALC RTU ASCII. Deze functieblok zorgt ook voor de vertaalslag naar RTU zodat het programma er mee verder kan werken. In figuur 4.23 is de Check, Calc RTU functieblok te zien.



figuur 4.23. CHECK CALC RTU ASCII functieblok.

Er is gekozen voor deze functieblok omdat deze berekening vrij groot is, waardoor het anders veel ruimte in beslag zou nemen.
In figuur 4.24 is de broncode van de CHECK CALC RTU ASCII te zien. In deze code wordt er gebruikt gemaakt van de functiecodes ASCII to RTU en bereken lrc.

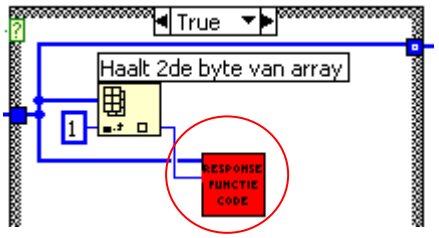


figuur 4.24. Broncode CHECK CALC RTU ASCII.

Als het bericht van de Slave is binnen gekomen en als de CRC of de lrc klopt mag het bericht worden uitgelezen. Aan de hand van de functiecode zal er een bepaalde actie worden uitgevoerd.
De acties zijn:

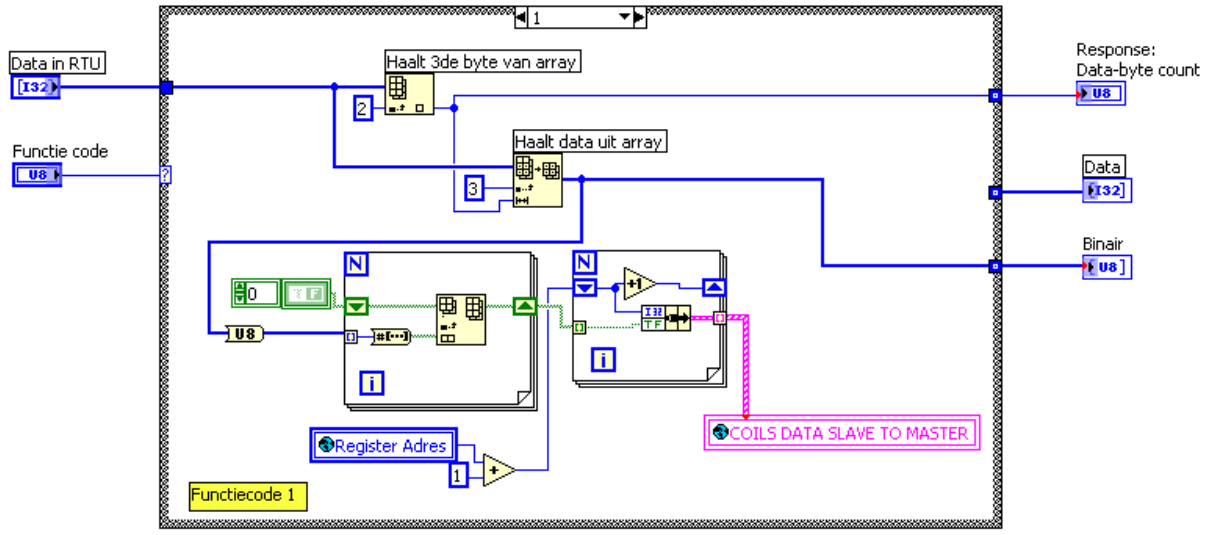
- Bij functiecode 15 en 16 wordt het responsebericht getoond,
- Bij functiecode 1 wordt de coils data uitgelezen, deze data wordt weggeschreven in de globale variabele “COILS DATA SLAVE TO MASTER”,
- Bij functiecode 3 wordt de registerdata uitgelezen, deze data wordt weggeschreven in de globale variabele “REGISTER DATA SLAVE TO MASTER”.

In figuur 4.25 is de functieblok “Response Functiecode” te zien. Er is voor deze functieblok gekozen omdat het veel ruimte in beslag neemt wat de leesbaarheid niet ten goede brengt.

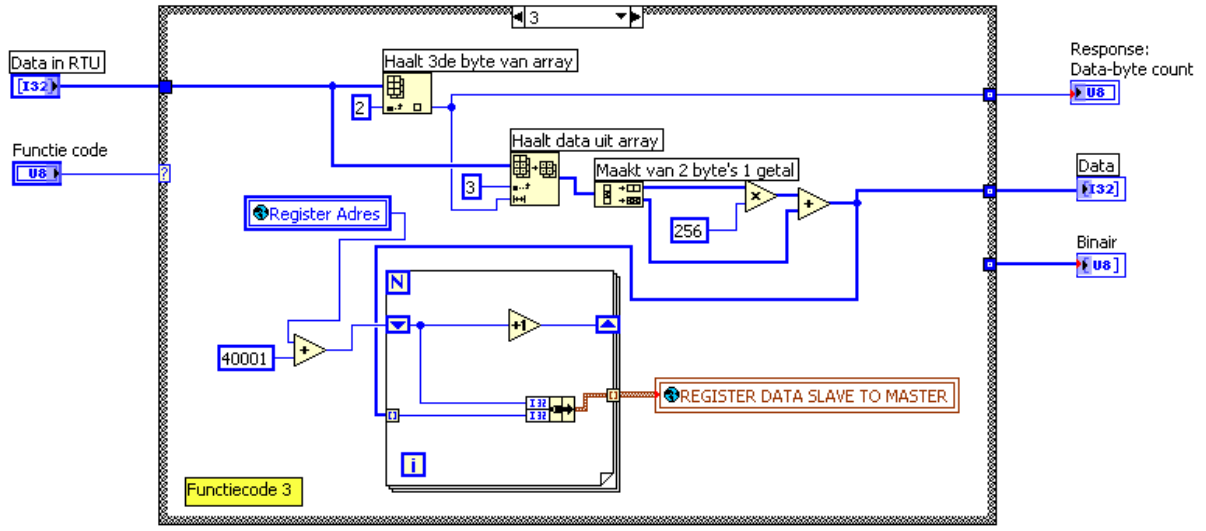


figuur 4.25. Functieblok Response functiecode.

In figuur 4.26 en 4.27 zijn de broncodes voor de functiecodes 1 en3 te zien.



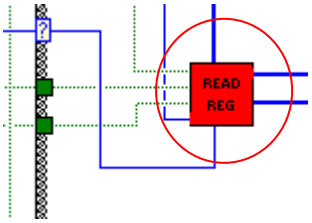
figuur 4.26. Broncode Response functiecode, functiecode 1.



figuur 4.27. Broncode Respons functiecode, functiecode 3.

Read red

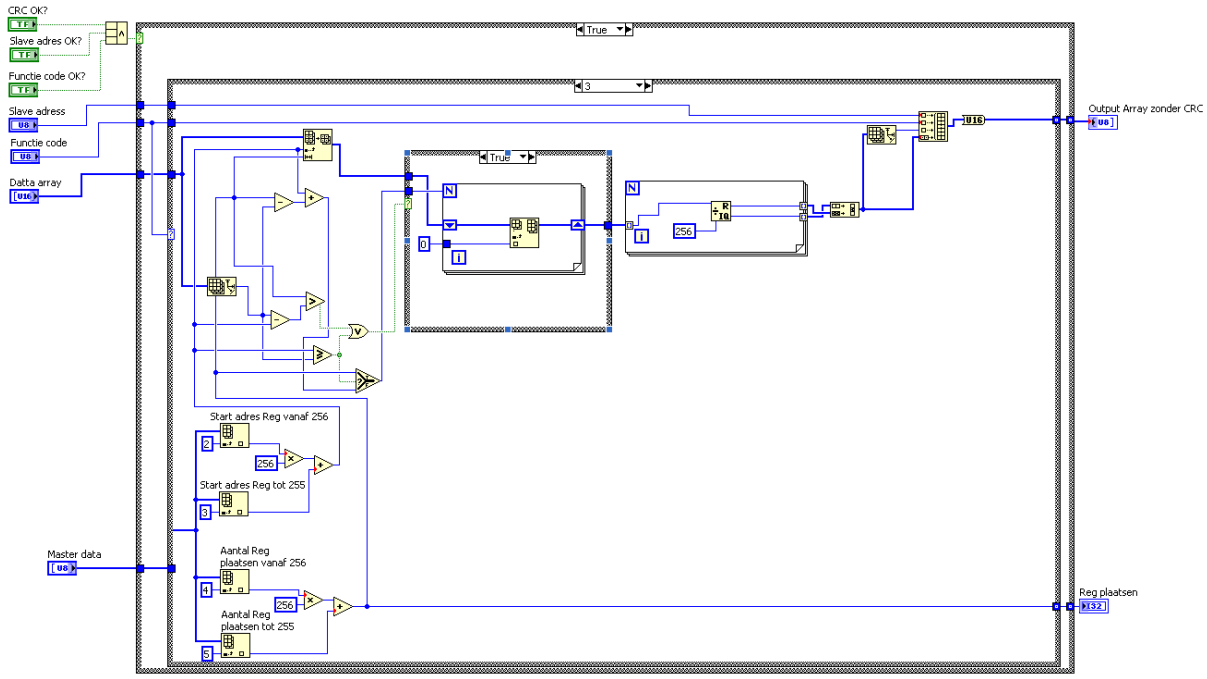
In figuur 4.28 is het functieblok READ REG te zien. Deze functieblok wordt gebruikt om de data van de slave in te lezen en een bericht van te maken.



figuur 4.28. READ REG.

Het bericht bestaat uit het Slave adres, functiecode en de data. Het bericht wordt pas gemaakt als de functiecode, Slave adres en CRC allemaal goed zijn.

In figuur 4.29 is de broncode van READ REG te zien.



figuur 4.29. Broncode READ REG.

4.4 Oplevering

Nadat de MODBUS simulaties zijn ontworpen en zijn gerealiseerd moeten de simulaties worden getest en worden opgeleverd.
Het testen wordt uitgevoerd samen met de bedrijfsbegeleider. Het testen gebeurd tussen twee laptops waarbij een MODBUS communicatie wordt opgezet. Hierbij worden verschillende situaties behandeld om de software zo goed mogelijk te kunnen testen. Bij verschillende situaties moet men denken aan verschillende functiecodes met verschillende data.

Nadat de software getest is wordt de software geïmplementeerd in de bestaande software.

4.5 Conclusie

De software is naar tevredenheid ontworpen en goed geïmplementeerd in de bestaande software. Eerst was het niet mogelijk om MODBUS te kunnen testen bij FAT testen omdat er geen test apparatuur voor MODBUS aanwezig was. Hierdoor kon er past worden getest als het besturingssysteem op site wordt geïnstalleerd. Met de MODBUS simulaties kunnen MODBUS componenten nu al in de fabriek worden getest.

Imtech Vonk en haar werknemers zijn zeer tevreden en enthousiast met de software.

5. "User-friendly" IO configuratie

In de huidige situatie moeten de IO's handmatig worden ingevoerd in de hardware simulator. Dit kost vrij veel tijd en is erg omslachtig. Daarom moet er een "User-friendly'IO configuratie komen. Om een goed werkend software programma te krijgen moet er worden nagegaan waaraan de software moet voldoen, welke stappen moet het ondernemen en welke handelingen moet er worden uitgevoerd.

5.1 Ontwerp "User-friendly" IO configuratie.

Om tot een goed ontwerp te kunnen komen moeten de eisen op een rij worden gezet. De volgende eisen worden aan de "User-friendly" IO configuratie gesteld:

- Er moet één main lijst komen. Hierin staan alle IO tags die moeten worden verdeeld over de verschillende in- en uitgangen. Deze lijst wordt gevuld door een zogenaamde dump file. Dit is een file die wordt gegenereerd door E-plan. In E-plan worden alle elektrische tekeningen gemaakt. In deze file komen alle IO tags van de desbetreffende besturingskast te staan.
- Er moeten 5 lijsten komen; digitaal in, analoog in, digitaal uit, analoog uit en trash. In deze lijsten kunnen de IO tags komen te staan.
- Er moet een highlight functie komen om de geselecteerde IO tags in de lijsten weer te geven.
- Door middel van verschillende knoppen (lees DI, AI, DO, AO en TRASH) kan worden aangegeven waar de IO tag naar moet worden verplaatst. Hierbij wordt de eerste IO tag in de lijst verplaatst.
- Per in- of uitgang moet er een move up, move down functie komen. Met deze functie kan een IO tag door de lijst heen verplaatsen.
- Er moet een back functie komen. Mocht er een fout zijn gemaakt, bijvoorbeeld de IO tag staat in de verkeerde lijst, dan moet via de back knop de IO tag terug kunnen worden gezet in de main lijst. Er is ervoor gekozen om de IO tag achter aan de main lijst toe te voegen.
- Er moet een delete knop komen om de IO tag gelijk te kunnen verplaatsten naar de TRASH lijst.
- De hardware simulator wordt aangesloten door middel van E56 connectoren aangesloten op de te testen besturingskast. Een E56 connector bestaat uit 56 aders. Een IO maakt gebruik van 2 aders. Per connector kunnen er dus 56/2 = 28 IO's op één connector worden aangesloten.

  Het kan zich voordoen dat er twee besturingskasten zijn, de één heeft 20 IO's de andere heeft er 8. Theoretisch zou dit op één E56 connector passen. Maar omdat het twee besturingskasten betreft is het wenselijk om dit te verdelen over 2 connectoren.
  De eerste connector wordt dan "gevuld" met 20 IO's. De overige 8 plekken moeten worden opgevuld met lege plekken, ook wel spare genoemd.
  Hiervoor dient de spare knop. Als er op de knop wordt gedrukt, zal vanaf de eerste plek naar de geselecteerde IO tag de rest worden opgevuld met spare tot aan het totaal 28 IO is. Als voorbeeld is tabel 5.1 gegeven, hierin staan 28 IO's.

Nummer	IO tag
1	11-SI-2078
2	11-SI-2079
3	11-LIA-2051B
4	11-PDIA-2166
5	11-PDIA-2176
6	11-TIA-2172
7	11-TIA-2173
8	11-PDIA-2175
9	11-LIZA-2051A
10	11-TIZA-2174
11	11-PIA-2165
12	11-PIZA-2153A
13	11-PIZA-2153B
14	11-PIZA-2153C
15	11-TIZA-2164
16	11-TIZA-2165
17	11-TIZA-2166
18	11-TIZA-2167
19	11-TIZA-2168
20	11-TIZA-2169
21	11-FIA-2061
22	11-FIA-2062
23	11-FIA-2063
24	11-PDIA-2171
25	11-PDIA-2174
26	11-PIA-2154B
27	11-PIA-2155B
28	11-PIA-2156B

tabel 5.1. IO lijst.

Vanaf plaats 21 moet de rest op een tweede connector komen te zitten. Door middel van de spare knop worden de plekken 21 t/m 28 gevuld met spares. De overige IO tags komen op een nieuwe connector te zitten. De lijst komt er als volgt uit te zien en is te zien in tabel 5.2.

Nummer	IO tag
1	11-SI-2078
2	11-SI-2079
3	11-LIA-2051B
4	11-PDIA-2166
5	11-PDIA-2176
6	11-TIA-2172
7	11-TIA-2173
8	11-PDIA-2175
9	11-LIZA-2051A
10	11-TIZA-2174
11	11-PIA-2165
12	11-PIZA-2153A
13	11-PIZA-2153B
14	11-PIZA-2153C
15	11-TIZA-2164

16	11-TIZA-2165
17	11-TIZA-2166
18	11-TIZA-2167
19	11-TIZA-2168
20	11-TIZA-2169
21	spare
22	spare
23	spare
24	spare
25	spare
26	spare
27	spare
28	spare
1	11-FIA-2061
2	11-FIA-2062
3	11-FIA-2063
4	11-PDIA-2171
5	11-PDIA-2174
6	11-PIA-2154B
7	11-PIA-2155B
8	11-PIA-2156B

tabel 5.2. IO lijst.

- Er moet een overzicht komen waarin te zien is hoeveel connectors voor elke in- uitgang in gebruik zijn.
- Naast de IO tags in de lijsten DI, AI, DO, AO moet te zien zijn op welke connector de IO tag is aangesloten.
- Naar dat alle IO tags zijn verdeeld zullen de IO tags met één druk op de knop worden ingeladen in de bestaande software van de hardware simulator. Er moet ook een lijst komen met daarin hoe de hardware simulator moet worden aangesloten op de besturingskast.

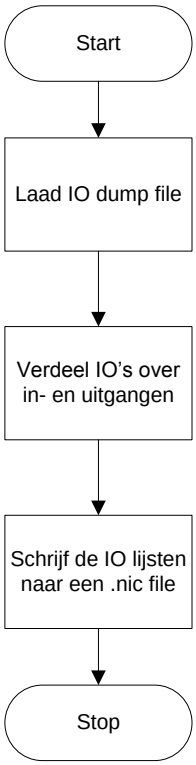
De bovenstaande eisen zijn er om een doel te realiseren, het doel is,
Het is de bedoeling dat er een IO dump file komt. Deze IO dump file wordt in het tekenprogramma Eplan gemaakt. In deze file staan alle IO tags, klemmenreeks en klemnummer. Deze IO dump file kan dan worden ingeladen in de User-friendly IO configuratie. Om er voor te zorgen dat de IO dump file steeds deze zelfde opbouw heeft is met de afdeling CAD engineering afgesproken dat er een bepaalde formaat komt voor de IO dump file.
Het formaat van de IO dump file is te zien in tabel 5.3.

IO tag	Klemmen reeks	Klem nummer
BP01-S1	113X2	5
BP01-S2	113X2	6
BP01-P1	113X2	8
BP01-P1	113X2	7
BP01-S1	113X2	5
BP01-S2	113X2	6
BP01-S1	113X2	5
BP01-S2	113X2	6

tabel 5.3. IO dump file.

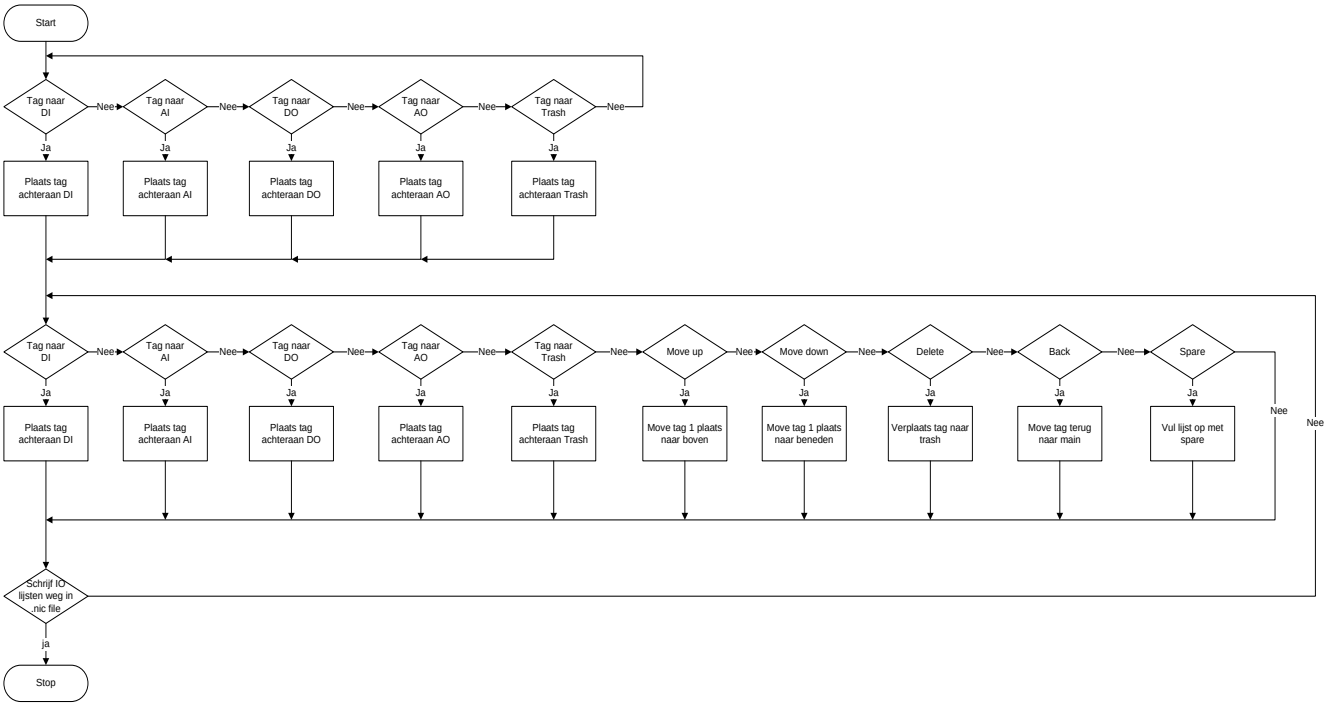
Nadat er een IO dump file is moet deze worden ingeladen.
Nadat de IO dump file is ingeladen moeten de IO's worden verdeeld over de in- en uitgangen.
Daarna moeten de in- en uitgangslijsten worden ingeladen in de hardware simulator, dit wordt gedaan door de in- en uitgangslijsten weg te schrijven in een .nic file. Deze file kan dan weer worden in geladen in de hardware simulator.

Naar dat alle eisen bekent zijn en er een globale idee is welke stappen het programma moet uitvoeren moet het programma structuur worden uitgedacht. Dit wordt uitgedacht door middel van de Koninklijke weg. Deze is te zien in figuur 5.1.



figuur 5.1. Flowchart "User-friendly" IO configuratie.

Het belangrijkste onderdeel van de "User-friendly" IO configuratie is het onderdeel "Verdeel IO's over in- en uitgangen". Dit onderdeel wordt ook apart behandeld in een flochart. Zie figuur 5.2.



figuur 5.2. Flowchart Verdeel IO's over in- en uitgangen.

Aan de hand van de fowchart kan de software worden gemaakt. De uitwerking is te zien in paragraaf 5.2 Uitwerking "User-friendly" IO configuratie.

5.2 Uitwerking "User-friendly" IO configuratie

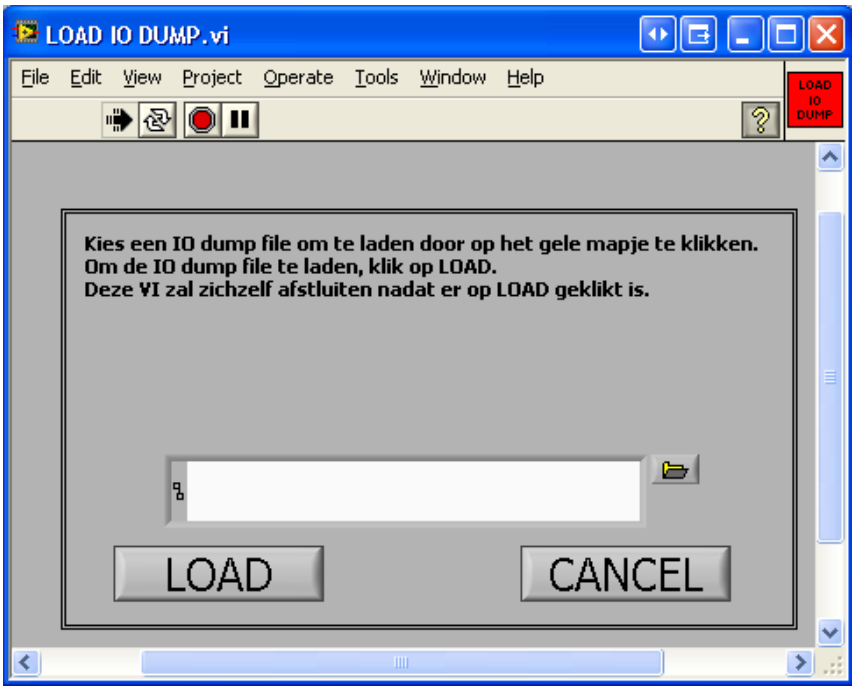
Aan de hand van de flowchart in de vorige paragraaf is de software ontwikkeld. Tijdens het ontwikkelen van de software kunnen er problemen ontstaan die terplekke moeten worden opgelost.

De "User-frienldy" IO configuratie bestaat uit 3 onderdelen. Het laden van de IO dump file. Het verdelen van de IO's over de in- en uitgangen. En het wegschrijven van de in- en uitgangen naar een .nic file. De User-friendly IO configuratie wordt aangeroepen vanuit de bestaande software.

Het programma bestaat uit 3 functieblokken, het eerste functieblok is de "LOAD IO DUMP".

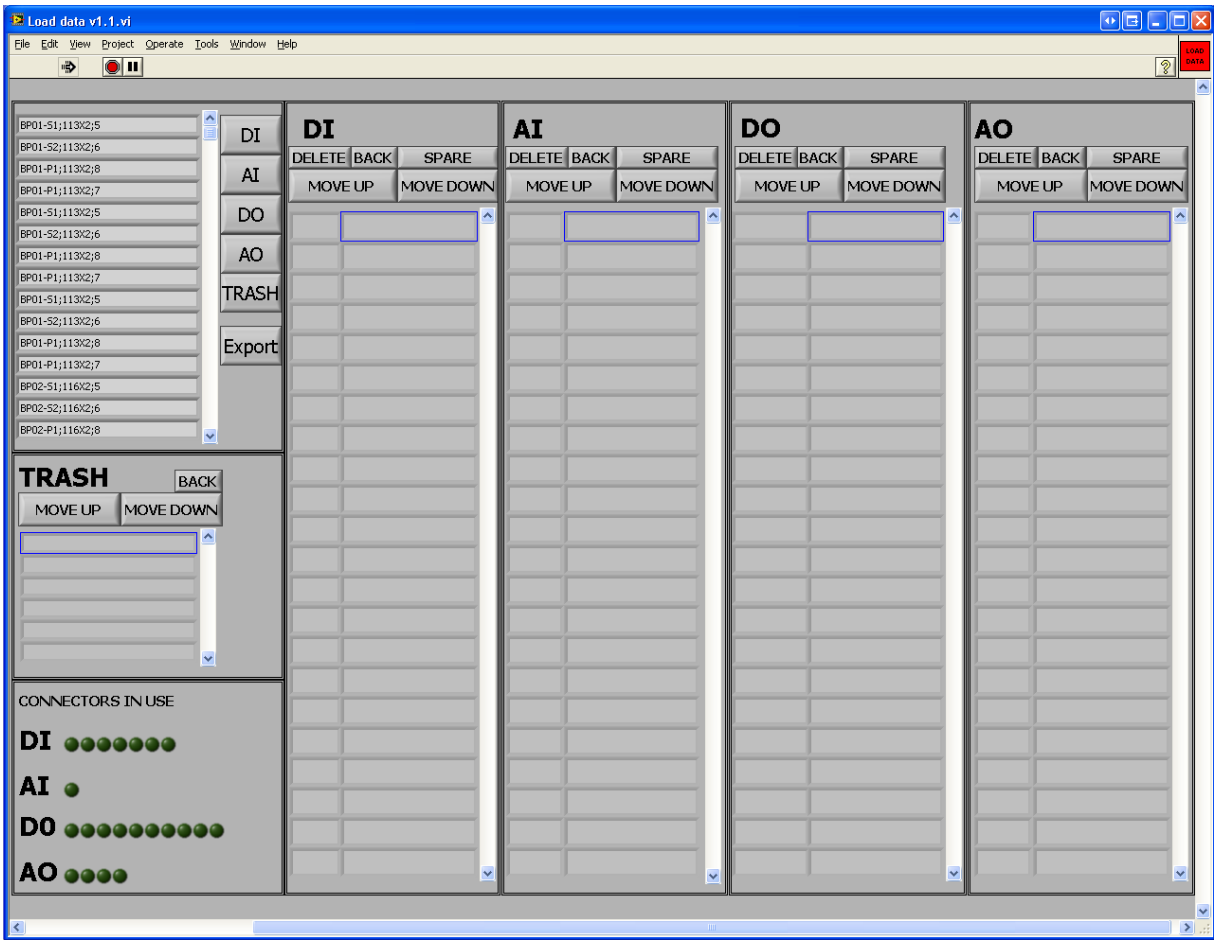
5.2.1 Load IO dump

In deze functieblok komt het VI dat te zien is in figuur 5.3 tevoorschijn. In deze VI kan de IO dump file worden gekozen. De functieblok zorgt er voor dat de IO tags allemaal onder elkaar komen te staan en dat de IO lijst naar het volgende functieblok wordt geladen.



figuur 5.3. LOAD IO DUMP.

Zodra de IO dump file is ingeladen wordt de functieblok LOAD DATA aangeroepen. Als deze functieblok wordt aangeroepen dan verschijnt de VI dat is weergegeven in figuur 5.4.



figuur 5.4. LOAD DATA VI.

In de LOAD DATA VI kunnen de IO tags worden verdeeld over de in- uigangen DI, AI, DO AO of TRASH. Tevens kunnen de tags in de lijsten worden verplaatst van positie, terug worden gezet naar de Main lijst of worden verwijderd naar TRASH.
De DI, AI, DO en AO kunnen worden opgevuld met lege plekken oftewel spare.

Achter elke knop zit een functie die de desbetreffende actie uitvoert. Elke functie wordt in paragraaf 5.2.2 uitgewerkt.

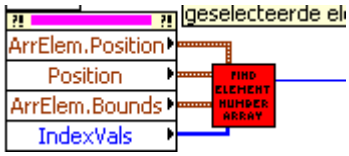
5.2.2 Uitwerking functies

Find element number array.

De eerst functie is de Find element number array. Als er een IO TAG in een lijst wordt geselecteerd, geeft de functie de positie in de lijst weer. Deze positie wordt gebruikt voor de functieblokken: MOVE TO IO LIST, MOVE BACK TO MAIN, ARRAY HIGHLIGHT en de MOVE UP & DOWN functie.

Een eigenschap van deze functie is dat de positie waarde pas veranderd als er een andere IO tag wordt geselecteerd.

In figuur 5.5 is de aanroep van deze functie te zien.



figuur 5.5. Aanroep FIND ELEMENT NUMBER ARRAY.

Als ingang heeft hij de weergave eigenschappen van de array, de uitgang is de positie van de geselecteerde element.

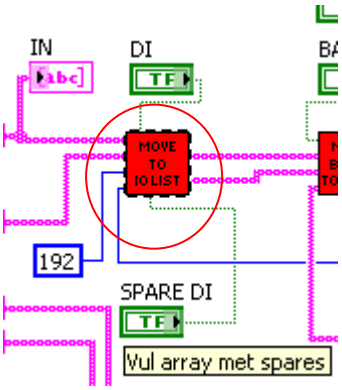
Move To IO list.

De volgende functie is de functie “Move To IO list”. Deze functie zorgt er voor dat de eerste tag in de main lijst naar de desbetreffende IO lijst wordt verplaatst. In figuur 5.6 is de main lijst met de keuze knoppen.

figuur 5.6. Main lijst met keuze knoppen.

Met de knoppen DI, AI, DO, AO en TRASH kan de eerste tag uit de lijst worden verplaatst naar de gewenste lijst. Er is ervoor gekozen de keuze alleen geld voor de 1ste tag in de rij, zodat alle tags aan bod komen.

In figuur 5.7 is de aanroep van de functieblok Move to IO list te zien. Omdat de functie tags naar een lijst verplaatst, dus in principe de lijst aanvult met tags, is er voor gekozen om de functie “SPARE” ook in deze functieblok toe te voegen.



figuur 5.7. Aanroep functie MOVE TO IO LIST.

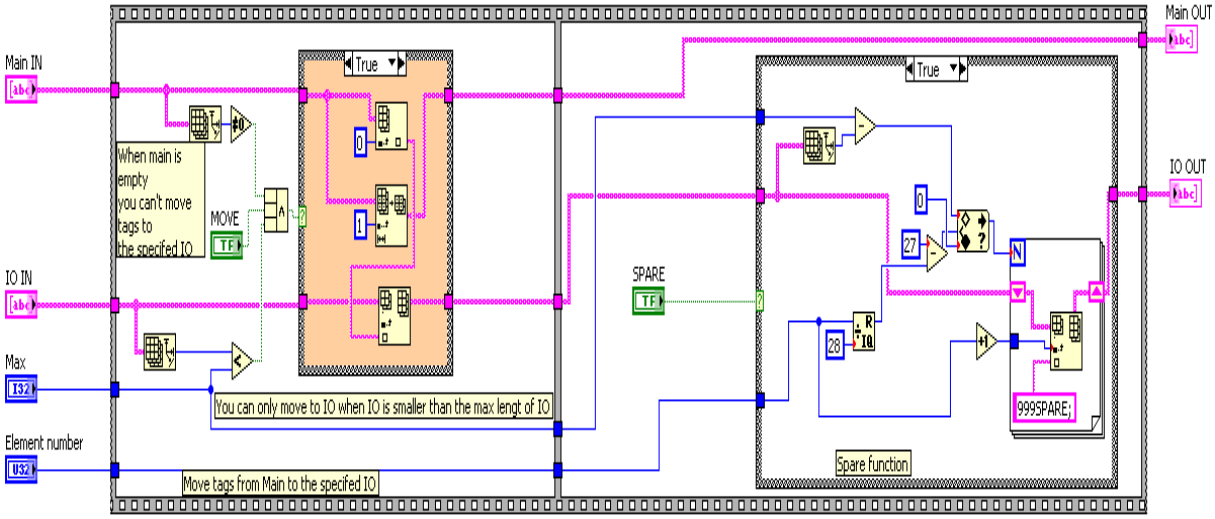
De functie heeft 6 ingangen en 2 uitgangen. De ingangen zijn:

- Main lijst in, hier komt de Main lijst naar binnen,
- IO lijst in, hier komt de desbetreffende IO lijst naar binnen,
- Maximum aantal tags, geeft aan hoeveel tags er kunnen worden ingeladen,
- Elementnummer, voor het aangeven van de geselecteerde element,
- Move knop, voor het verplaatsen van de tag in Main naar de IO lijst,
- SPARE, voor het opvullen van de IO lijst met tags tot aan de eerste 28.

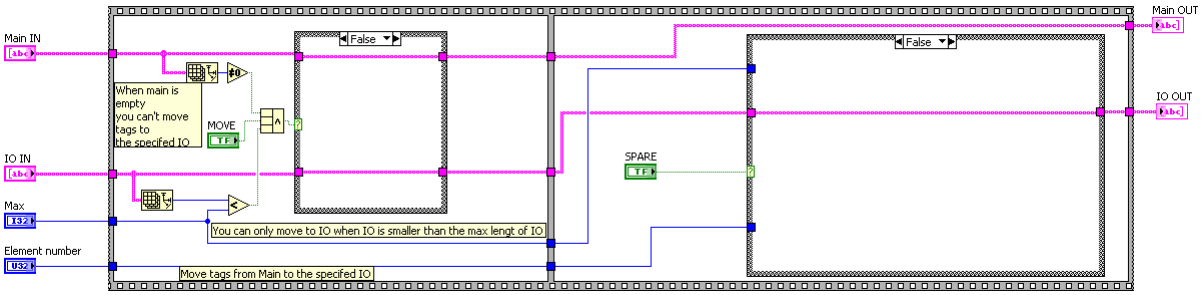
De uitgangen zijn:

- Main lijst out, is verandert als de functie move of spare heeft plaats gevonden.
- IO lijst out, is veranderd als de functie move of spare heeft plaats gevonden.

In figuur 5.8.1 en 5.8.2 is de broncode van de functie te zien.

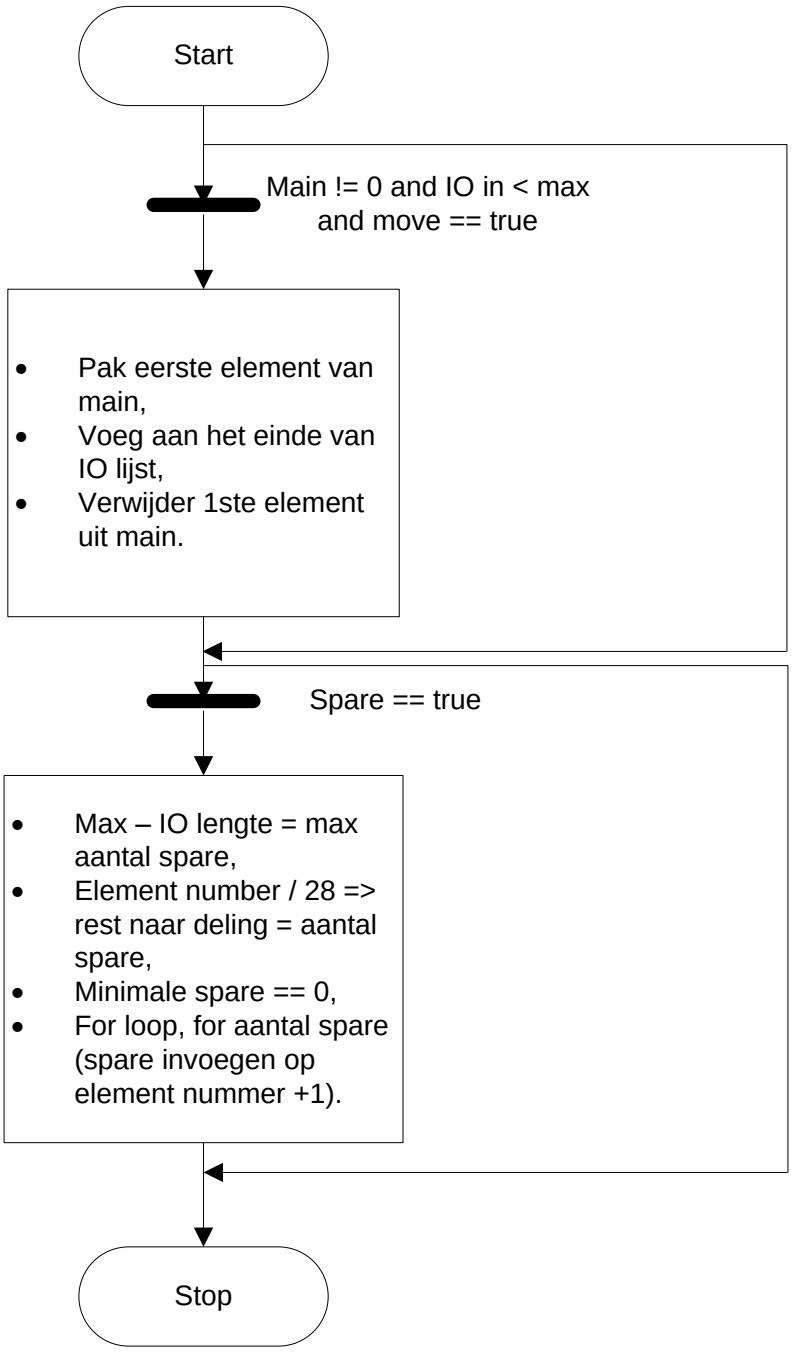


figuur 5.8.1. Broncode Move IO to list.



figuur 5.8.2. Broncode Move IO to list.

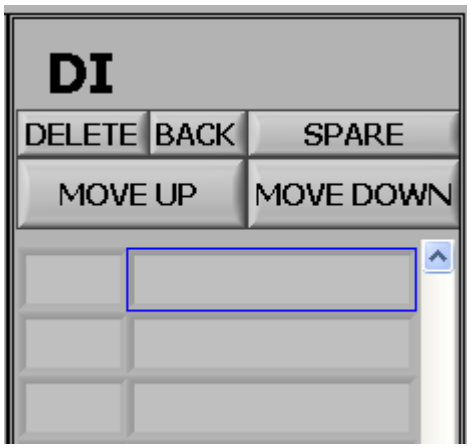
De werking van de Move to IO list is schematisch weergegeven in figuur 5.9.



figuur 5.9. Schematische werking Move to IO list.

Move back to main.

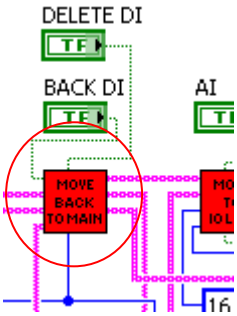
De volgende functie is “Move Back to Main”. Deze functie zorgt er voor dat de IO’s die door een (menselijke) fout verkeerd zijn komen de staan weer terug kunnen worden gezet door middel van de “BACK” knop, als de tag in de TRASH lijst hoort te staan kan dat worden gedaan door de “DELETE” knop. In figuur 5.9 zijn de knoppen te zien.



De knoppen “DELETE” en “BACK” worden gebruikt voor deze functie. De selecteerde tag zal worden gebruikt voor de functie.

Figuur 5.9. DI lijst met keuze knoppen.

In figuur 5.10 is de functie aanroep van Move Back to Main te zien.



figuur 5.10. Aanroep functie MOVE BACK TO MAIN.

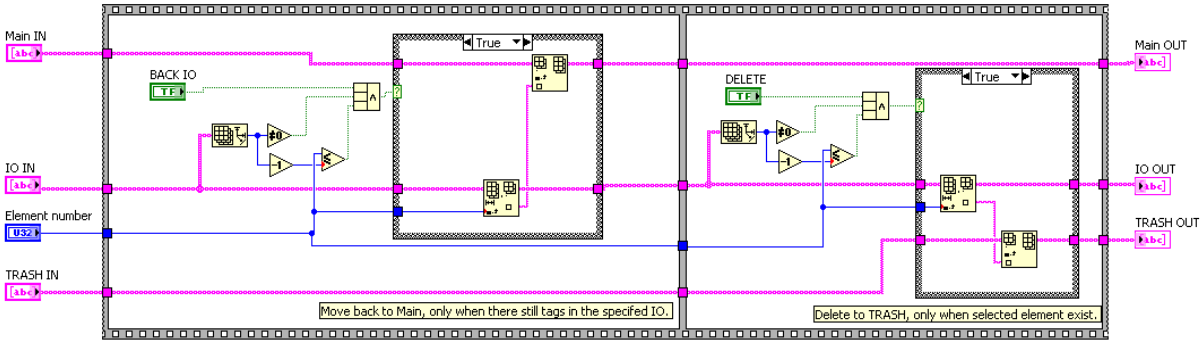
De functie heeft 6 ingangen en 2 uitgangen. De ingangen zijn:

- Main Lijst in; Hier komt de Main lijst naar binnen,
- IO lijst in; Hier komt de desbetreffende IO lijst naar binnen,
- TRASCH in; Hier komt TRASH lijst naar binnen,
- Element number; Deze dient ervoor om de geselecteerde element aan te geven die moet worden verplaatst,
- DELETE; Hiermee wordt de functie delete geactiveerd,
- BACK IO; Hiermee wordt de functie back geactiveerd.

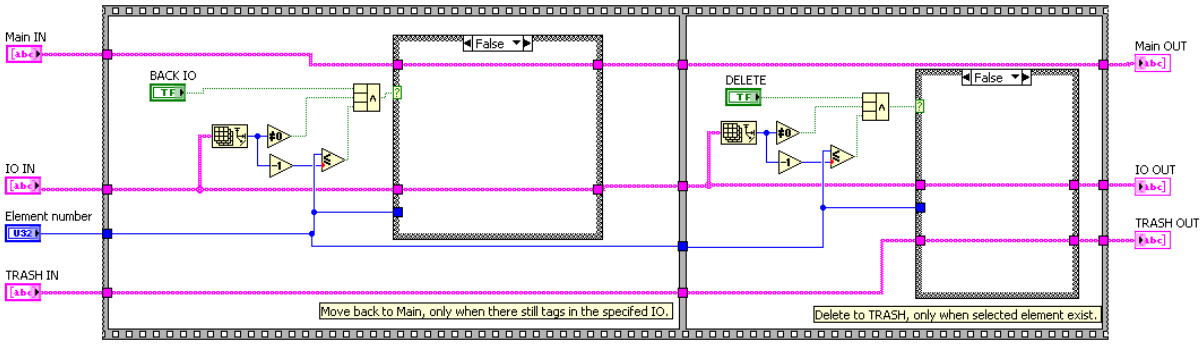
De uitgangen zijn;

- Main out; Is veranderd als de functie back is uitgevoerd,
- IO out; Is veranderd als de functie back of delete is uitgevoerd,
- TRASH; Is veranderd als de functie deletie is uitgevoerd.

In figuur 5.11.1 en 5.11.2 is de broncode van de functie te zien.

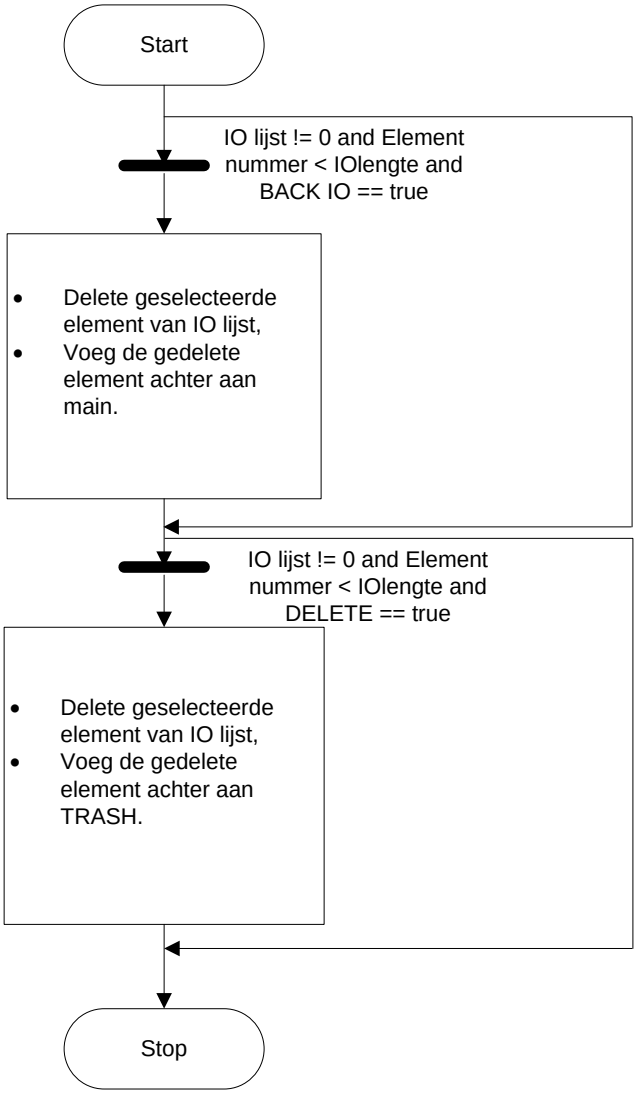


Figuur 5.11.1. Broncode Move Back to Main.



figuur 5.11.2. Broncode Move Back to Main.

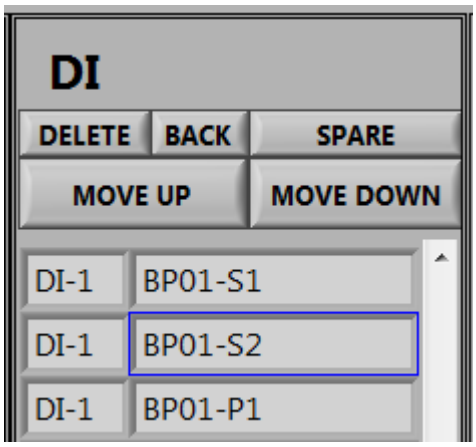
De werking van de Move Back to Main is schematisch weergegeven in figuur 5.12.



figuur 5.12. Schematische werking MOVE BACK TO MAIN.

MOVE UP & MOVE DOWN.

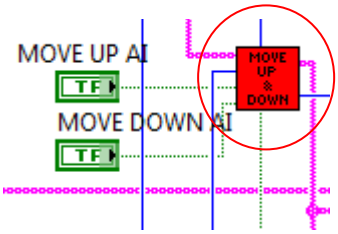
De volgende functie is de “MOVE UP & MOVE DOWN”. Deze functie zorgt er voor dat de IO tags in de lijsten kunnen worden verplaatst. Door de te verplaatsen IO tag de selecteren en of de “MOVE UP” of de “MOVE DOWN” de drukken kan de IO tag worden verplaatst. Dit is te zien in figuur 5.163.



De tag kan worden geselecteerd. Door middel van de knoppen “Move UP’’ & “MOVE DOWN” zal de tag worden verplaatst. Een belangrijke eis is dat door meerdere keren op bijvoorbeeld de move down knop te drukken, dat de tag zich steeds zal verplaatsen door de lijst zonder dat de tag opnieuw wordt geselecteerd. De tag blijft dus geselecteerd.

figuur 5.13. DI lijst met keuze knoppen.

In figuur 5.14 is de functie aanroep van MOVE UP & MOVE DOWN te zien.



figuur 5.14. Aanroep functie MOVE UP & MOVE DOWN.

De functie heeft 6 ingangen en 2 uitgangen. De ingangen zijn:

- IO in; De inkomende IO lijst,
- Element nummer; De werkelijke element nummer,
- Element shift; Het element nummer dat is veranderd door een “MOVE UP” of “MOVE DOWN” actie,
- “MOVE UP”; Hiermee wordt de actie “MOVE UP” uitgevoerd,
- “MOVE DOWN”; Hiermee wordt de actie “MOVE DOWN” uitgevoerd,
- Differens in Element Value; Hiermee wordt aangegeven of er een nieuwe TAG is geselecteerd.

De uitgangen zijn:

- IO lijst uit, de uitgaande IO lijst; Kan zijn veranderd als er een move up of move down actie is uitgevoerd.
- Element number out; Dit is de uitgaande element nummer.

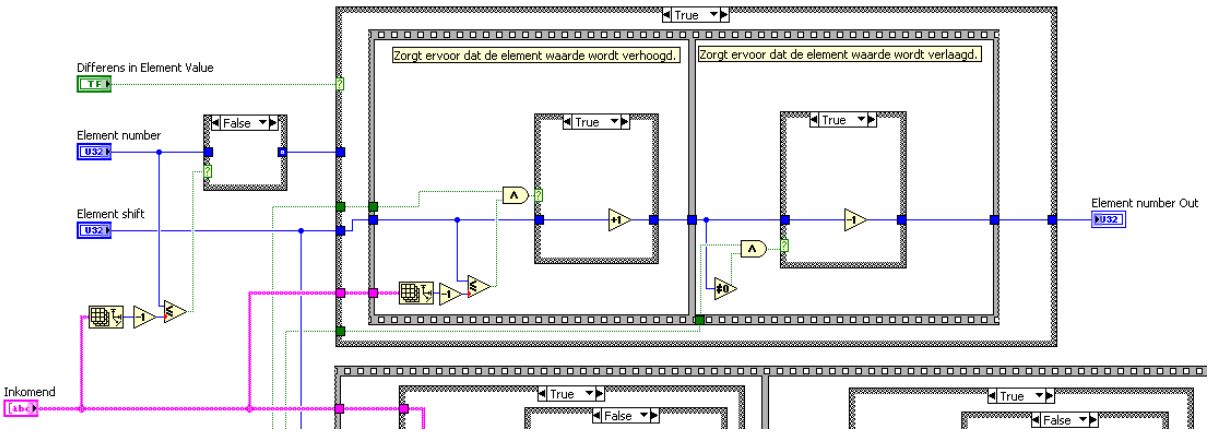
De functie move up, move down bestaat uit twee onderdelen. Het eerste geeft aan om welke tag IO (element) het gaat.

De werking is als volgt:

(De volgende stappen worden buiten de functieblok uitgevoerd).

De IO die moet worden verplaatst wordt geselecteerd. Hierbij wordt de positie in de lijst bepaald (adres nummer array. Daarna wordt er een vergelijking gemaakt met de huidige positie en de vorige positie (wordt dit voor de eerste keer uitgevoerd dan wordt de huidige positie vergeleken met 0). Als beide waardes gelijk zijn betekend dat de er geen andere tag geselecteerd is. Zijn de waardes ongelijk aan elkaar, dan is er voor een andere tag gekozen.

In figuur 5.15 is het stuk broncode van de eerste functie te zien.

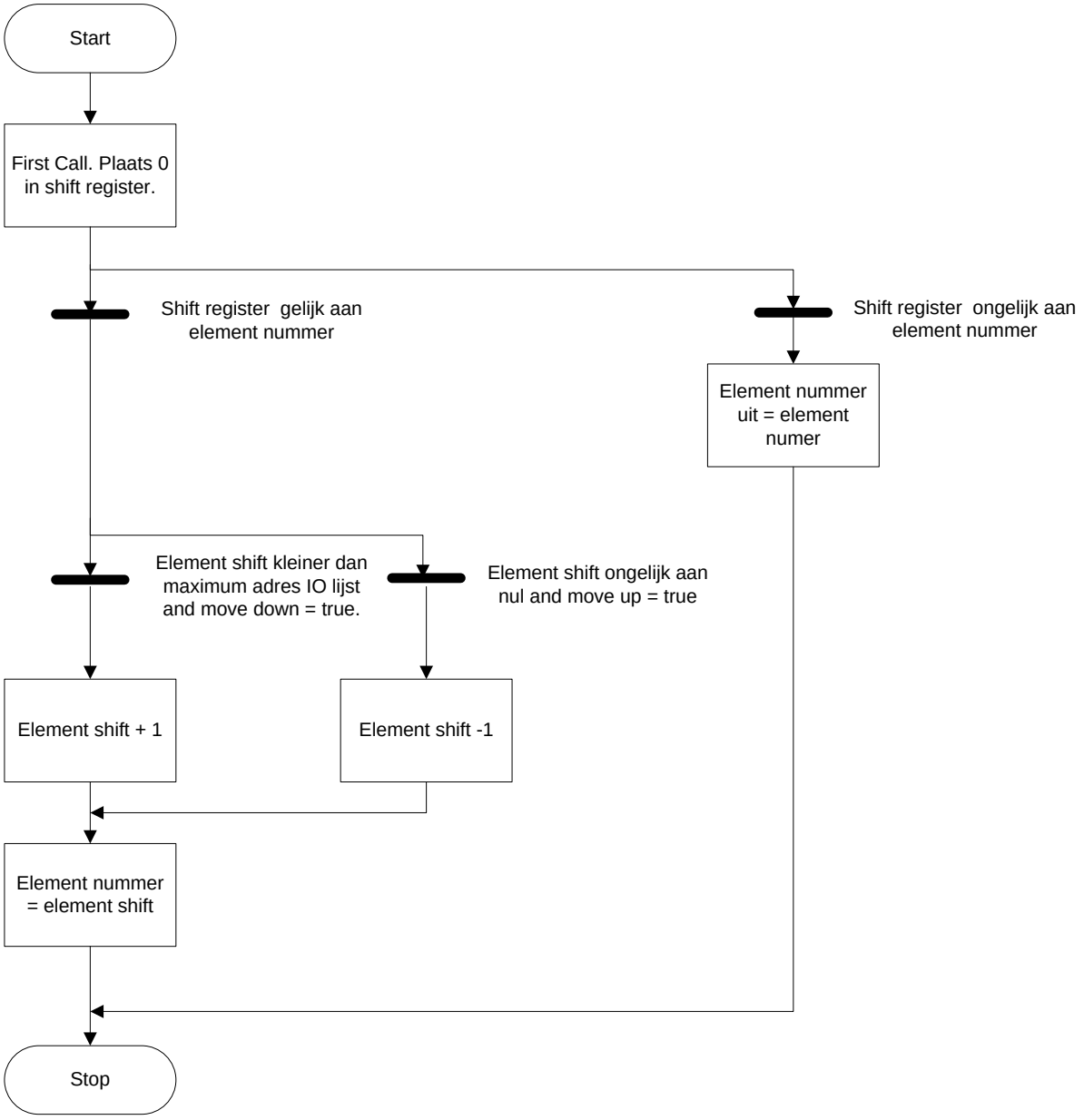


figuur 5.15. Broncode van de eerste functie.

(De volgende stappen worden in de functieblok uitgevoerd).

Als er een verschil tussen de twee posities zit wordt het originele element nummer doorgegeven. Mocht er gen verschil zitten, dan wordt aan de hand van of move up of move down het element nummer verhoogd of verlaagd, voor move up geld dat de IO niet de eerste in de lijst mag zijn en voor move down geld dat de IO niet de laatste in de lijst mag zijn.

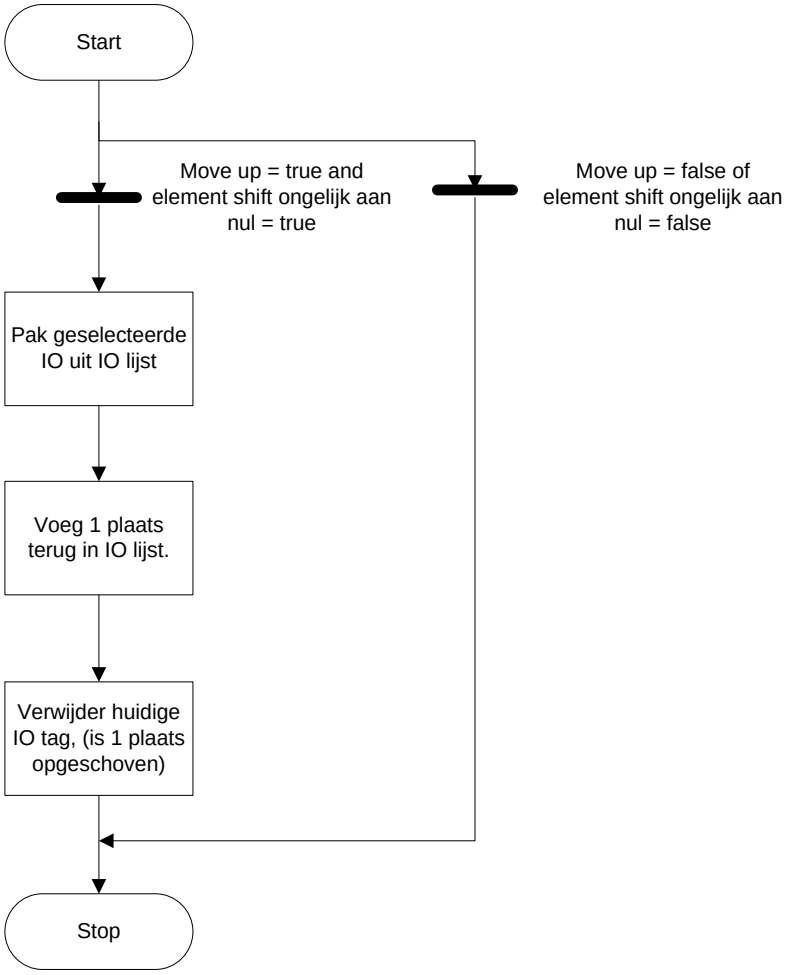
Een schematische werking is te zien in figuur 5.16.



figuur 5.16. Schematische werking 1ste functie Move up & Move down.

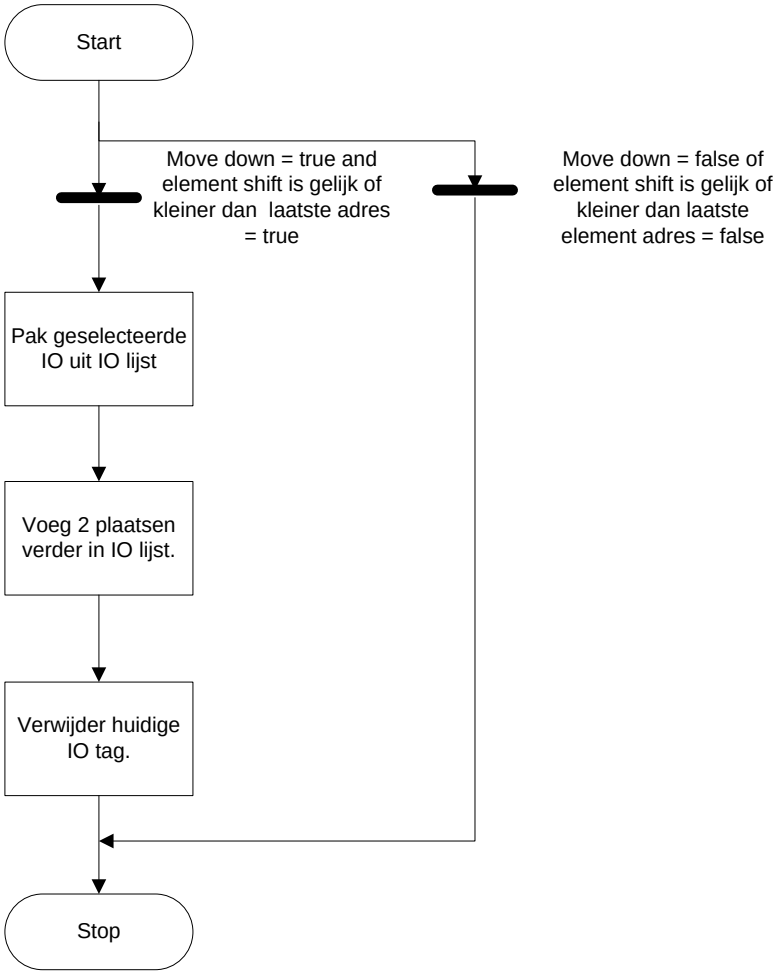
De 2de functie is de echte Move up & Move down functie. Deze functie zal aan de hand van de knoppen Move up of Move down de IO verplaatsten. Om de IO tag naar boven te verplaastsen mag de IO tag niet de eerste in de rij zijn omdat anders de IO tag zal verdwijnen. De IO tag mag ook niet de laatste in de rij zijn voor de move down functie anders zal er een lege plek ontstaan.

De voorwaarde voor de move up functie is dat: de tag mag niet de eerste in de rij zijn (tag adres moet ongelijk aan nul zijn, uitkomst is dan true) en de knop move up moet true zijn. Als deze waardes beide true zijn dan mag de move up functie worden geactiveerd.
Zodra de functie is geactiveerd gebeurt het volgende: Aan de hand van de geselecteerde IO wordt deze gekopieerd uit de IO lijst en één plaats onder zijn huidige positie in de IO lijst geplaatst. De IO lijst is nu 1 IO tag groter en de originele IO tag is 1 plaats naar onder geschoven, deze IO tag zal dan ook worden verwijdert. Een schematische weergaven van de werking is te zien in figuur 5.17.



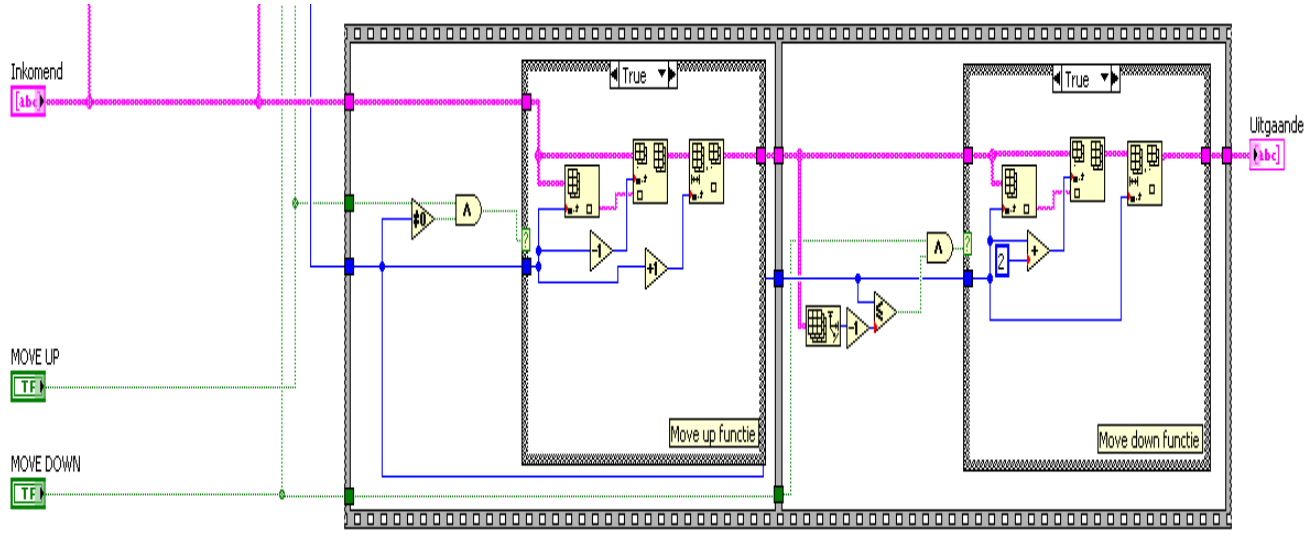
figuur 5.17. Schematische werking Move up.

De voorwaarde voor de Move down zijn: De Move down knop moet true zijn, de element mag niet buiten de lijst vallen. Als deze waardes true zijn dan wordt de functie Move down uitgevoerd. De volgende acties worden uitgevoerd bij de Move down functie. De geselecteerde IO wordt gekopieerd, wordt 2 plaatsen verder in de lijst geplaatst (dit omdat als de tag 1 plaats wordt opgeschoven, hij achter het origineel wordt gezet, als het origineel wordt verwijderd is er niks veranderd), daarna wordt het origineel verwijderd.
Ter verduidelijking is in figuur 5.18 de schematische werking te zien.



figuur 5.18. Schematische werking Move down.

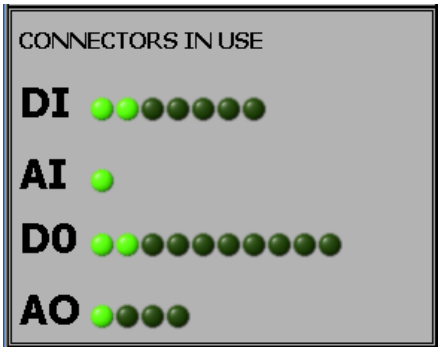
De broncode van het 2de gedeelte van de Move up, move down functie is te zien in figuur 5.19.



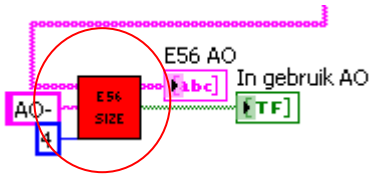
figuur 5.19. Broncode 2de gedeelte Move up, Move down.

E56 SIZE.

Een eis is dat er een overzicht komt waarin in staat weergegeven hoeveel E56 connectoren in gebruik zijn er hoeveel er nog beschikbaar zijn. In de IO lijsten moet zijn aangegeven op welke connector de tag zich bevind. De eisen worden ingewilligd door de functieblok E56 Size. Het overzicht van de E56 connectoren is te zien in figuur 5.20. De functie aanroep is te zien in figuur 5.21.



figuur 5.20 Overzicht E56 connectors in gebruik



figuur 5.21. Aanroep E56 SIZE functie.

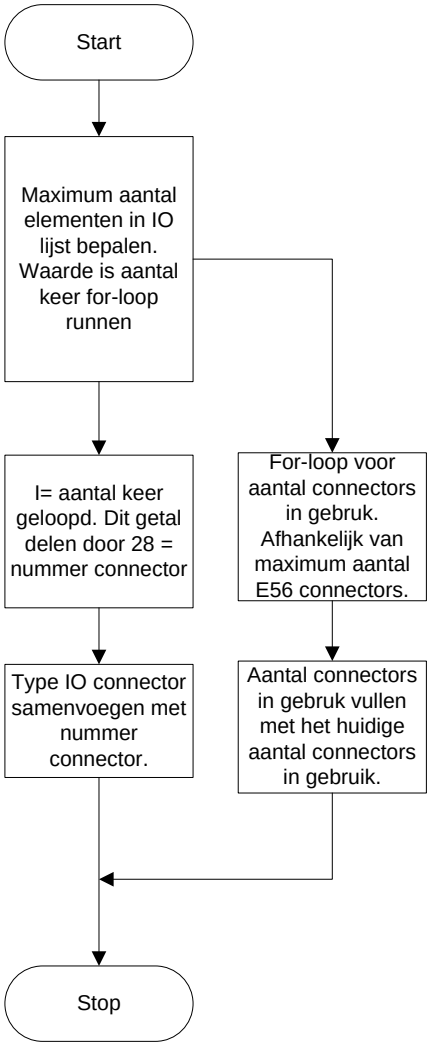
De functieblok heeft 3 ingangen en 2 uitgangen.
De ingangen zijn:

- IO lijst; Dit is de desbetreffende IO lijst, wordt gebruikt om te bepalen hoe groot de IO lijst is,
- Naam voor E56 connector.
- Maximaal aantal E56 connectors.

De uitgangen zijn:

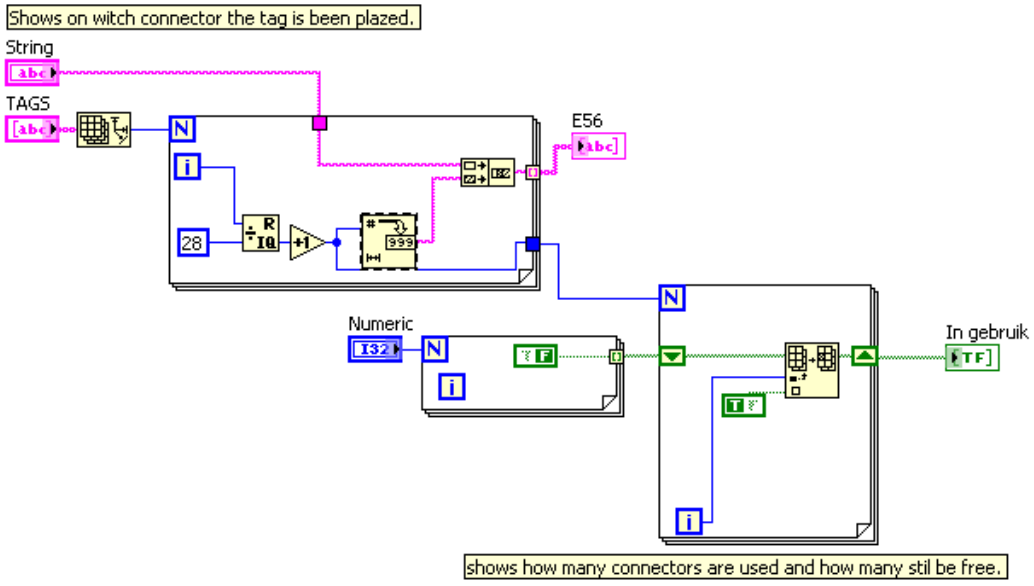
- E56 Lijst; Een weergave waarbij wordt aangegeven op welke connector de tag is geplaatst.
- In gebruik E56 connector; Geeft aan hoeveel connectors in gebruik zijn en hoeveel er nog beschikbaar zijn.

De werking is als volgt: de desbetreffende IO lijst komt binnen, hier wordt lengte van de lijst bepaald, dus hoeveel tags er in de lijst staan. Met deze waarde wordt aangegeven hoe vaak een for-loop[2. blz 56] moet worden uitgevoerd. Het nummer dat weergeeft hoe vaak een for-loop is geloopd wordt gedeeld door 28. Het getal wat hier uitkomt geeft aan welke connector in gebruik is. Dit getal wordt samengevoegd met de naam van de desbetreffende type E56 connector, dit kan zijn DI, AI, DO of AO.
De tweede for-loop wordt gebruikt om de een lijst van aantal E56 connectoren per IO lijst te genereren. De waarde die uit de deling komt wordt ook gebruikt voor een andere for-loop. Hiermee wordt een for-loop aangestuurd die aangeeft hoeveel E56 connectors in gebruik zijn.
In figuur 5.22 is een schematische werking te zien.



figuur 5.22. Schematische werking E56 SIZE.

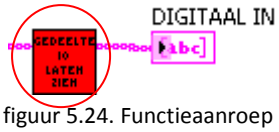
In figuur 5.23 is de broncode te zien.



figuur 5.23. Broncode E56 SIZE.

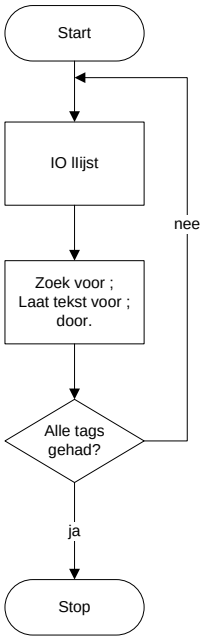
Gedeelte IO laten zien.

De IO tags bestaan uit een tag code, een nummer van een klemmenreeks en de klem nummers. Omdat voor het verdelen van de tags over de IO lijsten is het alleen maar van belang om de tag codes te zien en niet de klemmenreeks of klemnummers. Daar dient de functie Gedeelte IO laten zien voor. Zie figuur 5.24 voor de functieaanroep.



figuur 5.24. Functieaanroep Gedeelte IO laten zien.

De verschillende onderdelen zijn gescheiden door een ;. De lijst wordt in een for-loop geladen. Bij elke waarde wordt gezocht naar de eerste ; en de tekst voor de ; (de IO tag code) wordt doorgelaten, de rest wordt niet weergegeven. In figuur 5.25 is een schematische werking te zien. In figuur 5.26 is de broncode van de functie te zien.



figuur 5.25. Schematische weergave Gedeelte IO laten zien.

Figuur 5.26. Broncode Gedeelte IO laten zien.

Array highlight.

Het is de bedoeling dat de geselecteerde tag in een IO lijst gehighlight wordt. Hiermee wordt bedoelt dat als er een IO tag wordt geselecteerd, dat dit ook te zien is door middel van bijvoorbeeld een kader om de IO tag. De functie Array highlight zorgt hiervoor. De aanroep is te zien in figuur 5.27.

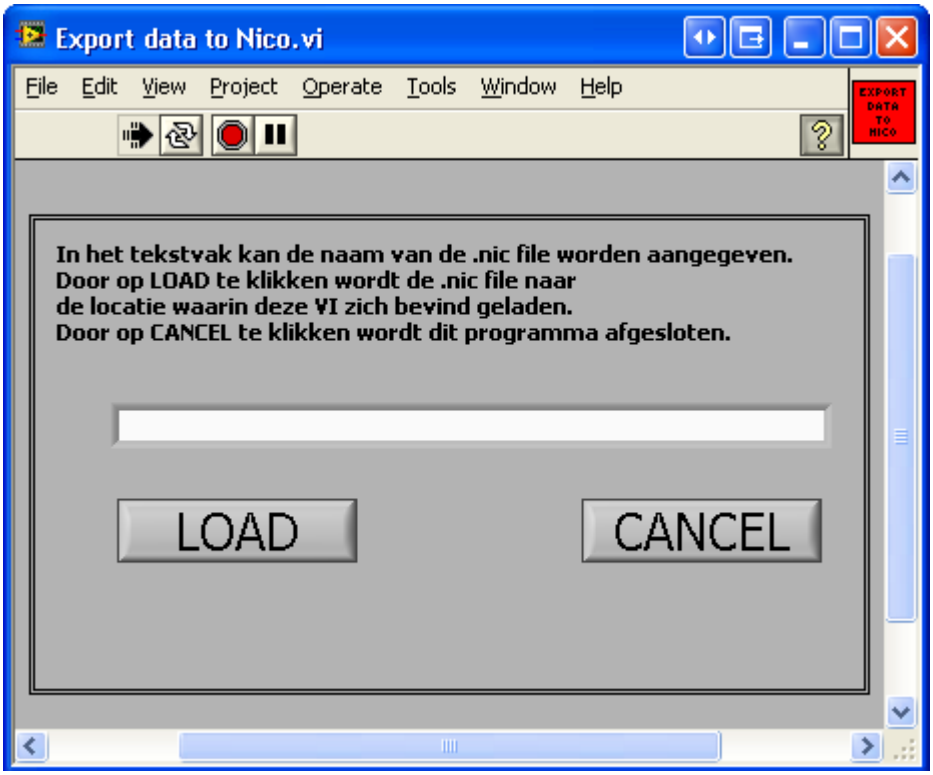
figuur 5.27. Functieaanroep Array Higlight.

Hij krijgt de element adres van de geselecteerde IO tag van de Move up, move down functie. Deze waarde wordt in een array gestopt op het eerste adres. Deze array wordt door een speciale functie in LabVIEW uitgelezen. Deze functie is gekoppeld aan de desbetreffende IO lijst en zorgt er voor dat de geselecteerde IO tag wordt “gehiglighted”.

5.2.3 Export data to hardware simulator.

In de laatste functieblok wordt de data exporteerd naar een .nic file. Deze file wordt ingelezen door de bestaand software. Als deze lijst in ingeladen zijn alle IO tags geconfigureerd.

Als de functie Export data to hardware simulator wordt aangeroepen komt de VI die in figuur 5.27 te zien is te voorschijn.



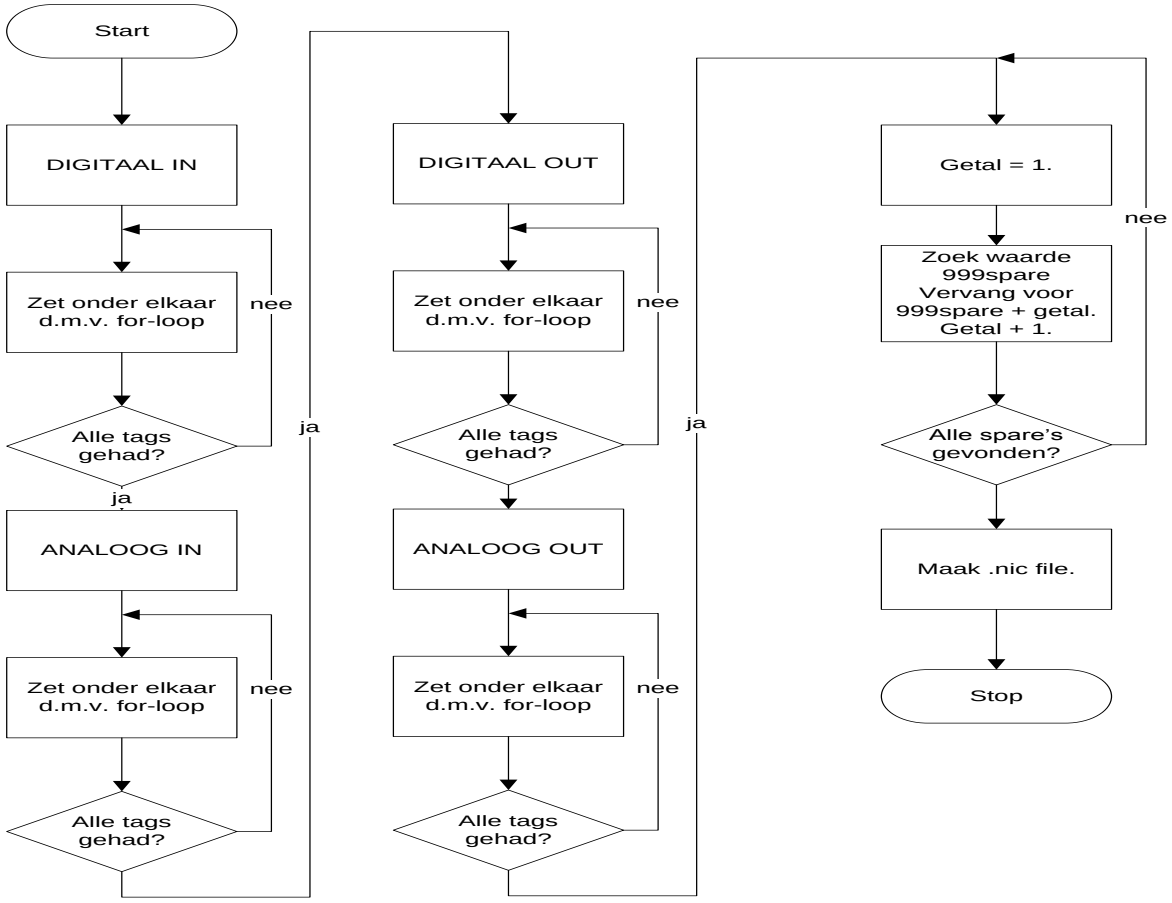
figuur 5.27. VI Export data to hardware simulator.

In een witte tekstvak kan een bestandsnaam worden opgegeven. Achter de bestandsnaam wordt de extensie .nic toegevoegd en dit bestand wordt opgeslagen op de plek deze VI staat opgeslagen.

Een voorbeeld van zo'n .nic file kan zijn:

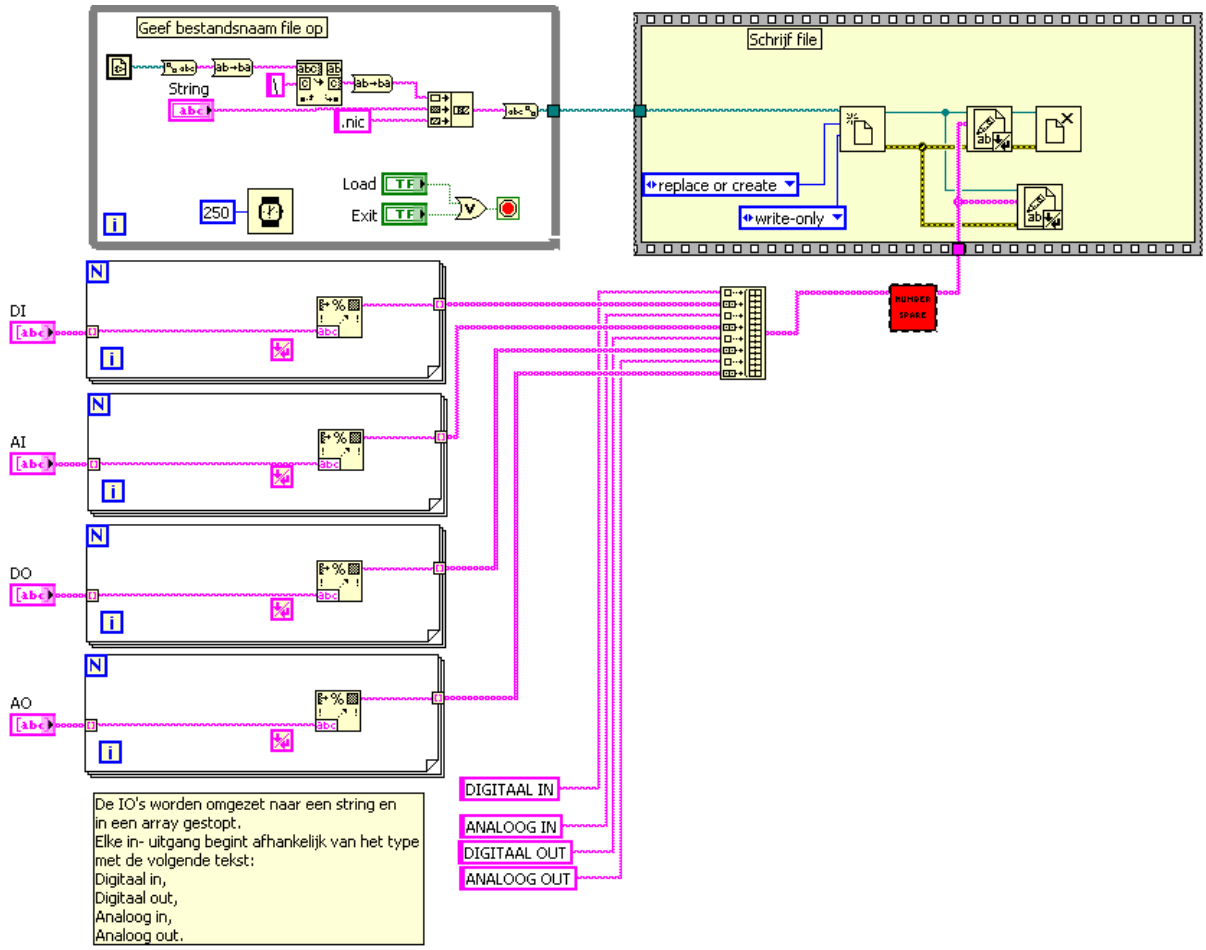
DIGITAAL IN
BP01-S1;113X2;5
BP01-S2;113X2;6
ANALOOG IN
BP01-P1;113X2;7
BP01-S1;113X2;5
DIGITAAL OUT
BP01-P1;113X2;8
BP01-P1;113X2;7
ANALOOG OUT
BP01-S2;113X2;6
BP01-P1;113X2;8
BP01-P1;113X2;7

De werking is als volgt: De tags worden onder elkaar gezet. Elke soort ingang wordt begonnen met respectievelijk DIGITAAL IN, ANALOOG IN, DIGITAAL OUT, ANALOOG OUT. Als er eventuele spares aanwezig zijn, worden deze vervangen door 999spare met een nummer. Dit nummer wordt doorgeteld als de er meerdere spares aanwezig zijn. In figuur 5.28 is een schematische weergave te zien.



figuur 5.28. Schematische weergave Export to hardware simulator.

In figuur 5.29 is de broncode te zien.



figuur 5.29. Broncode Export to hardware simulator.

5.3 Oplevering

Nadat de User-friendly IO configuratie software is geschreven kan de software worden opgeleverd. De software wordt aan de bedrijfsbegeleider opgeleverd. De software wordt samen met de bedrijfsbegeleider getest op functionaliteit. Zodra de software is getest en is goedgekeurd wordt deze in de bestaande software geïmplementeerd .

5.4 Conclusie

De software is naar tevredenheid ontworpen en goed geïmplementeerd in de bestaande software. Imtech Vonk is zeer tevreden met de software. Met deze software zal het configureren van de hardware simulator veel effectiever gaan.

6. Uitbreiding simulatie mogelijkheden

In de huidige situatie zijn de simulatie mogelijkheden gering. De simulatie in de huidige situatie gaat als volgt: er kunnen bijvoorbeeld 2 digitale ingangen worden gebruikt, A en B. Deze ingangen kunnen voor 1 functie gebruikt worden bijvoorbeeld de AND of OR functie. Bijvoorbeeld A and B = C

Dit geld ook voor de analoge ingangen. Hier kunnen ook maar 2 ingangen worden gebruikt bijvoorbeeld X en Y. Deze kunnen worden gebruikt voor de functies *, /, +, of -. Bijvoorbeeld X * Y = Z

De bedoeling is dat er meerdere functies/ formules achterelkaar kan worden gebruikt. Bijvoorbeeld met digitale ingangen: ((A AND B) OR (C AND D)) = E.

Of met analoge ingangen: ((X+Y)*(Q+ P)) = Z.

6.1 Ontwerp simulatie mogelijkheden

6.1.1 Standaard simulatie mogelijkheden

Om tot een goed software programma te komen en om te zorgen dat het software programma goed gaat werken moeten er afspraken worden gemaakt over de simulatie mogelijkheden.
Hierbij moet men denken aan: welke simulaties moeten er komen, hoe moeten deze simulaties worden in gevoerd (qua volgorde invoeren). Per type IO (digitaal of analoog) worden er nieuwe simulaties toegevoegd.

De volgende simulaties moeten worden toegevoegd bij de digitale IO’s:

- And functie,
- Or functie,
- Not functie,
- Xor functie,
- Nand functie,
- Nor functie,
- Nxor functie.

De volgende simulaties moeten worden toegevoegd bij de analoge IO’s:

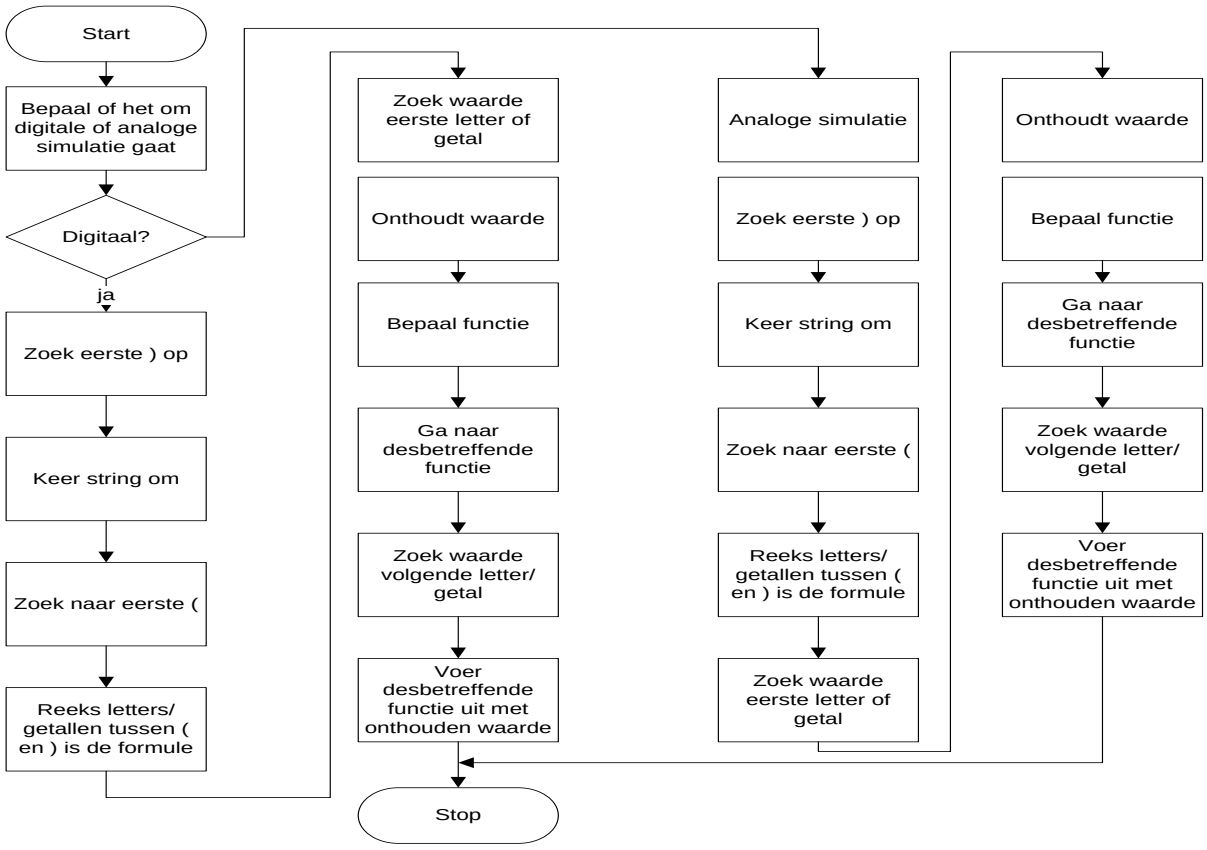
- +,
- -,
- *,
- /,
- x^n,
- $x^{1/n}$,
- sinus,
- cosinus,
- rand (random bereik kiezen),

Met deze simulatiemogelijkheden kunnen de meest uit één lopende situaties worden gesimuleerd. Er kunnen: kleppen, motoren, vloeistofniveaus in tanks, regelsignalen en temperaturen worden gesimuleerd.

Om misvattingen te voorkomen moeten er duidelijke afspraken worden gemaakt met oog op het invoeren van de simulaties/ formules. De volgende afspraken zijn gemaakt.

- De digitale en analoge functies moeten met HOOFDLETTERS worden geschreven,
- Functies en berekeningen dienen door middel van een spatie gescheiden te worden van getallen en of IO tags.
- Om te zorgen dat er bij berekeningen de regenregels worden opgevolgd moeten de berekeningen onderling worden gescheiden door haken,
- De functie RAND dient als volgt geschreven te worden: (10 * RAND 5) waarbij 10 het zogenaamde grond getal is en 5 zowel het positieve als negatieve bereik is (+5 en -5),
- De sinus of cosinus functie dient als volg geschreven te worden: (5 + SIN) of (5 * COS). Waarbij de + functie de sinus of cosinus in zijn geheel een x aantal plekken verplaatsen op de y as en de * functie zal de amplitude vergroten met een waarde x.

In de huidige situatie kan er maar één digitale functie worden gesimuleerd bijvoorbeeld Z = (A AND B). Het is de bedoeling de mogelijkheid moet komen dat Z aan meer voorwaarden moet doen bijvoorbeeld Z = ((A AND B) OR (C AND D)). Aan de hand van deze eisen wordt een flowchart gemaakt waarin de stappen van het programma te zien zijn. Per onderdeel wordt er een flowchart gemaakt.Voor de digitale en analoge simulaties worden de zelfde stappen doorlopen. In figuur 6.1 is de flowchart te zien.



figuur 6.1. Flowchart Digitale simulatie.

De stappen die de flowchart doorloopt zijn voor één berekening bijvoorbeeld (A AND B). De flowchart voert 1 berekening tussen (en) haken. Mocht de formule uit bijvoorbeeld ((A AND B) OR (C AND D)) bestaan wordt de berekening 3 keer uitgevoerd.
Per berekening wordt de waarde opgeslagen, als er nog een berekening komt wordt deze waarde mee genomen, als het de laatste berekening is wordt deze waarde de uitkomst van de simulatie.

Dezelfde principe geld voor de analoge simulatie. Per getal/letter of IO tag wordt de waarde uitgezocht. Deze waarde wordt onthouden en daarna wordt gezocht naar de volgende waarde of berekening. Dit wordt voor alle IO tags tussen de haken gedaan. Dus per (en) haken worden berekeningen uitgevoerd. Nadat alle (en) haken zijn afgewerkt rolt er een uitkomst uit de simulatie.

6.1.2 Speciale simulatie mogelijkheden

Om kleppen, oplopende volume eenheden te kunnen simuleren moeten er zogenaamde “specials” simulatie mogelijkheden komen.

De “specials” houden het volgende in:

Voor de digitale simulaties gaat het om vertragingen. Het vertraagd opkomen of vertraagd afvallend signaal. Hiermee kunnen bijvoorbeeld open en dicht meldingen van kleppen worden gesimuleerd. Als een klep wordt open gestuurd valt de dicht melding meteen weg, maar de open melding komt pas op als klep open is, dit moet dus vertraagd gebeuren.
Het zelfde geldt voor het dicht sturen van een klep. De open melding zal direct weg vallen, maar de dichtmelding komt pas op als de klep dicht is, dit moet ook vertraagd gebeuren.

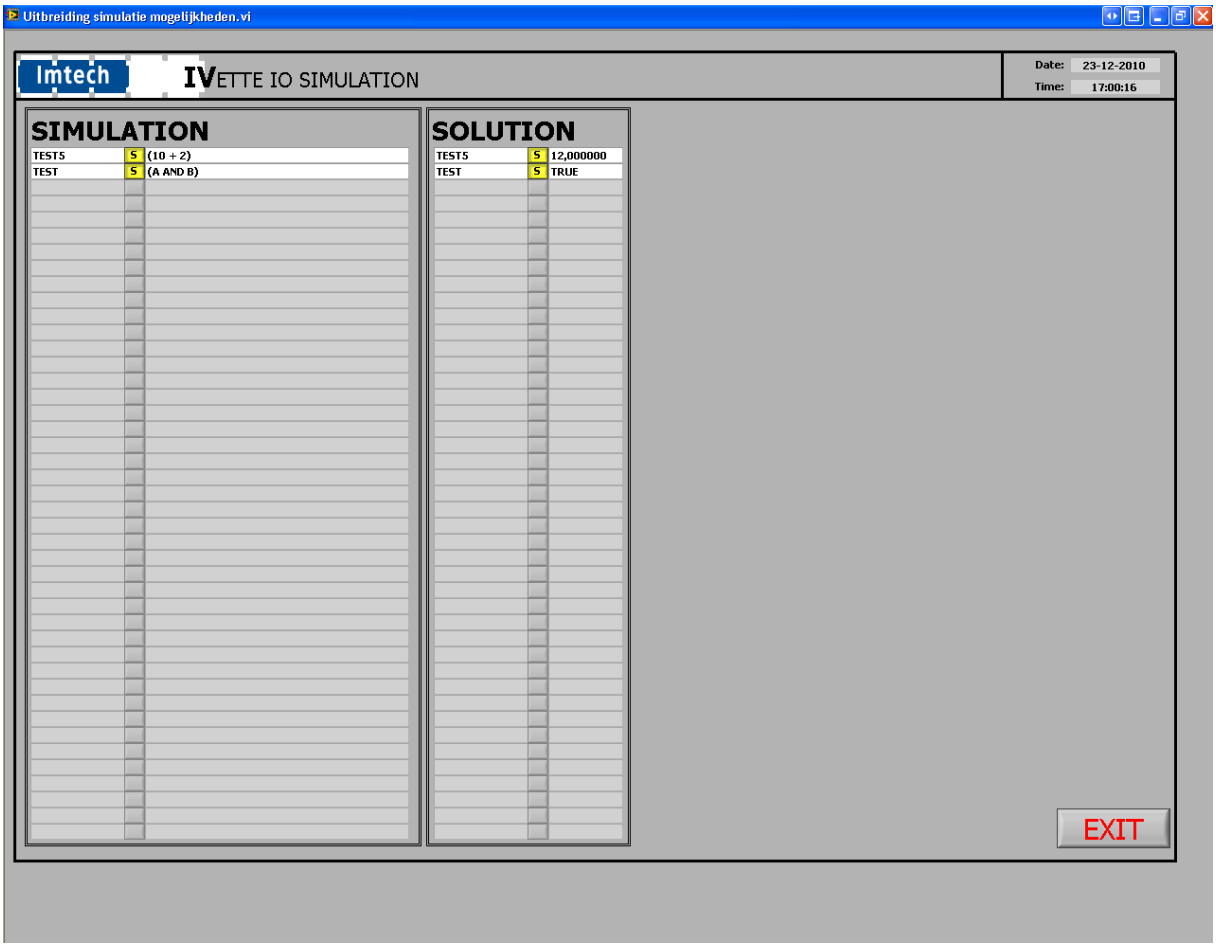
Voor analoog gaat het om het aansturen van signaal. Bijvoorbeeld het vullen van een tank met een vloeistof. Een tank is niet van het ene op het andere moment gevuld, dit duurt een tijd. Om dit te kunnen simuleren moet een analoge signaal ook in een bepaalde tijd oplopen in plaats van gelijk naar de eindwaarde te gaan.

Deze “specials” worden aangegeven met [en]. Voor digitaal wordt DON gebruikt voor vertraagd op komen en DOFF voor vertraagd afvallen.
Voor analoog wordt RAMP gebruikt.
Een digitale simulatie kan dan als volgt uitzien (A AND B) [DON 5]. Waarbij 5 voor het aantal seconden staat voordat de uitkomst wordt weergegeven.
Een analoge simulatie kan er als volgt uit zien (K * 2) [RAMP 5]. Waarbij de waarde van de RAMP de eenheid %/seconde heeft. In het voorbeeld zal de waarde 5% van de eindwaarde per seconde stijgen.
In het begin wordt gekeken of er een “special” moet worden uitgevoerd. Als een “special” moet worden uitgevoerd wordt dit gedaan aan de einde van de simulatie, word er geen “special” uitgevoerd dan wordt de uitkomst van de simulatie onveranderd doorgestuurd.

6.2 Uitwerking simulatie mogelijkheden

Nadat de ontwerpeisen bekend zijn het kan de software worden geschreven.

De simulatie VI is te zien in figuur 6.2.



figuur 6.2. VI Simulatie mogelijkheden.

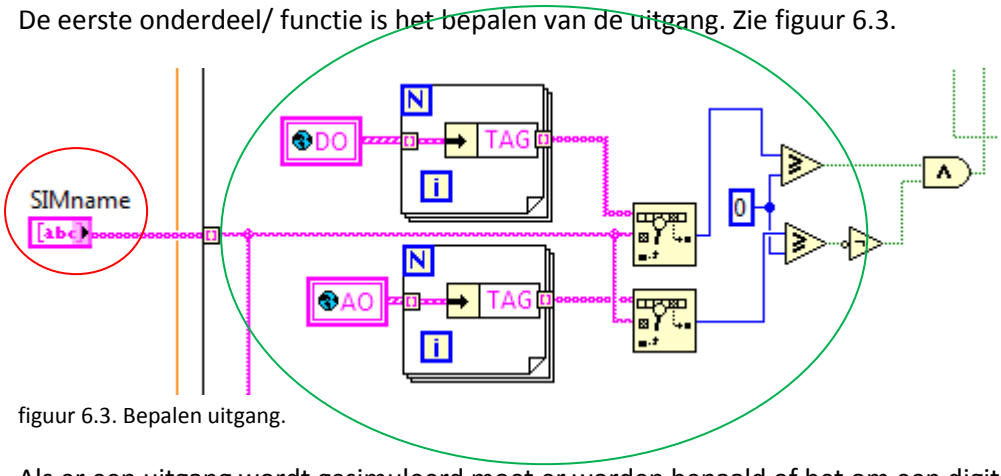
In figuur 6.2 zijn de kolommen SIMULATION, SOLUTION en VALVES te zien. In de kolom SIMULATION zijn 3 rijen te zien. In de eerste rij wordt aangegeven welke uitgang moet worden aangestuurd. Aan de hand van de ingevulde TAG zoekt het programma de desbetreffende uitgang op.
In de tweede rij word aangegeven of de simulatie moet worden uitgevoerd of niet.
In de derde rij wordt de simulatie formule weer gegeven. In deze rij kan de simulatie formule worden weergegeven.

De tweede kolom is de SOLUTION. In deze kolom komt de oplossing van de simulatie te staan.
In de eerste rij komt de desbetreffende uitgang als indicatie te staan.
In de tweede rij komt staat of de simulatie wordt uitgevoerd of niet.
De laatste rij geeft de uitkomst van de simulatie weer. Wordt de simulatie uitgevoerd, dan wordt de huidige uitkomst weergegeven, wordt de simulatie niet uitgevoerd dan wordt voor digitaal een FALSE doorgegeven en voor analoog een 0 doorgegeven.

Het derde vak “VALVES” zijn de klepsimulaties. Hier in kunnen samen met de eerste kolom “SIMULATION” kleppen worden gesimuleerd.

Per functieonderdeel wordt de broncode uitgelegd.

De eerste onderdeel/ functie is het bepalen van de uitgang. Zie figuur 6.3.



figuur 6.3. Bepalen uitgang.

Als er een uitgang wordt gesimuleerd moet er worden bepaald of het om een digitale of analoge uitgang gaat. De te simuleren uitgang staat in de array SIMname (zie rode cirkel). Door middel van een for-loop wordt voor elke uitgang gekeken om wat voor een uitgang het gaat.
Dit wordt gedaan in het gebied dat is omcirkeld met een groene cirkel.
Doormiddel van een for-loop wordt gezocht door de DO en AO lijsten. Elke waarde in deze lijst wordt vergeleken met de te simuleren uitgang.
Als de uitgang is gevonden wordt de postie van de uitgang in de lijst doorgegeven. Aan de hand van de positie kan met bepalen of het een digitale of analoge uitgang is.
In figuur 6.4 is een schematische weergave te zien.

Bepalen of er een "special" in de formule zit.

De volgende stap is de simulatie zelf. De simulatieformule zelf wordt tussen (en) geschreven. Een eventuele "special" wordt achter de) geschreven tussen [en]. Een formule ziet er dan uit met special: (*formule*) [*special*] of zonder special: (*formule*).

Om te bepalen of er een special in de formule zit is er de functieblok () [] scheiden. In figuur 6.5 is de functie aanroep te zien.

figuur 6.5. Functieaanroep ()[] scheiden.

De functie zal kijken of er een] in voorkomt, komt er een] in voor dan wordt de formule tussen (en) staat en de formule die tussen [en] staat apart van elkaar doorgestuurd. De formule tussen (en) wordt uitgevoerd en over de uitkomst wordt de "special"die tussen de [en] staat uitgevoerd.

Mocht er geen] in de formule voorkomen dan wordt de formule tussen (en) in zijn geheel doorgegeven.

In figuur 6.6 is een schematische weergave te zien van de werking.

Nadat bekend is om welke uitgang het gaat (digitaal of analoog) en bekend is of er een "special" wordt uitgevoerd, wordt de simulatie uitgevoerd. Dit kan om een digitale of analoge simulatie gaan.

Zowel de digitale als de analoge simulatie werkt op de volgende manier:

Een formule/ simulatie kan uit meerdere (en) haken bestaan. Per (en) haak wordt er een berekening uitgevoerd. De uitkomst van die berekening vervangt de formule die tussen de (en) haken staat. Zo worden alle (en) haken bij langs gegaan tot dat er een eind een uitkomst van de simulatie over blijft.

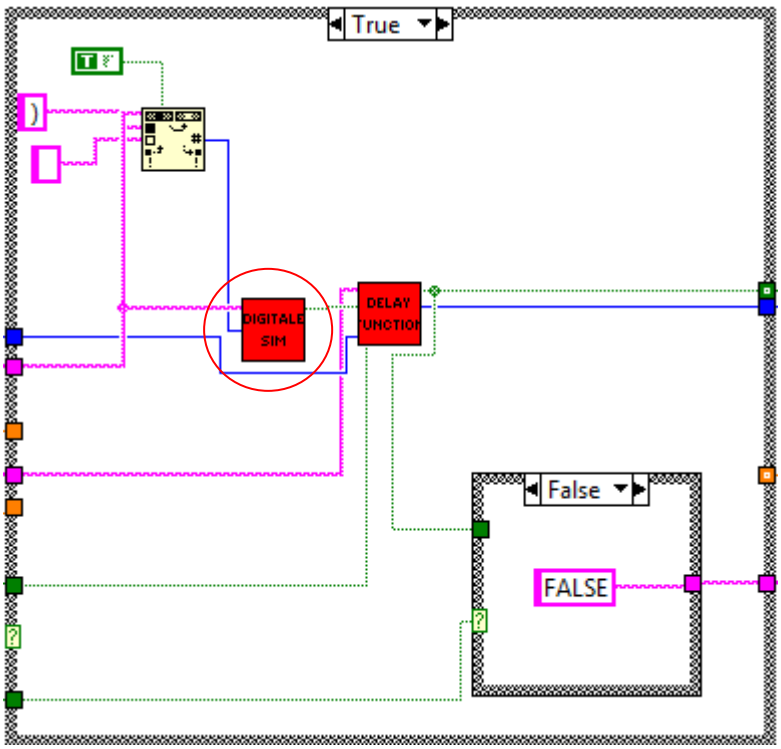
De digitale en analoge simulatie worden hieronder verder beschreven. De beschrijving zal gaan over 1 berekening tussen (en). Mocht de formule er als volgt uitzien: ((A AND B) OR (C AND D)) dan wordt de digitale simulatie 3 keer uitgevoerd.
Een schematische werking is te zien in figuur 6.6.

6.2.1 Digitale simulatie

De volgende functies komen voor bij de digitale simulaties:

- And functie,
- Or functie,
- Not functie,
- Xor functie,
- Nand functie,
- Nor functie,
- Nxor functie.

In figuur 6.7 is de aanroep van de functie te zien.



figuur 6.7. Functie aanroep digitale simulatie.

De functie krijgt als ingang de formule code en het aantal keer dat de functie een loop moet doorlopen. De uitgang is de uitkomst van de simulatie. De uitgang van de simulatie is de uitkomst van de berekening.

De werking van de digitale simulatie is als volgt: als voorbeeld is de formule ((A AND B) OR (C AND D)), hierbij zijn: A = TRUE en B = TRUE, C = TRUE en D = FALSE.

Stap 1:
Er wordt gezocht naar het eerst). Daarna wordt de string omgekeerd en daarna gezocht naar de eerste (. Hiermee is het eerste stuk uit de formule genomen. Hier wordt de uitkomst van berekend en wordt het stuk (*formule*) vervangen door de uitkomst. Het berekenen van de formule wordt in stap 2 beschreven.

Stap 2:
De formule wordt als volgt berekend:
Het programma begint in de status "0". In de status "0" kunnen bepaalde situaties voorkomen, deze situaties zijn als volgt gedefinieerd:

In de status "0" is de functie "NOT" met als voorwaarden TRUE en FALSE, deze status leest een shift register uit (is register leeg of staat er wat in), bevat deze shift register een waarde dan zal afhankelijk van deze waarde de NOT functie worden uitgevoerd.
De waardes die in deze shift register staan zijn de digitale functies: AND, OR, XOR, NAND, NOR, XNOR. De NOT functie zal dan over de laatste IO tag in de formule worden uitgevoerd. Is de shift register leeg dan is er geen NOT functie (status TRUE) en gaat de functie naar de volgende situatie.

De volgende situatie is kijken wat voor een type karakter tot aan de eerste spatie staan in de string. De karakters kunnen er als volgt uit zien:

- Unknown: hierbij gaat het om een IO tag, de IO tag wordt dan op gezocht in de DI en DO lijsten en de desbetreffende status (TRUE of FALSE) wordt hierbij opgezocht. Het programma gaat dan weer naar situatie "0".
- 0: er kan een waarde 0 zijn ingevuld in de formule. De waarde FALSE wordt dan weggeschreven en het programma gaat dan weer naar de situatie "0".
- 1: er kan een waarde 1 zijn ingevuld in de formule. De waarde TRUE wordt dan weggeschreven en het programma gaat dan weer naar de situatie "0"
- AND: hier gaat het om de AND functie. De simulatie zal naar de situatie "1" gaan. Deze situatie wordt later in de scriptie verder uitgelegd.
- OR: hier gaat het om de OR functie. De simulatie zal naar de situatie "2" gaan. Deze situatie wordt later in de scriptie verder uitgelegd.
- NOT: hier gaat het om de OR functie. De simulatie zal naar de situatie "3" gaan. Deze situatie wordt later in de scriptie verder uitgelegd.
- XOR: hier gaat het om de XOR functie. De simulatie zal naar de situatie "4" gaan. Deze situatie wordt later in de scriptie uitgelegd.
- NAND: hier gaat het om de NAND functie. De simulatie zal naar de situatie "5" gaan. Deze situatie wordt later in de scriptie verder uitgelegd.
- NOR: hier gaat het om de NOR functie. De simulatie zal naar de situatie "6" gaan. Deze situatie wordt later in de scriptie verder uitgelegd.

- XNOR: hier gaat het om de XNOR functie. De simulatie zal naar de situatie “7” gaan. Deze situatie wordt later in de scriptie verder uitgelegd.

Situatie “1” is de AND functie. Hierbij is al een waarde van het eerste karakter van de string gevonden.
In deze situatie kunnen 3 situaties voorkomen.

De eerste situatie is Unkwown. Dit houdt in dat er een AND functie moet worden uitgevoerd tussen twee IO tags of tussen een waarde 1 of 0 en een IO tag. De waarde van de eerste IO tag/ status 1 of 0. is al bekend. De waarde van de laatste IO tag wordt gezocht in de lijsten DI en DO en over deze twee waardes wordt de AND functie uitgevoerd.

De tweede situatie is de NOT situatie. Dit houdt in dat over de tweede IO tag eerst de not functie moet worden uitgevoerd. Er wordt AND weggeschreven in de shiftregister. Situatie “0” maakt hier verder gebruik van.
De 3de situatie is 0. Dit beteken dat er een 0 in de formule staat. Deze waarde is FALSE en wordt de eerste waarde samen met de waarde FALSE gebruikt voor de AND functie.

De 4de situatie is 1. Dit betekend dat er een 1 in de formule staat. Deze waarde is TRUE en wordt samen met de eerste waarde gebruikt voor de AND functie.

De situaties 2 t/m 7 werken op de zelfde manier als situatie 1. Het enige verschil is dat er andere functies worden gebruikt: situatie 2 is de OR functie, situatie 3 is de NOT functie, situatie 4 is de XOR functie, situatie 5 is de NAND functie, situatie 6 is de NOR functie en situatie 7 is de XNOR functie.

Zo worden alle eenheden in de formule bijlang gegaan om hun waarde te bepalen.
Om terug te komen op het voorbeeld ((A AND B) OR (C AND D)), komt de simulatie als volgt uit te zien:

Stap 1:
De eerste) haak is degene naar de B. Met de gedeelte voor deze haak word verder gewerkt (((A AND B). De string wordt omgekeerd (B DNA A (( ) en er wordt gezocht naar de eerste (haak. Nadat dit gevonden is wordt het gedeelte weer omgekeerd en krijg je de functie A AND B.

Stap 2:
Hierbij wordt gezocht naar de eerste spatie. Het stuk wat er voor zit is in dit geval de A.
Omdat A geen 0, 1 of de functie NOT is gaat het om een IO tag, ook wel situatie Unkown. A wordt opgezocht in de DI en DO lijsten, in dit voorbeeld heeft A de waarde TRUE. Deze waarde wordt onthouden. De simulatie gaat nu verder met het stuk wat naar A + spatie zit.
De simulatie gaat weer opzoek naar de eerste spatie. Het stuk wat voor de eerste spatie zit word nu bekeken. Dit is in dit geval AND. AND heeft de situatie 1, de simulatie gaat naar situatie 1. Hierbij wordt de AND + spatie verwijderd uit de formule.

In situatie 1 wordt gekeken naar de volgende spatie. In dit geval is er geen spatie dus de simulatie zal neemt het overbleven stuk van de formule. Dat is in dit geval B. Hier wordt ook de waarde van opgezocht, in dit geval TRUE. Nu zijn beide waardes bekend en wordt van TRUE en TRUE de AND functie berekend. De uitkomst is in dit geval TRUE.De uitkomst van A AND B is dan TRUE, dit wordt vervangen door 1 en in de formule geschreven in plaats van (A AND B). De formule wordt nu (1 OR (C AND D)).

Nu wordt de 1ste stap weer herhaald. Er wordt gezocht naar het eerste) haak. Het stuk formule dat voor de) haak zit wordt gebruikt en omgekeerd “D DNA C(RO 1(“. Er wordt daarna gezocht naar de eerste (. Het stuk ertussen wordt gebruikt “D DNA C(“ en omgekeerd (C AND D. De (haak wordt verwijdert en er wordt gerekend met C AND D.

Stap 2:

Hierbij wordt gezocht naar de eerste spatie. Het stuk wat er voor zit is in dit geval de C.
Omdat C geen 0, 1 of de functie NOT is gaat het om een IO tag, ook wel situatie Unkown. C wordt opgezocht in de DI en DO lijsten, in dit voorbeeld heeft C de waarde TRUE. Deze waarde wordt onthouden. De simulatie gaat nu verder met het stuk wat naar C + spatie zit.
De simulatie gaat weer opzoek naar de eerste spatie. Het stuk wat voor de eerste spatie zit word nu bekeken. Dit is in dit geval AND. AND heeft de situatie 1, de simulatie gaat naar situatie 1. Hierbij wordt de AND + spatie verwijderd uit de formule.

In situatie 1 wordt gekeken naar de volgende spatie. In dit geval is er geen spatie dus de simulatie zal neemt het overbleven stuk van de formule. Dat is in dit geval D. Hier wordt ook de waarde van opgezocht, in dit geval FALSE. Nu zijn beide waardes bekend en wordt van TRUE en FALSE de AND functie berekend. De uitkomst is in dit geval FALSE. De uitkomst van C AND D is FALSE, dit wordt vervangen door 0 en in de formule geschreven in plaats van (C AND D). De formule wordt nu (1 OR 0).

Nu wordt weer de 1ste stap herhaald. De uitkomst zal 1 OR 0 worden.

Stap 2:

Hierbij wordt gezocht naar de eerste spatie. Het stuk wat er voor zit is in dit geval de waarde 1.
1 is een voorgegeven waarde. De waarde 1 staat voor TRUE. Deze waarde wordt onthouden en de simulatie gaat nu verder met het stuk wat naar 1 + spatie zit.
De simulatie gaat weer opzoek naar de eerste spatie. Het stuk wat voor de eerste spatie zit word nu bekeken. Dit is in dit geval OR. OR heeft de situatie 2, de simulatie gaat naar situatie 2. Hierbij wordt de OR + spatie verwijderd uit de formule.

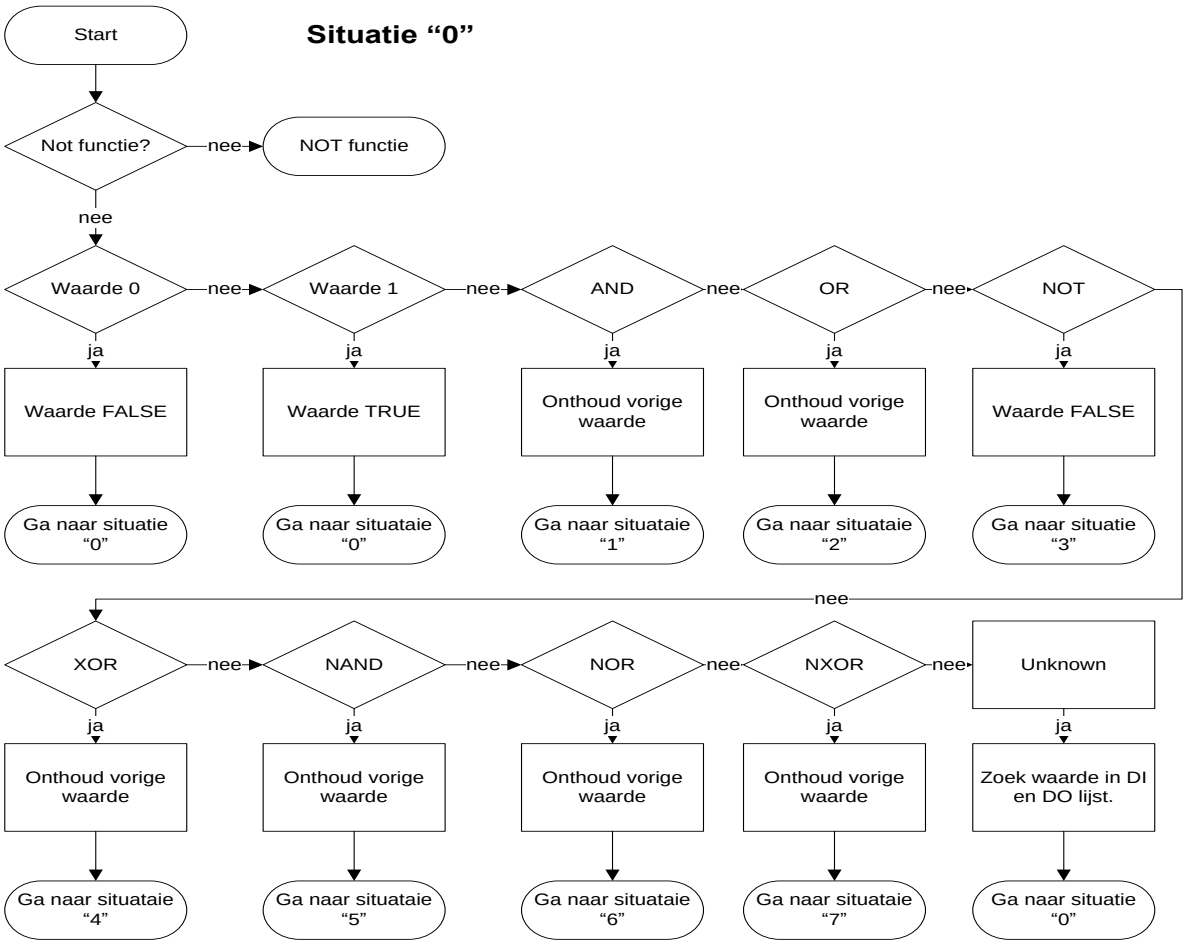
In situatie 2 wordt gekeken naar de volgende spatie. In dit geval is er geen spatie dus de simulatie zal neemt het overbleven stuk van de formule. Dat is in dit geval 0. 0 = de waarde false. Nu zijn beide waardes bekend en wordt van TRUE en FALSE de OR functie berekend. De uitkomst is in dit geval TRUE.

De hele formule ((A AND B) OR (C AND C)) is bij langs gegaan en de uitkomst van de simulatie is TRUE.

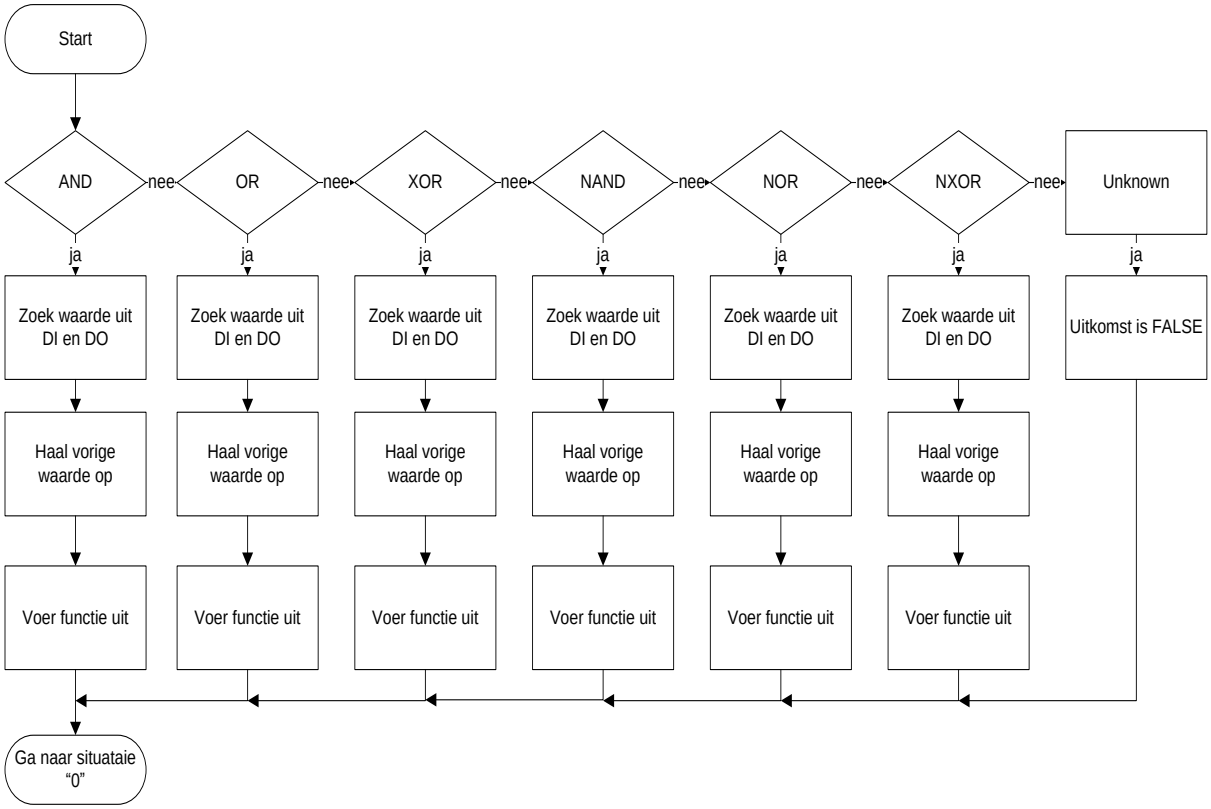
Een schematische werking is te zien in de onderstaande figuren te zien.
In figuur 6.8 is situatie “0” te zien.

In figuur 6.9 is de NOT functie te zien.
In figuur 6.10 is de situatie “1” te zien. Deze flowchart geld ook voor de situaties 2 t/m 7.

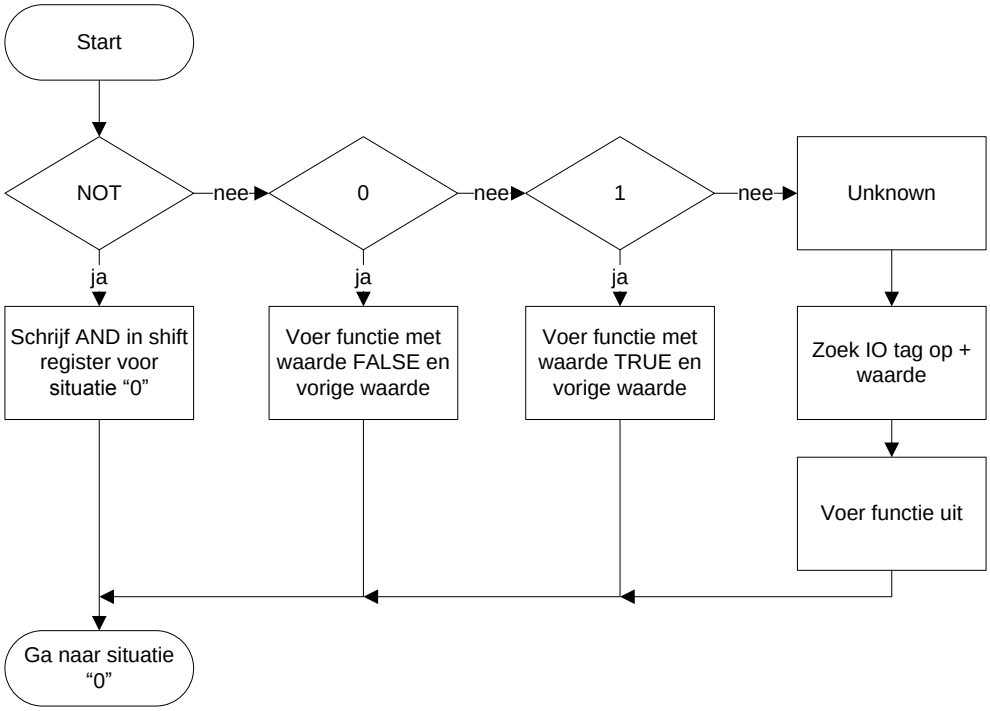
Omdat de broncode vrij groot en complex is, is de broncode weergegeven in bijlage D.



figuur 6.8. Flowchart situatie “0”.



figuur 6.9. Flowchart NOT functie.



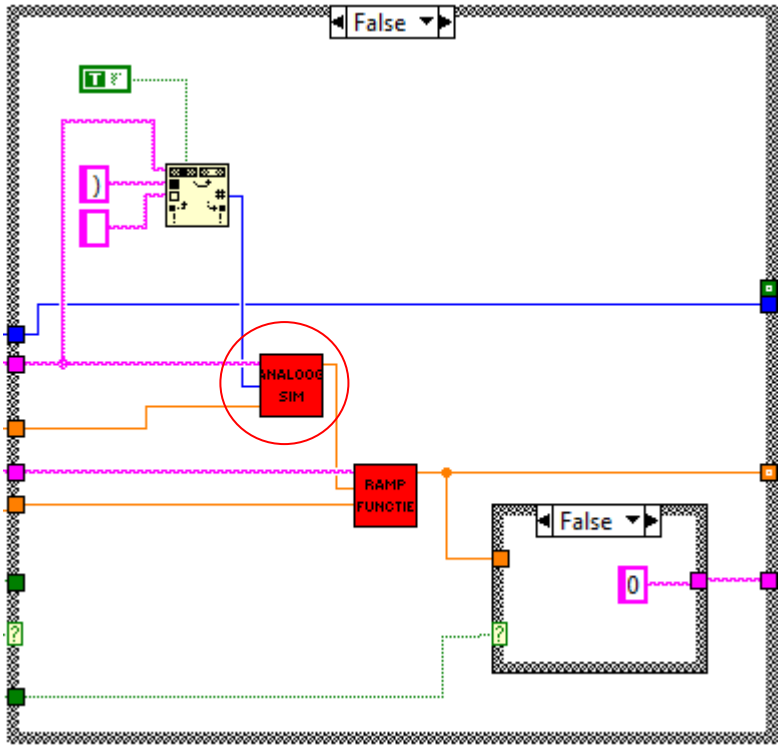
figuur 6.10. Flowchart situatie "1".

6.2.2 Analoge simulaties.

De volgende simulaties zijn de analoge simulaties. De analoge simulaties bestaan uit de volgende functies:

- +,
- -,
- *,
- /,
- x^n,
- rand,
- sinus,
- cosinus.

In figuur 6.11 is de functie aanroep te zien.



figuur 6.11. Functieaanroep analoge simulaties.

De functie krijgt als ingang de formule code, het aantal π/radiale voor de sinus en cosinus en het aantal keer dat de functie een loop moet doorlopen. De uitgang is de uitkomst van de simulatie. De uitgang van de simulatie is de uitkomst van de berekening.

De werking van de analoge simulatie is als volgt: als voorbeeld is de formule ((E + 3) * F). Waarbij: E = 2 en F = 2.

De analoge simulatie werkt op de zelfde principe als de digitale simulatie, alleen maakt de analoge simulatie gebruik van getallen in plaats van booleaans waardes.

Stap 1:

Er wordt gezocht naar het eerst). Daarna wordt de string omgekeerd en daarna gezocht naar de

eerste (. Hiermee is het eerste stuk uit de formule genomen. Hier wordt de uitkomst van berekend en wordt het stuk (*formule*) vervangen door de uitkomst. Het berekenen van de formule wordt in stap 2 beschreven.

Stap 2:

De formule wordt als volgt berekend:
Het programma begint in de status "0". In de status "0" kunnen bepaalde situaties voorkomen, deze situaties zijn als volgt gedefinieerd:

In de status "0" kan zich voordoen dat de formule niet begint met een IO tag maar met een getal. Mocht de formule beginnen met een getal dan wordt dit getal in een shiftregister gezet en gaat de simulatie weer naar heb begin van situatie "0". Mocht de formule niet beginnen met een getal maar met een IO tag dan volgen in deze situatie meerdere situaties. Als het niet om een getal in de formule gaat kan het om het volgende gaan:

- Unknown: het gaat om een IO tag, De IO tag wordt opgezocht in de AI en AO lijst. Als de IO tag is gevonden wordt de desbetreffende waarde erbij opgezocht. De simulatie gaat dan weer naar de situatie "0".
- +: het gaat hier om de + functie. De simulatie zal naar situatie "1" gaan. De situatie wordt later in dit scriptie uitgewerkt.
- -: het gaat om de – functie. De simulatie zal naar situatie "2" gaan. De situatie wordt later in dit scriptie uitgewerkt.
- *: het gaat om de * functie. De simulatie zal naar situatie "3" gaan. De situatie wordt later in dit scriptie uitgewerkt.
- /: het gaat om de / functie. De simulatie zal naar situatie "4" gaan. De situatie wordt later in dit scriptie uitgewerkt.
- ^: het gaat om de x^n functie. De simulatie zal naar de situatie "5" gaan. De situatie wordt later in dit scriptie uitgewerkt.

Situatie "1" is de + functie. Er is dan al een waarde van, of een IO tag of een getal in de formule bekend. Er kan het volgende bekend zijn, bijvoorbeeld 2 + … Het … is dan de rest van de formule wat nog onbekend is.

> In deze situatie kunnen zich weer meerdere situaties voordoen. De eerste situatie is, er is een getal in de formule. … kan een getal zijn bijvoorbeeld 2. Is dit het geval dan wordt de functie + uitgevoerd met als ingangen 2 en 2. De uitkomst is dan 4.
>
> Is … geen getal, dan … de volgende betekenis hebben:
>
> - Unknown: het gaat hier om een IO tag, de IO tag wordt opgezocht in de AI en AO lijst en de desbetreffende waarde die erbij hoort wordt opgeteld met de waarden die voor de + functie stond.
> - SIN: dit betekend dat er een sinus functie wordt gevraagd. De waarde die voor de + functie stond wordt opgeteld bij de sinus functie.
> - COS: dit betekend dat er een cosinus functie wordt gevraagd. De waarde die voor de + functie stond wordt opgeteld bij de sinus functie.

- RAND: dit is de rand functie. De rand krijgt een getal mee. De RAND functie geeft een random getal tussen het + en – van het meegegeven getal. De simulatie gaat naar situatie “6”.

Situatie “2” is de – functie. Er is dan al een waarde van, of een IO tag of een getal in de formule bekend. Er kan het volgende bekend zijn bijvoorbeeld 2 - … Het … is dan de rest van de formule wat nog onbekend is.

> In deze situatie kunnen zich twee situaties voordoen. De eerste situatie is, er is een getal in de formule. Als er een getal in de formule zit wordt dit getal met de vorige getal voor de – functie gebruikt als ingangen voor de – functie.
>
> Als … geen getal is gaat het om een IO tag. de IO tag wordt opgezocht in de AI en AO lijst en de desbetreffende waarde die erbij hoort wordt van de waarde die voor het – teken stond afgetrokken.

Situatie “3” is het zelfde als situatie “1”. Het verschil is echter dat het bij situatie “3” om de * functie in plaats van de + functie. De stappen zijn het zelfde.

De situaties 2, 4 t/m 7 werken op de zelfde manier als situatie 1. Het enige verschil is dat er andere functies worden gebruikt: situatie 2 is de - functie, situatie, situatie 4 is de / functie, situatie 5 is de x^n functie, situatie 6 is de + RAND functie en situatie 7 is de * RAND functie.

Zo worden alle eenheden in de formule bijlang gegaan om hun waarde te bepalen.
Om terug te komen op het voorbeeld ((E + 3) * F), komt de simulatie als volgt uit te zien:

Stap 1:
De eerste) haak is degene naar de 3. Met de gedeelte voor deze haak word verder gewerkt ((E + 3. De string wordt omgekeerd 3 + E ((en er wordt gezocht naar de eerste (haak. Nadat dit gevonden is wordt het gedeelte weer omgekeerd en krijg je de functie E + 3.

Stap 2:
Hierbij wordt gezocht naar de eerste spatie. Het stuk wat er voor zit is in dit geval de E. E is geen getal en geen functie +, -, * of /.
Het gaat dus om een getal. E wordt opgezocht in de AI en AO lijsten. In dit voorbeeld heeft E de waarde 2. Deze waarde wordt onthouden. E en de spatie worden verwijderd. Nu blijft er nog + 3 over.
Er wordt weer gezocht naar de eerste spatie. Het stuk wat voor de eerste spatie zit is de +. De simulatie gaat nu naar situatie “1”. De + en spatie worden verwijderd.

In situatie “2” bevinden zich (zoals hierboven uitgelegd) meerdere situaties. In dit voorbeeld gaat het om het getal 3. De simulatie zal dan ook de waarde 2 (van E) bij het getal 3 optellen. De uitkomst is 5. De gedeelte van E + 3 wordt vervangen door 5.
De formule komt dan als volgt eruit te zien: (5 * F).

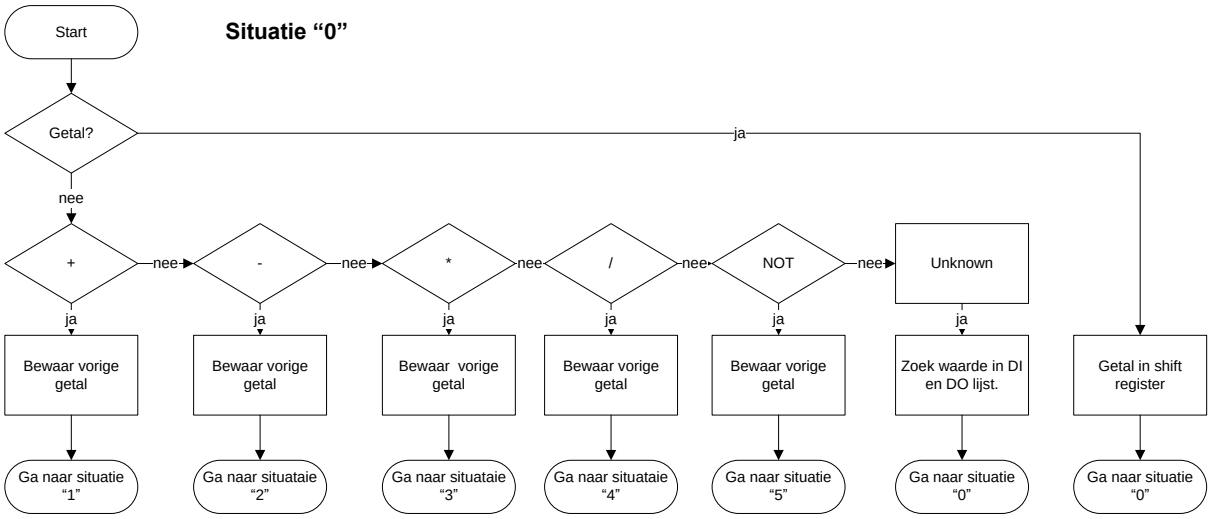
Nu herhaald de simulatie weer, er wordt gezocht naar het eerste) haak, de string wordt omgekeerd, er wordt gezocht naar de eerste (haak. Het stuk daartussen wordt mee gerekend. In dit geval is dat 5 * F.

De simulatie begint weer in situatie “0”. Er wordt gezocht naar de eerste spatie. In dit geval is het stuk ervoor 5. Dit getal wordt onthouden en + de spatie verwijderd.
Er wordt weer gezocht naar de eerste spatie. Het stuk wat er nu voor zit is *. De simulatie gaat nu naar situatie “3”.
In situatie “3” kan het om een getal gaan of een andere type waarde. Als het niet om een getal gaat (wat in dit geval is) kan het om een IO tag, een sinus, een cosinus of een RAND gaan. In dit geval gaat het om een IO tag. Deze IO tag wordt dan ook in de AI en AO lijsten opgezocht. De desbetreffende waarde wordt erbij gezocht. In dit geval is dat 2.
De uitkomst van E + 3 => 5 wordt vermenigvuldigt met 2. De uitkomst van de formule ((E + 3) * F) is dan ook 10.

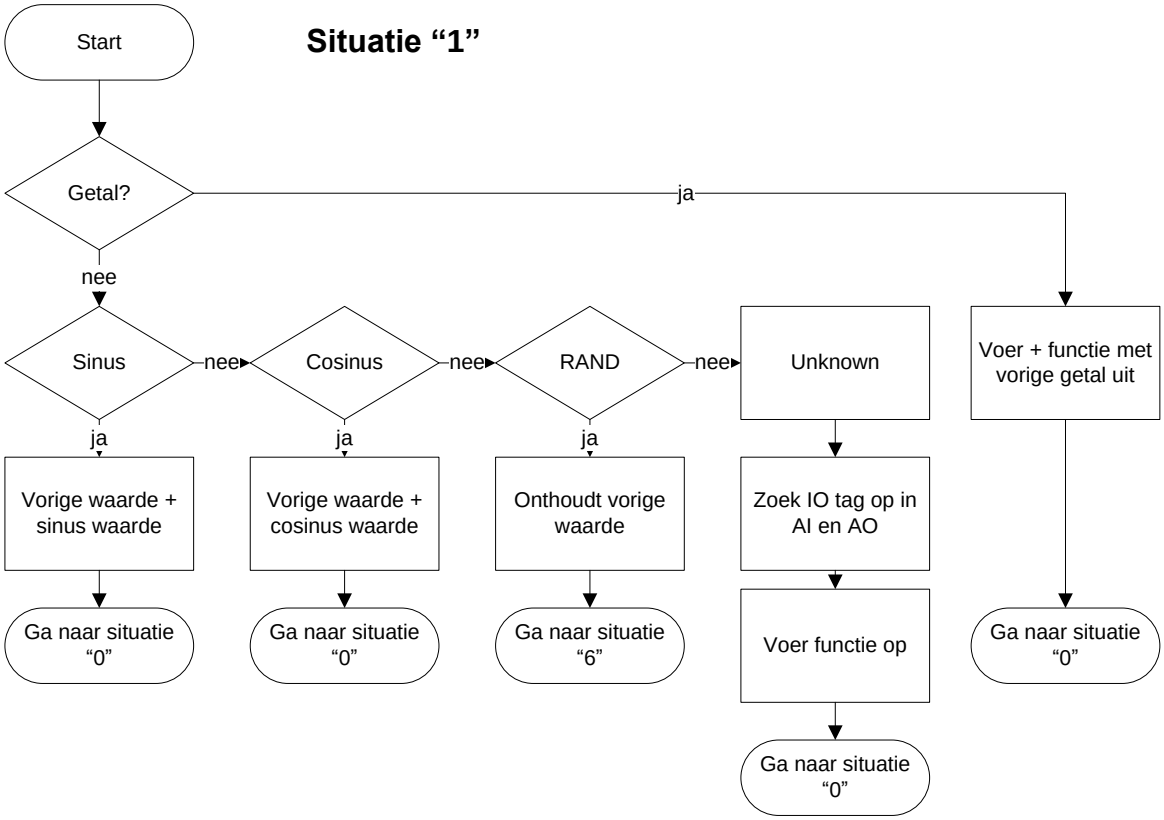
In de onderstaande figuren is de een schematische werking van de analoge simulatie te zien.
In figuur 6.12 is een schematische werking te zien van situatie “0”.
In figuur 6.13 is een schematische werking te zien van situatie “1”.
In figuur 6.14 is een schematische werking te zien van situatie “2”.

Omdat situatie “3” hetzelfde is als situatie “1” en situatie “4” t/m “7” hetzelfde zijn als situatie “2” (de functie is alleen anders, de stappen zijn het zelfde), zijn alleen situatie “1” en “3” weergegeven.

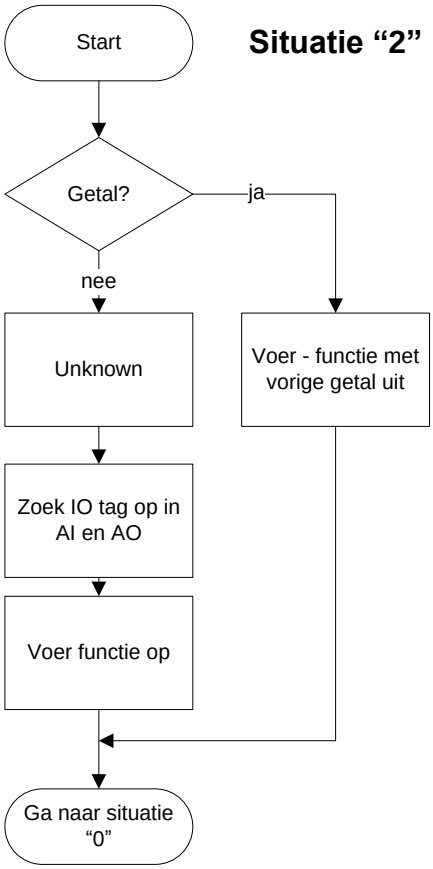
Omdat de broncode vrij groot en complex is, is de broncode weergegeven in bijlage D.



figuur 6.12. Schematische werking situatie “0”.



figuur 6.13. Schematische werking situatie “1”.

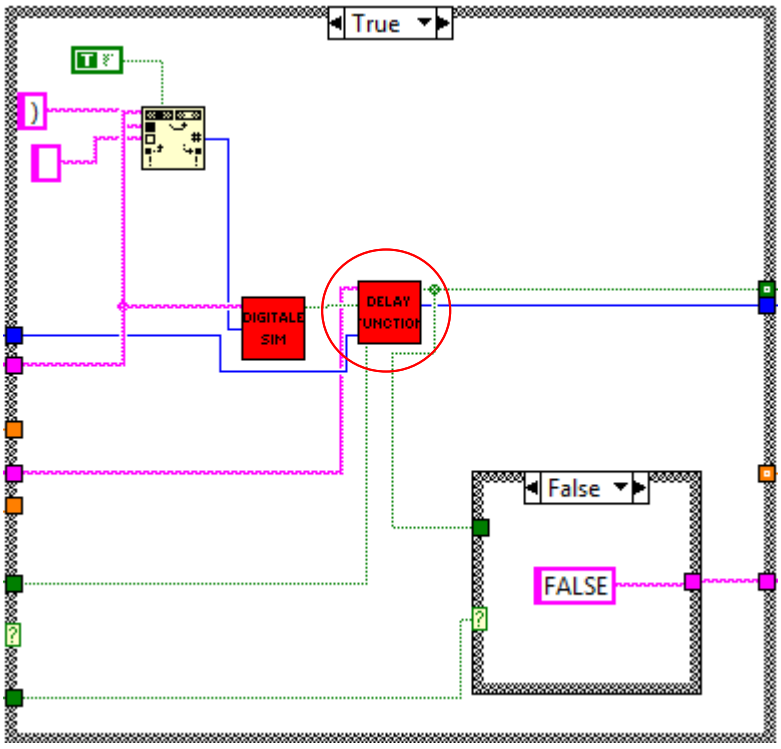


figuur 6.14. Schematische werking situatie “2”.

6.3 Specials

6.3.1 Digitale Special.

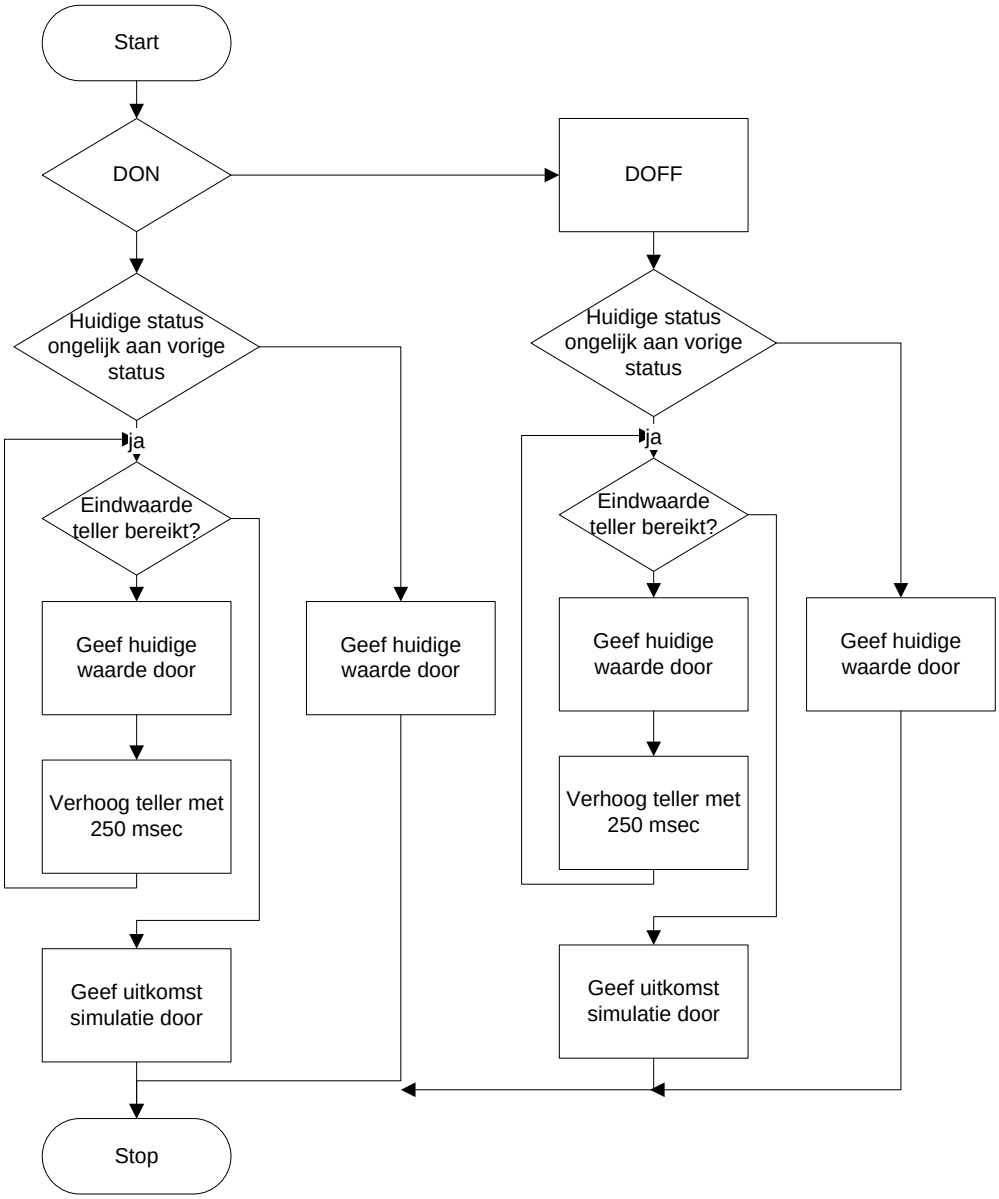
Als er in de digitale simulatie een [en] zit wordt de digitale special functie aan geroepen. Dit is te zien in figuur 6.15.



figuur 6.15. Functie aanroep special.

De werking is als volgt: als er een verschil zit in de huidige status van een uitgang en de status van de uitgang naar simuleren begint er een teller te lopen. Deze telt, afhankelijk van de ingestelde tijd door. Zodra de teller zijn eindwaarde heeft bereikt zal, aan de hand van DON of DOFF de uitgang vertraagd opkomen of afvallen. Een schematische weergave is te zien in figuur 6.16.

De broncode is te zien in bijlage D

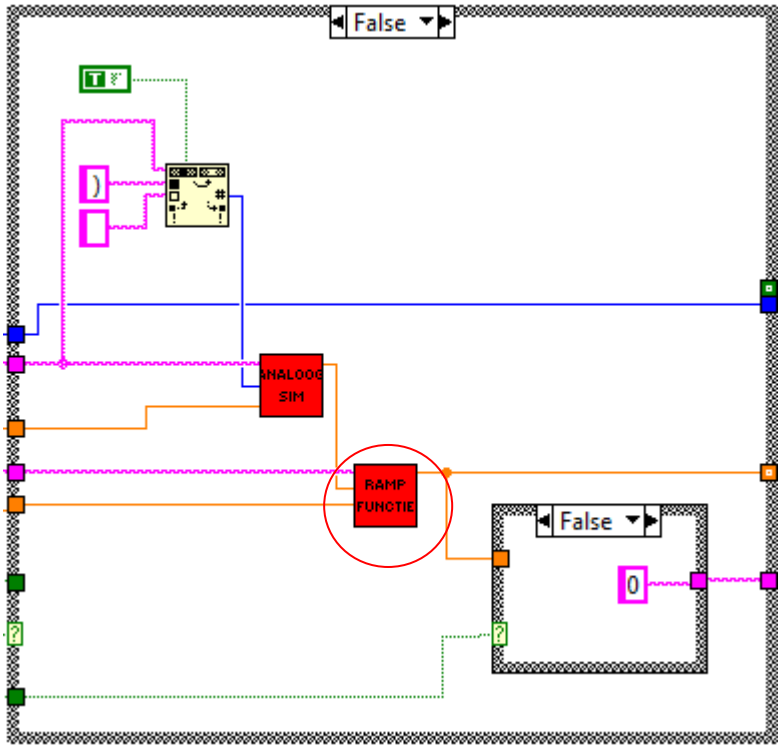


figuur 6.16. Schematische werking digitale special.

6.3.2 Analoge Special.

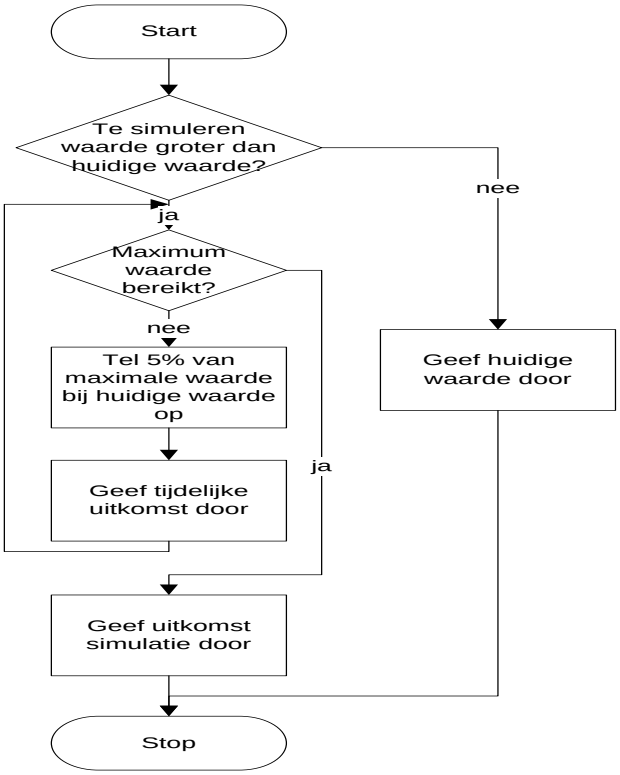
Als er in de analoge simulatie een [en] zit wordt de analoge special functie aan geroepen. Dit is te zien in figuur 6.17.

De broncode is te zien in bijlage D.



figuur 6.17. Functieaanroep analoge special.

De werking is als volgt: als er de te simuleren uitgang groter is dan de huidige ingang wordt de special gestart. In de formule wordt aangegeven hoeveel % van de eindwaarde de simulatie per seconde mag stijgen. Dit is de uitgang van de simulatie. Dit stijgen gaat net zolang door tot de maximale eindwaarde wordt bereikt. Als de maximale waarde wordt bereikt wordt die door gestuurd. In figuur 6.18 is de schematische werking te zien.



figuur 6.18. Schematische werking analoge special.

6.4 Oplevering

Nadat de simulatie mogelijkheden zijn ontworpen en zijn gerealiseerd moeten de simulaties worden getest en worden opgeleverd.
Het testen en opleveren wordt samen uitgevoerd met de bedrijfsbegeleider. Alle simulaties worden bij langs gegaan en uitvoering getest.
Nadat alle simulaties zijn getest en eventuele fouten zijn opgelost kunnen de simulaties mogelijkheden worden ingebouwd in de bestaande software. Het inbouwen wordt gedaan door de bedrijfsbegeleider.

6.5 Conclusie

De uitbreiding van de digitale en analoge simulaties is het meest complexe gedeelte van de opdracht. Met de nieuwe digitale en analoge simulatie mogelijkheden is er een basis gelegd voor de toekomst. Met deze simulaties kunnen namelijk kleppen, pompen, motoren, het verloop van het vullen van een tank met vloeistof worden gesimuleerd.

In de toekomst wil men typicals maken. Dit houdt in dat er standaard simulaties zijn zoals bijvoorbeeld een klep simulaties. Het enige wat moet gebeuren is dat de engineer de in en uitgangen aan deze typical moet toekennen. In de typical staan de voorwaarden voor een klep simulatie. Deze voorwaarden werken op de uitgebreider simulatie mogelijkheden die in dit scriptie zijn ontworpen.

De software engineers zijn zeer tevreden met de uitbreiding van de simulatie mogelijkheden. Met deze simulatie mogelijkheden kunnen ze de besturingssystemen nog beter testen.

Conclusie en aanbevelingen

Conclusie

Het doel van het project is de hardwaresimulator zo uitbreiden dat daarmee de gebruiksvriendelijkheid en de functionaliteit wordt verhoogd. Aan de hand van User-friendly IO configuratie, MODBUS simulatie Master en Slave en met de uitbreiding van de simulatie mogelijkheden worden deze punten gerealiseerd.
De software toepassingen zijn opgeleverd en ingebouwd in de bestaande software.

Niet alle doelstellingen zijn gehaald. De OPC koppeling is niet gerealiseerd. Dit komt doordat de complexiteit van de software toepassingen erg groot was. Om een goede MODBUS simulatie te kunnen ontwerpen is er veel theoretische onderzoek gedaan naar de MODBUS protocollen. Dit heeft veel tijd gekost.
Door tijd gebrek en in overleg met de bedrijfsbegeleider van Imtech Vonk is besloten dat de prioriteiten liggen bij de MODBUS simulatie, User-friendly IO configuratie en de uitbreiding van de simulatie mogelijkheden. Hierdoor is het onderdeel OPC koppeling te komen vervallen.

Alle software is geschreven in LabVIEW. De MODBUS simulaties bestaan uit een MODBUS RTU/ ASCII over RS232 zowel Master als Slave en uit een MODBUS RTU over TCP zowel Master als slave.

Met de User-friendly IO configuratie wordt er een IO dump van Eplan ingeladen. Door middel van de IO configuratie kunnen de IO tags worden verdeeld over de digitale in- en uitgangen en analoge in- en uitgangen. De User-friendly IO configuratie maakt .nic file die wordt ingelezen door de bestaande software.

De simulatie mogelijkheden is uitgebreid met het meer in kunnen vullen dan twee IO tags. Er kunnen nu meerder IO tags in een formule worden ingevuld. De simulatie mogelijkheden zijn verscheidende digitale en analoge reken functies. Hiermee is een basis gelegd voor complexere simulaties.

De gemaakte software toepassingen zijn goed in ontvangst genomen. Begin januari is de hardware simulator met de nieuwe software toepassingen officieel gepresenteerd aan Imtech Vonk door de stagebegeleider.

Aanbeveling.

De simulatie mogelijkheden kunnen worden uitgebreid met typicals. Dit houdt in dat voor simulaties een typical kan worden aangemaakt. Bijvoorbeeld een klep. De gebruiker hoeft alleen nog maar de in- en uitgangen definiëren. In de typical staan de formule berekeningen voor de desbetreffende simulatie. Deze formule berekeningen komen overeen met de uitgebreide simulatie mogelijkheden die nu zijn ontwikkeld.

Persoonlijke evaluatie.

Mijn afstudeer periode bij Imtech Vonk het ik als zeer prettig ervaren. De opdracht sprak mij zeer aan omdat er software moest worden ontworpen en geschreven die door Imtech Vonk echt in bedrijf wordt genomen.

Alle software moest worden geschreven in de “programeertaal” LabVIEW. Aangezien ik geen ervaring had met het programma LabVIEW was het een hele uitdaging om het programma en te taal onder de knie te krijgen.
In het begin ging het programmeren vrij langzaam omdat ik het programma en de kracht van LabVIEW niet kende. Maar naarmate de tijd verstrekte begon ik LabVIEW steeds meer te kennen, daardoor begon ik te denken als LabVIEW, hoe ik bepaalde problemen moet oplossen met de functies die LabVIEW heeft. Hierdoor ging het programmeren steeds makkelijker en sneller.

De functies die voorkomen in de software hebben veel raak vlakken met de programmeertaal C++. In de software komen functies zoals: arrays, for-loops, while-loops, strings en classes voor. Deze functies zijn ook allemaal aanbod gekomen in de lessen C++.
Hierdoor kwam de kennis die ik had op gedaan in de lessen van C++ goed van pas voor dit project. Tevens kwamen de onderwerpen die behandeld zijn in de lessen besturingstechniek, digitale techniek weer terug.

Het projectmatig werken beviel mij zeer goed. Ik vond het een heel leerzaam proces om van een probleem naar een werkende oplossing te komen en alle stappen te volgend die daar tussen horen.

Al met al kan ik terug kijken op een geslaagde afstudeer periode, met goed werkende software programma’s, een mooie scriptie en een tevreden afstudeer bedrijf.

Mark Haanstra.

Coevorden, Januari.

Literatuurlijst

Geraadpleegde boeken:

1. Rouland J.G. & Schrage J. J., Digitale techniek: Analyse en synthese van schakelingen, eerste druk, Houten, Stam techniek, 2007.
2. Laan G., Aan de slag met C++, vierde druk.
3. Panko R., Datanetwerken en telecommunicatie, vijfde druk, Amsterdam, Pearson Education Benelux, 2005.
4. Grit R., Projectmanagement, vierde druk, Groningen/ Houten, Wolters-Noordhoff bv, 2005.

Internet sites:

a. http://en.wikipedia.org/wiki/Modbus (13-09-2010)
b. http://www.simplymodbus.ca/ (13-09-2010)
c. http://www.ni.com/ (02-09-2010)
d. http://nl.wikipedia.org/wiki/OSI-model (14-10-2010)

Bijlagen

De bijlagen zijn opgenomen in een apart bijlagenboek. Dit is gedaan omdat er vanuit deze scriptie veel wordt gerefereerd naar de bijlagen.

Inhoudsopgave bijlagenboek: