Sturingsplatform voor de mechatronica

Van Herzele Gill

Promotor(en) :

dr. ir. Touhafi Abdellah
Prof. dr. ir. Lefeber Dirk

Eindwerk voorgedragen tot het behalen van de graad van

industrieel ingenieur elektronica, optie ontwerptechnieken

Academiejaar 2004-2005

Sturingsplatform voor de mechatronica

Van Herzele Gill

Promotor(en) :

dr. ir. Touhafi Abdellah
Prof. dr. ir. Lefeber Dirk

Eindwerk voorgedragen tot het behalen van de graad van

industrieel ingenieur elektronica, optie ontwerptechnieken

Academiejaar 2004-2005

Synopsis

I. Sturingsplatform voor de mechatronica

Deze thesis behandelt de miniaturisatie van de communicatie-hardware gebruikt bij de tweepotige robot Lucy, in ontwikkeling aan de faculteit toegepaste wetenschappen van de Vrije Universiteit Brussel (VUB). Lucy maakt, in tegenstelling tot de meeste robots, gebruik van pneumatiek ter hoogte van de gewrichten in plaats van elektrische motoren om voor beweging te zorgen.

Voor alle communicatie bezit, tot op heden, elk van deze gewrichten een eigen microcontroller. Deze laatsten zijn op hun beurt via Dual Ported Random Access Memory (DPRAM) verbonden met een externe computer, die de nodige verdere bewerkingen uitvoert op de gegevens.

De modernisatie van de communicatie hardware en het compacter maken van de elektronica is een eerste stap in het onderzoek naar het efficiënt gebruik van compacte stuurlogica door middel van Field Programmable Gate Array's (FPGA) voor Lucy.

Hiervoor werd de interne communicatie opgedeeld in drie onafhankelijke entiteiten, de kwadratuur decoder, de kleppensturing en de Serial Peripheral Interface (SPI), een bekend en veelgebruikt communicatie protocol tussen chips en microcontrollers. De externe communicatie vormt een aparte entiteit, waarvoor het Enhanced Parallel Port (EPP) – protocol werd gebruikt.

Met behulp van de Hardware Description Language (HDL) programmeertaal werd de interne en externe communicatie digitaal ontwikkeld en met succes getest. De code voor communicatie met de kwadratuur encoder, de kleppen en de sensoren, werkt naar behoren en voldoet volledig aan de gestelde verwachtingen.

Deze nieuwe ontwikkeling betekent een eerste stap naar een volledig onafhankelijk besturingsplatform. De opgedane kennis leidt tot efficiëntere beheersing van de beweging, welke op zijn beurt kan bijdragen tot de ontwikkeling van o.a. betere protheses en orthesen met kunstmatige spieren, die het comfort van de patiënt tegoed zullen komen.

II. Targeting platform for mechatronics

This thesis reports on the miniaturization of the communication hardware, used with the bipedal robot Lucy in development at the faculty of Applied Sciences of the Vrije Universiteit Brussel (VUB). Lucy, unlike most robots, uses pneumatics to create movement at each joint instead of electric engines.

Until now, for all communication, each articulation has its own microcontroller. These microcontrollers are connected through Dual Ported Random Access Memory (DPRAM) with an external computer, which applies the necessary operations on the data.

The modernization of the communication hardware and the process of making the electronics more compact, are a first step in the research to an efficient use of compact targeting logic with Field Programmable Gate Arrays (FPGA) for Lucy.

To this end, the internal communication hardware was divided into three independent entities, the quadrature decoder, the valve controller en the Serial Peripheral Interface (SPI), a well-known and much used communication protocol between chips and microcontrollers. The external communication forms a separate entity, for which the Enhanced Parallel Port (EPP) – protocol was used.

With the use of the Hardware Description Language (HDL), which is a programming language, the internal and external communications were digitally developed and successfully tested. De code needed for the communication with the quadrature encoder, the valves and the sensors is working properly and complies with the expectations.

These new developments signify a first step towards complete independence of the targeting platform. The acquired knowledge leads to more efficient control of the movement, which can contribute to the development of better prosthesis and orthesis equipped with artificial muscles, which could improve the comfort of the patients.

III. Plate-forme de pilotage pour la mécatronique

Cette thèse traite de la miniaturisation du hardware de communication utilisé avec le robot bipède Lucy, en développement à la faculté des Sciences Appliquées de la Vrije Universiteit Brussel (VUB). Lucy utilise, contrairement à la plupart des robots, la pneumatique pour créer du mouvement à chaque jointure et non des moteurs électriques.

Jusqu'à aujourd'hui, pour toute communication, chaque jointure est munie d'un propre microcontrôleur. Ces derniers sont à leur tour connectés à l'ordinateur grâce au Dual Ported Random Access Memory (DPRAM), qui applique les opérations nécessaires sur les données.

La modernisation du hardware de communication et le processus pour rendre l'électronique plus compacte, sont les premiers pas dans la recherche vers l'utilisation plus efficace de logique de pilotage compacte à l'aide du Field Programmable Gate Array (FPGA) pour Lucy.

A cette fin la communication interne a été divisée en trois entités, le décodeur de quadrature, le pilotage des soupapes, et le Serial Peripheral Interface (SPI), un protocole de communication sériel entre chips et microcontrôleurs, bien connu et souvent utilisé. La communication externe représente une entité séparée pour laquelle le protocole Enhanced Parallel Port (EPP) a été utilisé.

A l'aide d'un Langage de description d'hardware (HDL) la communication interne et externe a été développée digitalement et testée avec succès. Le code nécessaire pour la communication avec l'encodeur de quadrature, les soupapes et les capteurs, fonctionne dûment et répond à toutes les attentes.

Ce nouveau développement signifie un premier pas vers une plate-forme de pilotage complètement autonome. La connaissance acquise mène à une maîtrise du mouvement plus efficace, laquelle peut contribuer e.a. au développement de meilleurs prothèses et orthèses munies de muscles artificiels, qui serait bénéfique pour le confort des patients.

Voorwoord

Vandaag de dag streeft men er niet alleen naar alles steeds beter, maar ook compacter en gebruiksvriendelijker te maken. Men moet steeds kunnen inspelen op de dynamische noden. Een passend voorbeeld van zo'n compact en flexiebel hardware product is de Field Programmable Gate Arrays (FPGA).

Men kan de FPGA gemakkelijk aanpassen aan de veranderende noden door nieuwe designs in te voeren. Het gebruik van de FPGA laat ook toe efficient gebruik te maken van de beschikbare plaats, door slechts de nodige gebruikte functies in de FPGA te brengen.

Deze thesis werd verwezenlijkt als onderdeel van een ruimer project binnen de Vrije Universiteit Brussel, waarvoor ik mijn dank richt aan Prof. Lefeber van de VUB.

Het onderzoekswerk, ontwerp en implementatie werden deels thuis en deels in het labo Automatisering en Elektronica van de hogeschool verricht. Het meeste van het werk kon gemakkelijk overal worden uitgevoerd, wat onnodige verplaatsingen vermeed. Enkel sommige specifieke testen moesten worden uitgevoerd in de labo's, daar er meer gespecialiseerde aparatuur werd vereist.

Een startend project, waarbij een reeds bestaande architectuur moest worden vervangen, was het boeiende en interessante uitgangspunt van deze thesis. De huidige werking moest worden onderzocht en geanalyseerd om deze op een andere wijze te kunnen implementeren. Daar de bestaande architectuur reeds het werk deed, was het de vraag niet of het zou kunnen werken, maar hoe het moest geïmplementeerd worden opdat het zou werken.

Het spreekt voor zich dat ik Mr. Touhafi, mijn promoter, dankbaar ben dat hij het realiseren van deze thesis heeft mogelijk gemaakt. Verder wil ik ook een woord van dank richten aan Geert Braeckman, Björn Verrelst en Bram Vanderbroght bij wie ik steeds met vragen terecht kon.

Tot slot wil ik alle mensen bedanken die rechtstreeks of onrechtstreeks hebben bijgedragen bij het tot stand komen van deze thesis : familie, vrienden en kennissen.

Inhoudstafel

Lijst der figuren

Nomenclatuur

Acroniemen

AD	Analoog naar Digitaal
CW	ClockWise (wijzerszin)
CCW	Counter Clockwise (tegenwijzerszin)
DPRAM	Dual Ported Random Access Memory
EPP	Enhanced Parallel Port
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GCI	Global Communication Interface
GCK	Global Clock
HDL	Hardware Description Language
LCI	Lucy's Communication Interface
LSB	Least Significant Bit (Minst beduidende bit)
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit (Meest beduidende bit)
PAS	Pneumatisch Artificiële Spier
PC	Personal Computer
PLD	Programmable Logic Device
SCK	Serial Clock (seriële klok)
SPI	Serial Peripheral Interface
SPP	Standard Parallel Port
SS	Slave Select
TPU	Time Processing Unit
USB	Universal Serial Bus controller
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC HDL

1. Inleiding

1.1 Algemeen

De mechatronica is een onderzoeksdomein dat gebruik maakt van technieken uit de werktuigbouwkunde, de elektrotechniek en de computertechniek. Deze discipline zorgde en zorgt nog steeds voor veelvuldige evoluties binnen de industrie, maar ook daarbuiten. Denk maar aan de robotarmen bij autoconstructeurs, recente voertuigtechnologie of nog dichter bij huis : de wasmachine en de alom gekende robot Wars. Deze zijn allemaal toepassingen van de mechatronica. Mechatronica is dus reeds overal aanwezig is in onze huidige maatschappij.

1.2 Robotica

De laatste jaren is een grote vooruitgang geboekt op het vlak van robotica, een subdomein van de mechatronica. Robots worden niet enkel gebruikt in industriële omgevingen, maar ook voor militaire doeleinden zoals ontmijning en voor zuivere wetenschappelijke toepassingen waarvan de bekendste waarschijnlijk vulkanologie en ruimtevaart zijn.

Robots komen voor in allerlei vormen : groot, klein, op wielen, met poten (2 of meer), elektrische, pneumatische,... Sommige robots vinden hun toepassing, ter vervanging van de mens, voor gevaarlijke opdrachten, in al of niet moeilijk bereikbare plaatsen. Voor zulke toepassingen zijn mobiele meerpotige robots nodig, daar wielen hier onvoldoende grote mobiliteit verschaffen. Ieder jaar maken nieuwe robots hun opkomst.

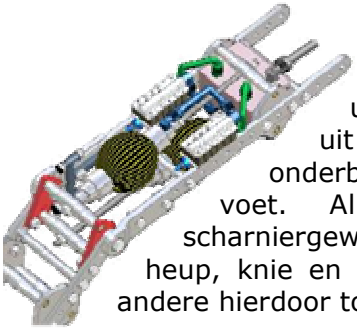
Het onderzoek naar twee potige robots is over de laatste decennia enorm in populariteit toegenomen. Grote hoogtechnologische bedrijven zoals Honda, Toyota, Sony, Fujitsu, e.a. zijn wel op de een of andere manier betrokken bij het onderzoek met tweepotige robotica.

Ook tal van universiteiten doen mee. Denk maar aan de technische universiteit van Delft met de robot Denise, de Technische universiteit van Munchen met Johnnie en nog tal van andere robots aan universiteiten in Frankrijk, Groot-Brittannië, Italië, Polen, de Verenigde Staten,...

Deze robots worden niet altijd gebouwd met commerciële bedoelingen. Zij dienen eerder beschouwd te worden als testcases met als uiteindelijk doel het uitvoeren van opdrachten, even goed of zelfs beter dan de mens. Alzo brengen de huidige robots ons enorm veel kennis bij over de menselijke voortbeweging. Deze kennis zou dan kunnen helpen bij de ontwikkeling van betere protheses (ter vervanging van verloren ledenmaten) en orthesen (voor de revalidatie van de aanwezige ledenmaten) met kunstmatige spieren die het comfort van de patiënt zouden tegoed komen.

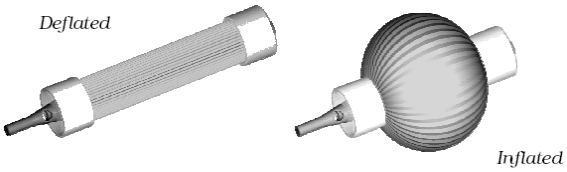
1.3 Lucy

Ook in de faculteit toegepaste wetenschappen van de VUB wordt een robot ontwikkeld. Deze tweepotige robot, Lucy, maakt in tegenstelling tot de meeste robots, gebruik van pneumatiek in plaats van elektrische motoren om voor beweging te zorgen.



Lucy werd gebouwd met de basisgedachte van modulariteit, zodat alle onderdelen dezelfde configuratie vertonen, uitgezonderd de voet. De twee benen zijn identiek opgebouwd uit vier modules, waarvan drie identieke die het bovenbeen, het onderbeen en torso voorstellen en een verschillende module voor de voet. Al deze modules zijn met elkaar verbonden door scharniergewrichten, die elk kunnen vergeleken worden met de menselijke heup, knie en enkelgewrichten. De bewegingsruimte van Lucy wordt onder andere hierdoor tot het sagittale vlak beperkt.

Elk van deze zes identieke modules bestaan uit twee Pneumatische Artificiële Spieren[1] (PAS). Deze spieren stellen op hun beurt de buigspier en trekspier voor, gelijkaardig aan de werking van de menselijke spieren. Elk van deze spieren wordt pneumatisch door middel van een kleppenblok geregeld qua druk.

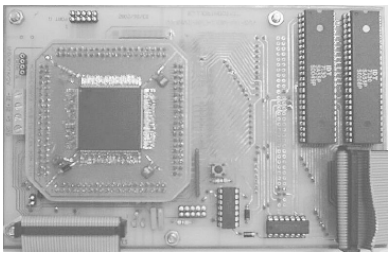


De druk in elkeen van de spieren wordt gemeten door een druksensor die binnenin de spier werd aangebracht. Deze sensoren worden dan verder verbonden met de communicatie-interface die de gegevens verder verwerkt.

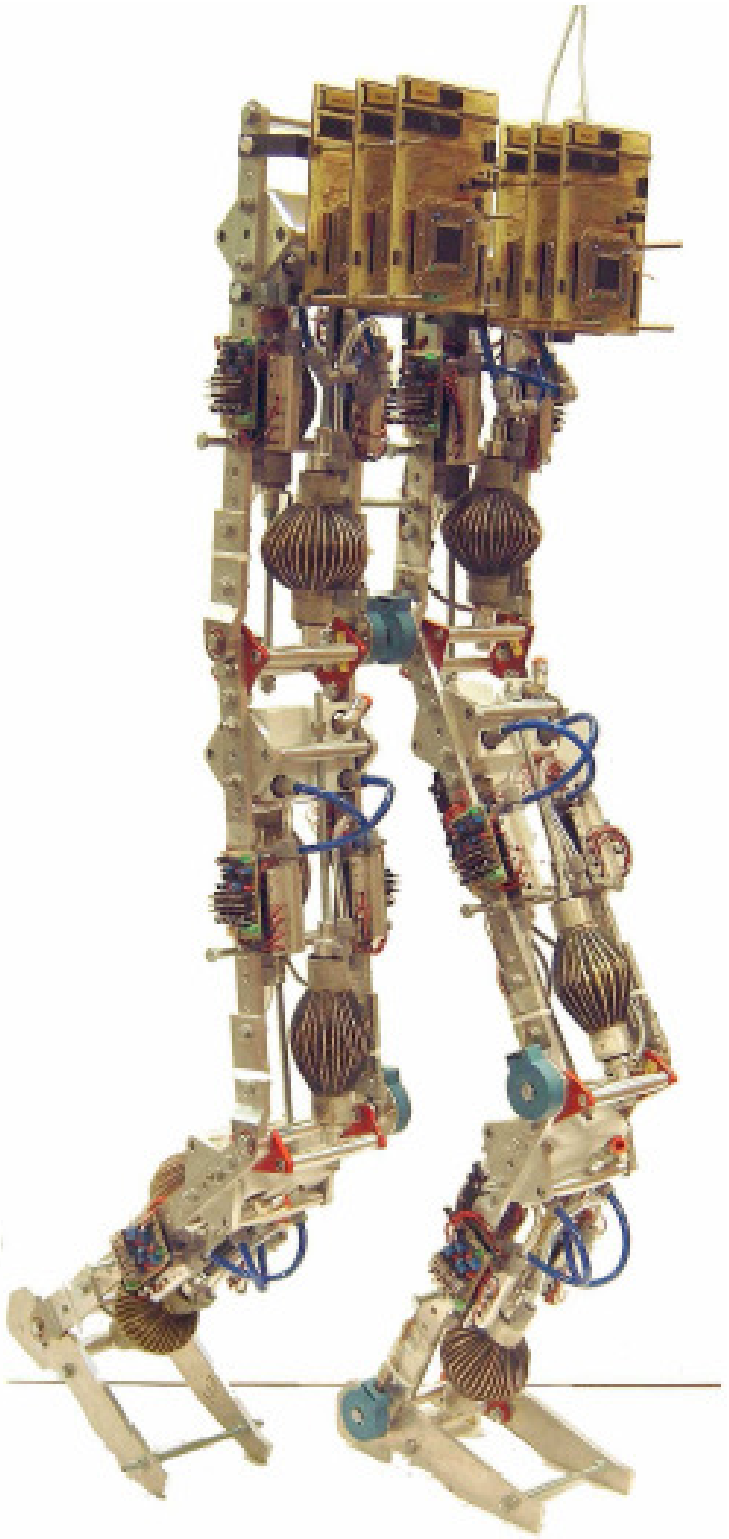
De kracht die de spieren uitoefenen is evenredig met de druk die in elk van de spieren wordt aangebracht. Er wordt dus gebruikgemaakt van een fijne drukregeling om tot de juiste hoek en stijfheid te komen. Zes ventielen zorgen voor de juiste druk in elke spier, waarvan twee dienen als inlaatklep en vier als uitlaatklep. De keuze van het aantal kleppen en de verdeling ervan is een bewuste ontwerpkeuze.

Naast spieren die voor beweging gaan zorgen zijn ook kwadratuur encoders nodig die de positieveranderingen van elk gewricht weergeven waarmede de benodigde druk in de spieren kan bepaald worden. Deze worden evenals de druksensoren verbonden met de communicatie-interface voor verdere verwerking.

Wat de communicatie-interface zelf betreft zijn elk van de gewrichten voorzien van hun eigen microcontroller. Dit vertaalt zich in zes microcontrollers om alle gewrichten te besturen en een zevende, als master voor de onderlinge communicatie. Deze microcontrollers worden op hun beurt, via DPRAM, verbonden met de computer die nodige verdere bewerkingen uitvoert op de gegevens.



[1] Pneumatische Artificiële Spieren



1.4 Dit eindwerk

Wat in dit eindwerk zal worden onderzocht is, of alle onderdelen nodig voor de elementaire communicatie (de functies die nu worden vervuld door de microcontroller) kunnen ingebracht worden in een herconfigureerbare chip (FPGA of Field Programmable Gate Array) die kan aangepast worden aan de wensen en noden van de ontwerper. Meer specifiek moet een kwadratuur decoder, een kleppensturing en een Serieel protocol (SPI) worden ontworpen die men vervolgens in deze herconfigureerbare chip kan onderbrengen. Het serieel protocol heeft betrekking tot de eerder vernoemde druksensoren, die hun drukmetingen doorspelen via een algemeen gekend en veel gebruikt serieel communicatie protocol. Er moet eveneens een communicatie protocol voorzien worden met de PC, daar de robot nog niet volledig autonoom functioneert.

Dit is een eerste stap naar een modernisatie van de communicatie hardware en het compacter maken van de elektronica, die het mogelijk zal maken verder onderzoek te verrichten naar het efficiënt gebruik van compacte stuurlogica door middel van FPGA's in de mechatronica en meer specifiek voor Lucy.

Eerst wordt de gebruikte methodologie toegelicht met uitleg over de gebruikte technieken. Vervolgens wordt elk onderdeel, waarin de communicatie werd ingedeeld, dieper uitgewerkt om een duidelijk beeld te verschaffen over de evolutie van situatie van voor de aanvang van dit eindwerk tot het uiteindelijke bereikte resultaat.

Wij eindigen ten slotte met een besluit waarin de bekomen resultaten van elk onderdeel worden besproken alsook de mogelijke verdere onderzoeksperspectieven.

2. Methodologie

2.1 Analyse

De eerste logische stap bij het oplossen van een probleem, is ongetwijfeld de analyse van het probleem dat zich voordoet. In ons geval vertrekken we echter niet zozeer van een probleem, maar vanuit een wens om de situatie te veranderen. En ook hier is een analyse van de nieuwe gewenste situatie vereist om tot een goed resultaat te komen.

Een manier om aan een probleem tegemoet te komen, is het bedenken en uitwerken van de benodigde gedachtegang die moet gevolgd worden. Vervolgens wordt dan een Finite state machine (FSM), dit is een grafische weergave van deze gedachtegang, opgesteld die deze weerspiegeld in de vorm van een duidelijk schema. Deze Finite state machines zijn aldus een praktische methode die het algemene idee weergeven van de logica die zal moeten gevolgd worden bij het uitwerken van een oplossing.

2.2 Programmering

2.2.1 Hardware Description Language

De moderne aanpak tot het ontwerpen van digitale systemen maakt gebruik van HDL (Hardware Description Language). VHSIC (Very high Speed Integrated Circuit) HDL of kortweg VHDL en Verilog HDL zijn de twee dominante standaarden van HDL die kunnen gebruikt worden om digitale systemen te programmeren.

Ogenschijnlijk is er niet zoveel verschil tussen Verilog en VHDL. Beide zijn IEEE standaarden en worden gebruikt voor het ontwerpen van digitale systemen. VHDL is een hogere programmeertaal dan Verilog, vooral gebruikt in Europa. Als hogere programmeertaal is zij eerder zeer omvangrijk maar flexibel. Verilog is een taal die op VHDL lijkt, maar eerder geschikt is voor laag niveau beschrijvingen, en wordt vooral in de Verenigde Staten gebruikt.

Hoewel VHDL een hogere programmeertaal is, vergt deze meer code om tot hetzelfde resultaat te komen. Vaak worden deze twee talen samen naast elkaar gebruikt, maar voor verschillende toepassingen.

Voor dit eindwerk werd gebruik gemaakt van VHDL. Deze beslissing vloeide niet voort uit een grondige analyse van de twee talen, maar is gebaseerd op de reeds beschikbare kennis met VHDL in de onderzoeksgroep. Een voorbeeld van deze code vindt u onderaan deze pagina.

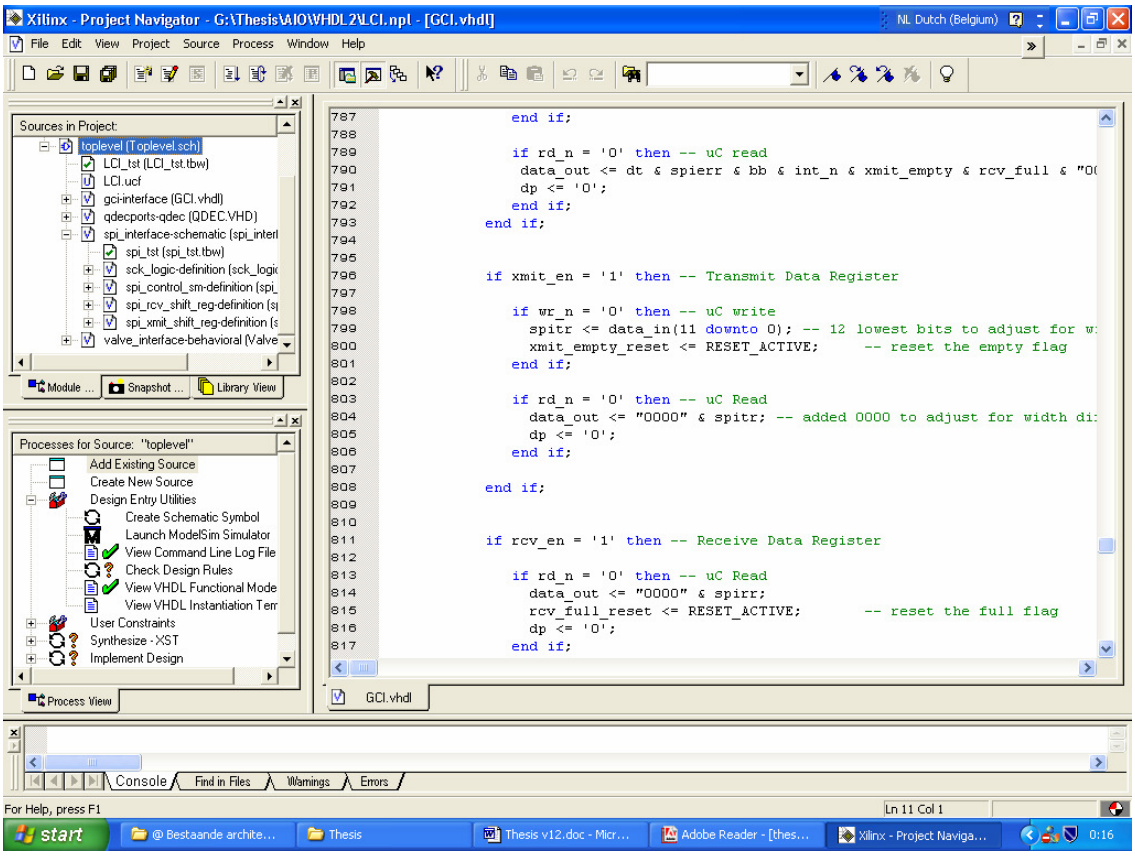
HDL-talen voor het programmeren van de digitale systemen volgen niet dezelfde denkwijze als andere programmeertalen die normaal gebruikt worden voor softwareontwikkeling.

Wanneer men spreekt over programmeren, zij het in C, C++, Delphi, Java of nog andere bekende talen, is de denkwijze waarmee men programmeert normaal gebaseerd op een sequentiële gedachtegang. Dit betekent dat de code die men schrijft, wordt uitgevoerd naarmate we deze tegenkomen. Van boven naar onder. Vergelijk het maar met het lezen van een boek.

Bij HDL-programmatie echter is de achterliggende denkwijze voor het schrijven en lezen van de code fundamenteel verschillend. Hier worden de instructies niet uitgevoerd van boven naar onder, maar allemaal gelijktijdig, zij het in lagen. Men zou het kunnen omschrijven als een parallelle programmatie, want het is ook op die manier dat de digitale logica functioneert. De code wordt omgezet naar logische poorten. Deze logische poorten veranderen op ieder ogenblik hun uitgang, afhankelijk van het signaal dat zich aan hun ingang bevindt. Men zou dus kunnen stellen dat de code niet sequentieel wordt uitgevoerd, maar in de volgorde waarin een aangelegd signaal deze code (logische poorten) gaat bereiken.

```
entity upcnt5 is
        port(   cnt_en  : in    STD_LOGIC;          -- Count enable
                clr     : in    STD_LOGIC;          -- Active low clear
                clk     : in    STD_LOGIC;          -- Clock
                qout    : out   STD_LOGIC_VECTOR (4 downto 0) );
end upcnt5;

architecture DEFINITION of upcnt5 is

        constant RESET_ACTIVE : std_logic := '0';
        signal   q_int : UNSIGNED (4 downto 0);

begin

    process(clk, clr)

    begin
        if (clr = RESET_ACTIVE) then        -- Clear output register
                q_int <= (others => '0');
        elsif (clk'event) and clk = '1' then        -- On rising edge of clock count

            if cnt_en = '1' then
                q_int <= q_int + 1;
            end if;

        end if;

    end process;

    qout <= STD_LOGIC_VECTOR(q_int);

end DEFINITION;
```

2.2.2 Xilinx ISE



Dit is een programma dat alle tools bevat nodig voor het programmeren in een HDL van een FPGA (herprogrameerbare chip of Field Programmable Gate Array).

Deze zet de geprogrammeerde code om naar een bitfile, een binaire voorstelling van de verschillende poorten die nodig zijn. Deze kan dan vervolgens worden ingeladen in een FPGA, die dan klaar is voor gebruik.

2.3 Simulatie

2.3.1 Modelsim

Modelsim laat toe simulaties te doen van de logica van het ontworpen design op de pc. Men kan dan signalen aanleggen aan de ingangen van ons schema en dit programma zal dan weergeven wat de uitgangssignalen zouden worden over een bepaalde periode.

Een simulatie blijft echter steeds een simulatie en geeft enkel weer of de logica achter de programmatie al of niet goed gebeurde. Er is dus bij een correct werkende simulatie geen garantie dat het design ook bij het werkelijk implementeren ervan correct zou werken. Het blijft hoe dan ook een handig en snel hulpmiddel om fouten in de logica te achterhalen en signaalveranderingen te zien, wat bij de implementatie al wat moeilijker wordt.

2.4 Implementatie

2.4.1 Introductie tot de FPGA

De FPGA of Field Programmable Gate Array is een vorm van logische chip die geherprogrammeerd kan worden. Een FPGA is gelijkaardig aan de PLD (Programmable Logic Device), maar waar PLD's over het algemeen beperkt zijn tot een honderdtal gates (AND & OR), kan een FPGA duizenden gates ondersteunen. Ook bestaan er FPGA's die uitgerust zijn met een powerpc (IBM®), die ze ook krachtig maakt voor het uitvoeren van berekeningen.

FPGA's zijn vooral populair bij het prototyperen van geïntegreerde circuit designs. Eens de design vast ligt wordt dan overgegaan naar het gebruik van hardwired chips

Het gebruik van FPGA's maakt het ontwerpen van schema's in de digitale elektronica een stuk gemakkelijker, sneller, maar vooral goedkoper.

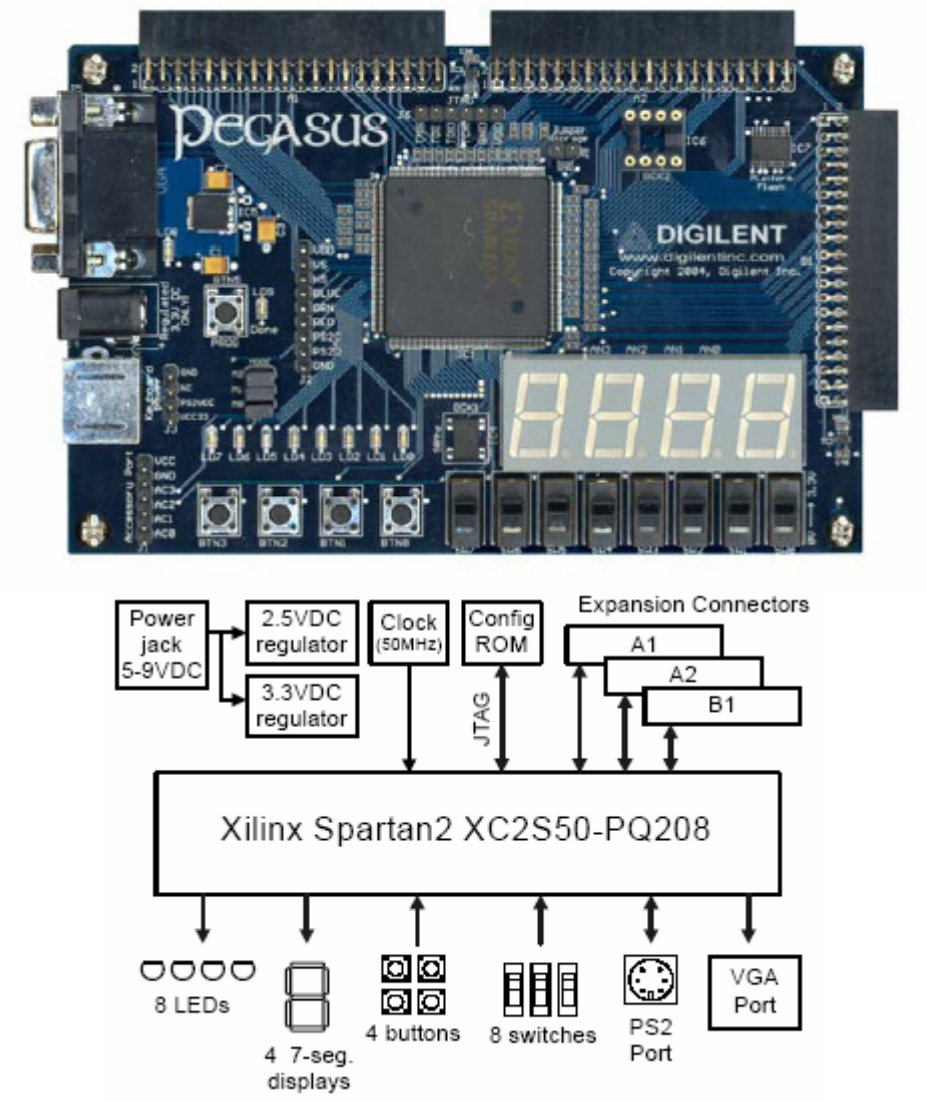
2.4.2 Pegasus Testboard

Voor de implementatie van de code tijdens dit eindwerk werd gebruik gemaakt van een testbord Pegasus van Digilent. Dit testbord bevat een FPGA, een aantal leds, schakelaars, knoppen, hexadecimale displays en uitbreidingsconnectoren.

Hieraan werden dan de nodige randtoestellen verbonden zodat de werking van het geprogrammeerde circuit kon nagegaan worden.

2.5 Lucy Testprint

Voor het uittesten van bepaalde aspecten van de code werd het gebruik van een testschakeling vereist. Ook voor de tijdelijke communicatie met de computer moest een verbinding voorzien worden. In de appendix vindt u een meer uitgebreide beschrijving van dit testbordje.



figuur 1: Pegasus testbordje

3. Onderdelen

3.1 Kleppen

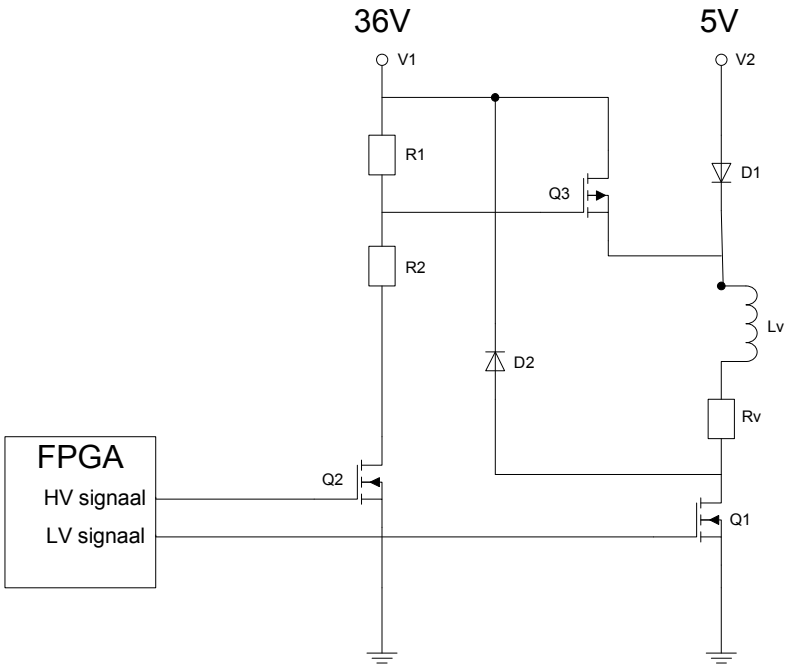
3.1.1 Werking

De kleppen regelen de druk in de spieren. Zoals al eerder werd vernoemd heeft elke spier 6 kleppen (2 inlaat en 4 uitlaat). Voor de principiële werking echter heeft deze informatie weinig belang.

Bij het sturen van de kleppen moet er eerst een hoge spanning worden aangelegd over de ventiel om deze open te krijgen. Na een korte tijdspanne mag de spanning over de ventiel worden verlaagd, daar een lagere spanning genoeg is om de ventiel open te houden. Meer nog : de hoge spanning, die werd aangelegd over de klep voor het open trekken ervan, mag slechts gedurende een korte periode aanwezig zijn, daar deze anders zou doorbranden. Bij het sluiten van de ventiel volstaat het om de spanning tot nul te brengen.

3.1.2 Huidige situatie

De sturing van de kleppen maakt dus gebruik van het "speed-up in tension" principe, waar eerst een hogere spanning wordt aangelegd over de klep, om deze sneller open te krijgen.



figuur 2 : Speed-up circuit

De tijd dat er over het ventiel een hoge spanning (36V) staat, wordt bepaald door een opamp, geschakeld als differentiator, die de afgeleide berekent doorheen de spoel van de klep. De maximale tijd (1 ms) dat er een hoge spanning over de klep mag staan, wordt hier bepaald door middel van een one-shot circuit, parallel geschakeld met het uitgangsmechanisme. Dit circuit verlaagt de spanning over de klep sowieso na deze tijdspanne.

3.1.3 Analyse

De regeling voor maximale tijdspanne die werd uitgevoerd door de one-shot circuit zal nu digitaal in de FPGA gebeuren. De rest van de analoge schakeling blijft behouden.

De communicatie-eenheid van elke module moet in staat zijn om 2 spieren aan te sturen. Mits men zes kleppen telt per spier, moeten er dus in het totaal 12 kleppen kunnen worden aangesproken. Er dient dus een register voorzien te worden van 12 bit, zodat elke klep apart kan worden aangesproken.

De speed-up circuit maakt gebruik van twee signalen per klep om de juiste spanningen over de kleppen de plaatsen. Bij het openen van een klep moeten deze twee signalen initieel actief zijn, waardoor een hogere spanning over de klep komt te staan. Na een korte tijdspanne moet een van de twee signalen worden gedesactiveerd, zodat de spanning over de klep lager komt te staan. Als men dan uiteindelijk de klep wilt sluiten, moet ook dit laatste actieve signaal, gedesactiveerd worden.

Er is aldus een systeem nodig waar, bij een actief wordend registerbit, eerst beide signalen worden hoog geplaatst. Vervolgens, na een vastgelegde tijdspanne, wordt één van de twee signalen naar nul gebracht. En tenslotte worden beide signalen tot nul herleid, bij het inactief worden van ditzelfde registerbit.

3.1.4 Programmering

Zoals bij de analyse al werd duidelijk gemaakt, moet men 12 kleppen aansturen die elk individueel aanspreekbaar zijn. Hiervoor werd een 12 bit breed register aangemaakt. Voor elke bit uit dit register, moet echter dezelfde bewerking gebeuren. Het volstaat dus de logica te bedenken voor één klep en deze uit te breiden naar alle 12.

Bij een verandering van de stand van de kleppen, wordt het kleppen register aangesproken. Dit register wordt bij elke stijgende klokflank gecontroleerd en rechtstreeks naar buiten gebracht. Dit signaal vormt het laagspanningssignaal.

Als de betrokken registerbit bij de vorige klokflank niet aan stond, wordt ook een tweede signaal actief. Dit laatste signaal, dat zorgt voor een verhoogde spanning over de kleppen, zal men het hoogspanningssignaal noemen.

Het hoogspanningssignaal mag slechts gedurende een korte periode actief zijn, daar men anders het risico loopt dat de klep zou doorbranden. De tijd waarvoor dit signaal actief is, wordt bepaald met een teller waar, op het tijdstip van actief worden van de betrokken registerbit, een vaste waarde aan wordt opgeteld en bijgehouden. Eens deze waarde wordt bereikt door de teller, gaat het hoogspanningssignaal terug naar nul.

Men had ook kunnen opteren (wat aanvankelijk ook het geval was) voor een systeem waarbij een teller telkens vanaf nul naar een bepaalde waarde telt (of omgekeerd). Eens deze waarde (of nul) bereikt, zou dan het hoogspanningssignaal naar nul worden gebracht.

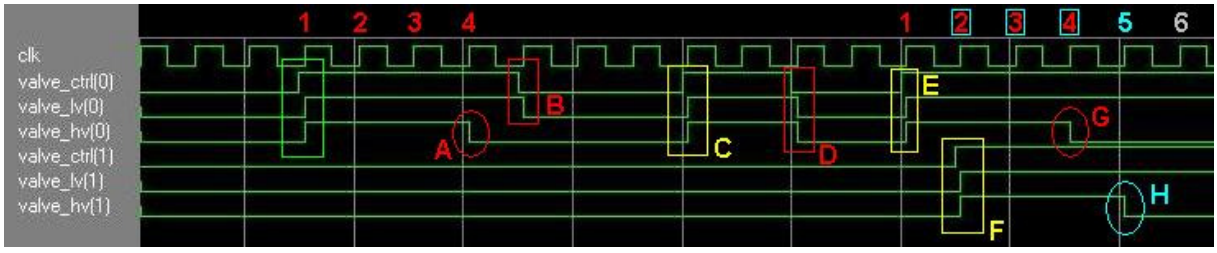
Waarom werd nu voort het eerste systeem gekozen en niet voor dit tweede? Beide systemen zouden naar behoren werken. Het tweede systeem zou voor een gegeven bit ook minder logica vereisen en dus ook minder plaats innemen. Maar zoals al eerder werd vermeld, hebben we niet één bit die gebruik maakt van deze logica, maar 12.

Als men dus de nodige logica gaat bekijken voor alle 12 bit van het register, komt men tot een totaal andere conclusie. Inderdaad, nu neemt de tweede methode niet minder, maar meer plaats in dan de eerste. De uitleg achter deze ommekeer is nochtans eenvoudig: daar waar bij de tweede methode alle 12 register bits een aparte teller gebruiken en één tijdelijk register, heeft de eerste methode voor al deze registerbits slechts één teller nodig en 12 tijdelijke registers. Het is al snel duidelijk dat 11 tellers omvangrijker zijn dan 11 tijdelijke registers. Hier is dus de beslissing tot het gebruik van één 'doorlopende' teller uit voortgevloeid.

3.1.5 Simulatie

Vanuit een praktisch oogpunt werd voor de simulatie de vaste tijdspanne van één milliseconde naar drie klokpulsen gereduceerd.

Zoals op figuur 3 te zien is, wordt het valve_HV signaal hoog als het overeenkomstige registerbit valve_ctrl op één komt te staan. Als er drie klokpulsen voorbij zijn, wordt het hoogspanningssignaal terug nul [A]. Het Valve_LV signaal daarentegen blijft op één staan tot het overeenkomstige registerbit op nul komt te staan [B]. Het valve_LV signaal is dus slechts een weerspiegeling van het registerbit.



figuur 3 : Simulatie van de kleppen aansturing

In C wordt het registerbit terug op één gezet waardoor zowel Valve_LV & Valve_HV op één komen te staan. Als het registerbit voor het einde van de vaste tijdspanne terug naar nul gaat [D], worden zowel het hoogspanning als het laagspanningssignaal terug naar nul getrokken.

Tenslotte, ziet men dat elk signaal met deze logica, onafhankelijk van de anderen, correct naast elkaar kan worden aangestuurd (ook al is het op verschillende tijdstippen) zoals in de eerste situatie simulatie. Daar wordt klep één, één klokpuls later [F] aangestuurd dan klep nul[E]. Als gevolg hiervan zouden de hoogspanningssignalen ook met één klokpuls verschil terug naar nul moeten gaan. Wat ook het geval is [G & H].

3.1.6 Resultaat

3.1.6.1 Aansturing

De implementatie van de aansturing werkt correct, zoals de voorafgaande simulaties sugereerden, mits de nodige kloksnelheidsbeperking. Daar de tijd dat er over de kleppen een hoge spanning wordt aangelegd zeker niet te hoog mag zijn, is deze ingebed in de programmatie. Die tijdswaarde kan dus niet veranderd worden via de externe communicatie. Dit om te vermijden dat verkeerde waarden de kleppen zou doen doorbranden.

3.1.6.2 Kloksnelheidsbeperking

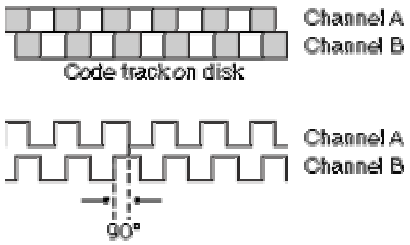
Opdat de schakeling correct zou werken moet er een bovengrens worden opgelegd op de kloksnelheid.

Bij kloksnelheden van 25 MHz werkt, in de uitgevoerde testen, de kleppenregeling correct. Maar als men deze kloksnelheid verhoogd naar 50 MHz, werkt de regeling, willekeurig, voor sommige kleppen niet. Men moet dus rekening houden met deze beperking bij het ontwerpen van een definitieve schakeling.

3.2 Kwadratuur decoders

3.2.1 Werking

De kwadratuur encoder, die de hoekverandering effectief meet, bepaalt de verdraaiing door middel van 2 codereeksen op de draaischijf. Deze codereeksen zijn identiek, maar 90° uit fase met elkaar opgesteld. Over elk van deze 2 codereeksen bevindt zich een fotodiode die de toestand van de onderliggende codereeksen (zwart of wit) weergeeft. Deze toestand wordt dan naar buiten gebracht om de positieverandering te kunnen bepalen.



figuur 4 : Kwadratuur encoder codereeksen

De kwadratuur decoder heeft dan de taak, met behulp van deze twee signalen, zijn positieregister op de gepaste wijze te veranderen. Deze moet dus de signalen kunnen interpreteren om zowel de draaizin als de grootte van de verdraaiing te bepalen.

Er moet ook een bepaalde waarde kunnen toegekend worden aan het positieregister. Deze waarde die tijdelijk in een apart register wordt opgeslagen, mag slechts in het positieregister terecht komen als een derde signaal afkomstig van de kwadratuur encoder op 1 komt te staan. Dit signaal doet zich 1 maal per rotatie voor. Enkel bij het optreden van dit signaal kan dus het positieregister geïnitialiseerd worden.

3.2.2 Huidige situatie

De opmeting van de hoek van elk gewricht gebeurt door middel van een incrementele encoder. Met al deze opmetingen kan men de positie van de verschillende modulen ten opzichte van elkaar bepalen. Wil men nu de volledige positie kennen van de robot, dan zijn nog een aantal extra metingen nodig, zoals de absolute hoek, de horizontale en de verticale positie van het bovenlichaam. Deze metingen zouden kunnen gebeuren door de master-microcontroller, maar de nodige sensoren zijn hiervoor nog niet aanwezig.

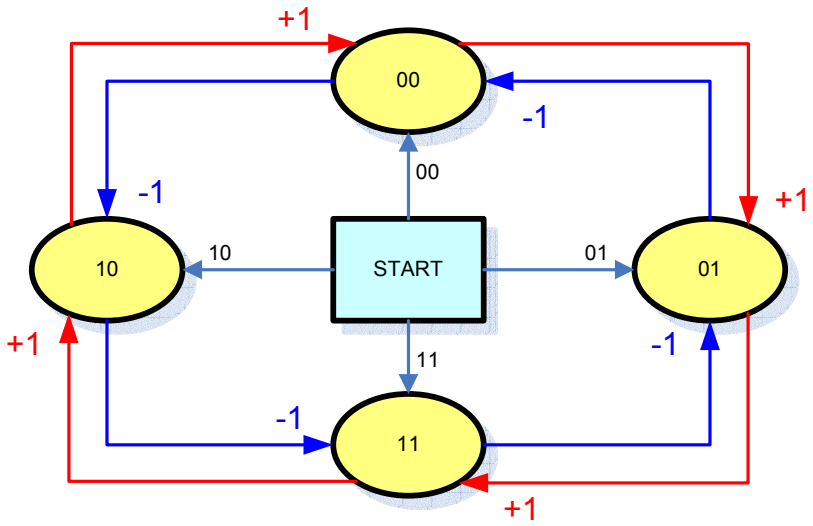
De microcontrollers van elke module verwerken de informatie die ze krijgen van de locale incrementele encoder met behulp van de TPU (Time Processing Unit). Deze bezit een functie, de Quadrature Decode TPU function, die deze informatie verwerkt en de hoekpositie steeds ter beschikking stelt in een register.

3.2.3 Analyse

Het doel, dat hier bereikt moet worden, is dat het door het TPU verrichtte werk, met de informatie afkomstig van de kwadratuur encoders, kan overgenomen worden door de FPGA.

3.2.3.1 Positie

Men gaat hier een 4x decoder implementeren. Dit wil zeggen dat men gaat tellen bij elke optredende flank en niet enkel na volledige pulsen. Dit algemeen gekend systeem zorgt voor een grotere resolutie, waardoor de positie vier maal nauwkeuriger kan worden bepaald.



figuur 5 : Principe schema

Op bovenstaande figuur 5 vindt u een eenvoudige weergave van het gebruikte principe. Bij de start wordt er nagegaan in welke toestand de kwadratuur encoder zich bevindt. Naar gelang de stand, zet de kwadratuur decoder zich in de juiste toestand om van daar uit correct te kunnen werken.

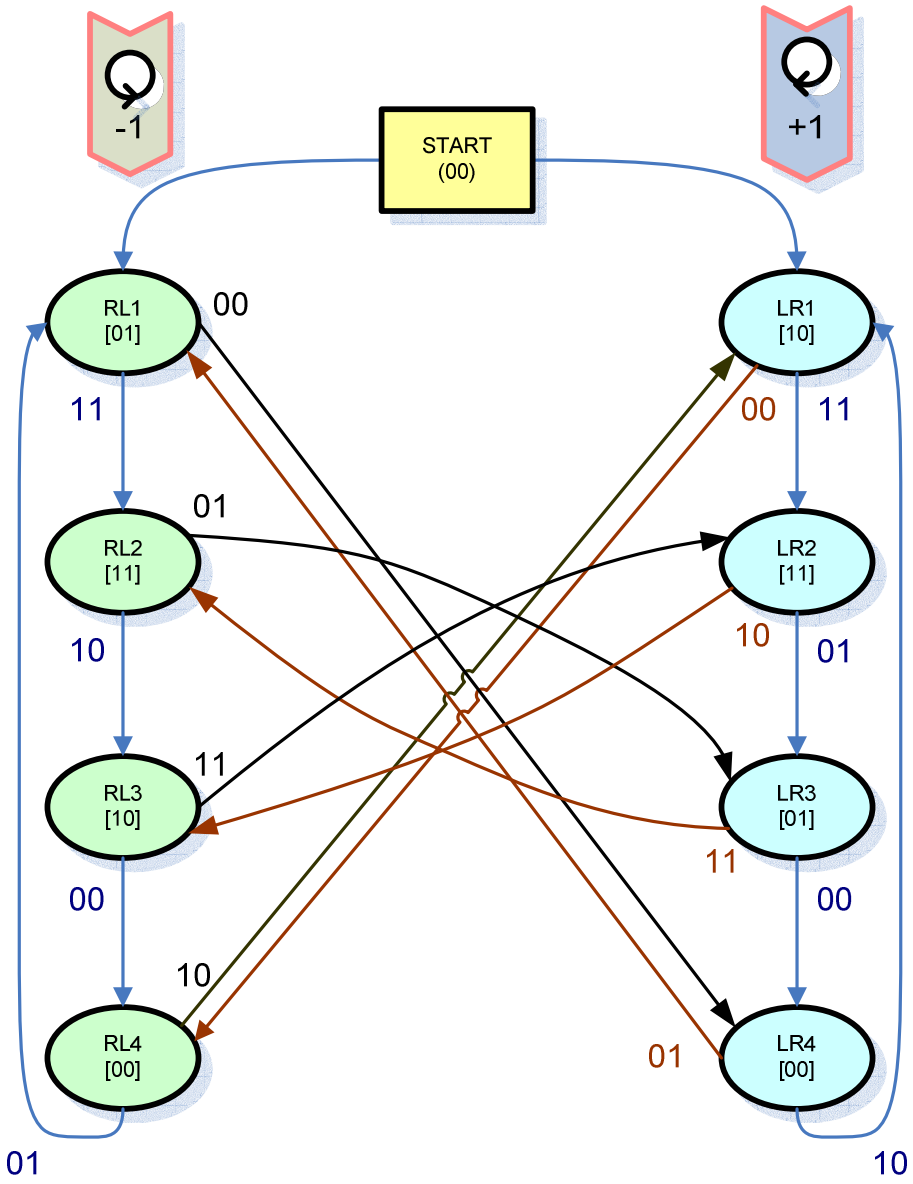
Bij elke toestandsverandering wordt de positieteller met één geïncrementeerd of gedecrementeerd. Laten we onderstellen dat een verdraaiing in wijzerszin zorgt voor een toename van de positieteller. Een verdraaiing in tegen wijzerszin zorgt dan weer voor een afname van de positieteller.

De toestandsverandering voortgebracht door een verdraaiing hangt natuurlijk af van de aansluiting van de kwadratuur encoder signalen. Om een tegengesteld effect te verkrijgen bij verdraaiing, hoeft men enkel de twee kwadratuur aansluitingen te verwisselen. Dit is natuurlijk slechts een principe schema en kan geenszins gebruikt worden als analyse voor de programmatie.

Een iets meer uitgewerkte vorm van het principe schema, die men al zou kunnen beschouwen als een vereenvoudigde FSM (finite state machine), is terug te vinden figuur 6.

Men ziet onmiddelijk dat er al heel wat meer toestanden terug te vinden zijn. Om tot een correcte werking te komen, werden hier de twee draaizinnen van elkaar gescheiden. De blauwe pijlen geven het normale verloop van de kwadratuur decoding. Dit is het verloop dat het meest gaat doorlopen worden. De bruine pijlen geven een draaizin ommekeer weer van wijzerszin naar tegenwijzerszin. De zwate pijlen vertegenwoordigen bijgevolg een ommekeer van tegenwijzerszin naar wijzerszin.

Om echter een complete analyse te bekomen, moet er ook rekening worden gehouden met de programmeertechniek en de foutenbehandeling. In de volledige FSM zijn er dus nog bijkomende voorwaarden en toestanden die nodig zijn en waarmee rekening moet worden gehouden. In Appendix A is een ontleding van deze volledige FSM terug te vinden.



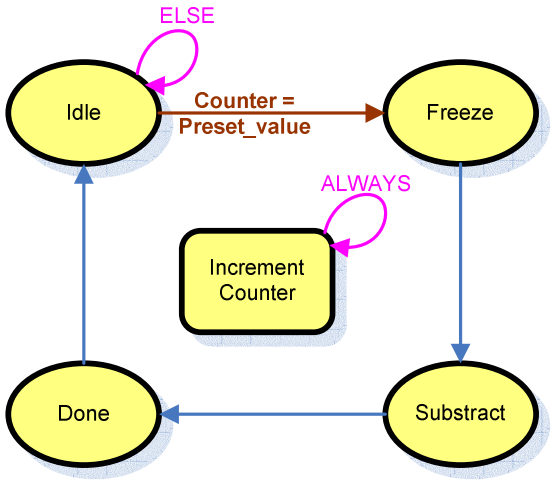
figuur 6 : Vereenvoudigde FSM voor de Kwadratuur decoder

3.2.3.2 Snelheid

Naast de positie moet ook de snelheid bepaald worden aan de hand van de ontvangen kwadratuur signalen.

Hiervoor bestaan verschillende mogelijkheden. Welke mogelijkheid er gebruikt werd en de analyse hiervan werden telkens tijdens het programmeren herbekeken. Want, hoewel de simulaties correct functioneerden, verkregen we incoherente waarden bij de implementatie en het testen van de verschillende "programmatiewijzen".

Uiteindelijk werd gekozen voor een proces waarbij het aantal kwadratuurflanken werd geteld binnen een bepaalde tijdsspanne. Hiervoor werd onderstaande FSM uitgewerkt.



figuur 7 : Kwadratuur decoder FSM

De werkflow evolueert als volgt: eerst telt een teller tot een welbepaalde waarde (dewelke onze tijdspanne voorstelt); eens deze waarde wordt bereikt, gebeurt een momentopname [Freeze]: de huidige momentwaarde van de teller wordt in een register bewaard. Vervolgens wordt de huidige waarde van dat register vergeleken met de register die de vorige waarde bevat [Substract]. Uiteindelijk wordt dit verschil naar buiten gebracht [Done] en wacht men verder tot de teller de vooropgestelde waarde bereikt om dit proces weer te doorlopen [Idle].

3.2.4 Programmering

3.2.4.1 Positie

Bijzondere aandacht diende worden besteed aan de voorvallen waar de kwadratuur signalen voor meerdere opeenvolgende klokflanken onveranderd bleven.

Bij gewone ideale werking, wordt bij elke klokflank van state veranderd in het FSM (zie hoger). Indien nu gedurende meerdere klokflanken de state onveranderd zou blijven, zou de teller niet steeds mogen incrementeren. Bijkomende wacht-states zijn dus nodig, opdat de teller slechts zou veranderen als dit vereist is.

Dit werd reeds in de FSM uit het vorig paragraaf opgenomen, maar was bij de initiële analyse van het probleem niet volledig voorzien. De initiële analyse had immers betrekking op een 1x decoder, waarbij zich dit probleem niet voordeed.

Ook het eenvoudig ijkingsignaal mag hier niet ontbreken. Doch dan moet bij de inbreng van de logica rond dit gebeuren rekening gehouden worden met mogelijke gevolgen op de snelheidsbepaling.

3.2.4.2 Snelheid

Het tellen van het aantal klokflanken tussen twee flanken van de kwadratuur encoder, om zo de snelheid te bepalen, leidt tot incoherente waarden.

Bij tijdsgebrek kon hier niet meer tijd in worden geïnvesteerd en werd beslist over te gaan naar een andere aanpak.

Geopteerd werd voor het tellen van het aantal kwadratuur flanken binnen een bepaalde tijdsspanne. Hierbij verkreeg men wel correcte waarden. Deze aanpak werd dan ook weerhouden.

Zoals al eerder werd aangehaald, had het invoeren van de ijkinglogica negatieve terugslagen op de snelheidsbepaling. Bij een verandering van het counterwaarde door ijking, verkreeg men één onjuiste snelheidsbepaling. Namelijk diegene waarvan het tijdsinterval de ijking omvat, daar deze waarde ook werd toegevoegd aan de snelheid.

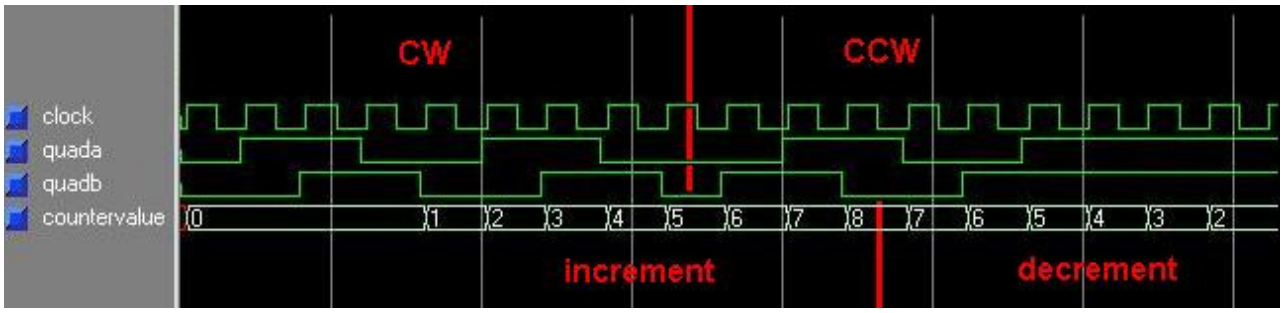
Men moest bijgevolg ook hiervoor aanpassingen aanbrengen zodat bij een ijking ook de signalen voor de snelheidsbepaling naar behoren zouden veranderd worden.

3.2.5 Simulatie

3.2.5.1 Positie

Voor dit onderdeel werden twee simulaties uitgevoerd.

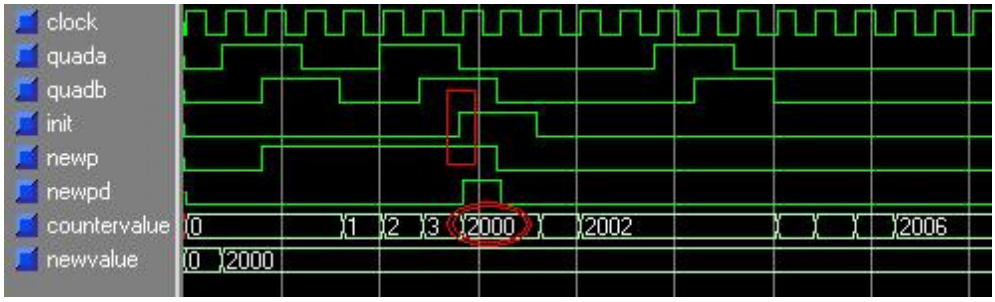
De eerste simuleert de normale werking van de kwadratuur decoder. Onder normale werking verstaan we het analyseren van de inkomende kwadratuur signalen [quada & quadb] en het telkens verhogen of verlagen van het positieregister [countervalue] met één eenheid. Dit is duidelijk afgebeeld in de onderstaande figuur.



figuur 8 : Normale verloop van de kwadratuur decodering

Het positieregister wordt pas na enkele klokflanken veranderd naar de nodige waarde. Dit tengevolge van vertragingen veroorzaakt door de te doorlopen logica. Globaal gezien echter vertaald zich dit in enkele nanoseconden, wat geen impact vertegenwoordigt op de bekomen resultaten.

De tweede simulatie behandelt de verandering van de positieregister-waarde met een nieuwe waarde bij ijking. Hier worden twee signalen beschouwd : Newp (die aanduidt dat er een nieuwe waarde ter beschikking is) & Init (welke afkomstig is van de kwadratuur encoder en ijking toelaat indien nodig).



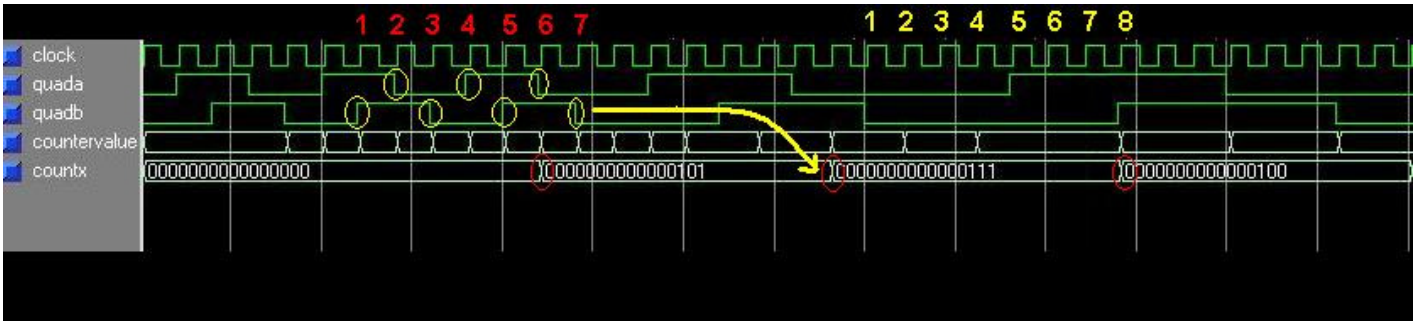
figuur 9 : Ijking van het positieregister

In deze simulatie wordt dus de waarde van het positieregister, tijdens het normale verloop, veranderd naar een andere geijkte waarde. Eens de ijking gebeurd, loopt het proces terug normaal verder maar dan vanaf de ijkingswaarde. Op figuur 9 telt de teller tot drie. Op dat moment wordt de waarde vervangen door de ijkingswaarde (2000) en telt de logica vanaf deze waarde verder tot 2006.

3.2.5.2 Snelheid

Vanuit een praktisch oogpunt werd, voor de simulatie, de tijdsperiode, waarbinnen het aantal kwadratuurflanken zou worden geteld, drastisch gereduceerd tot 7 klokflanken.

De snelheid wordt hier dus bepaald door het aantal kwadratuurflanken te tellen in een bepaald tijdsinterval. Dit is duidelijk te zien op figuur 10. Gedurende 7 klokflanken worden, in dit geval, ook 7 kwadratuurflanken geteld. Dit resultaat vindt men, een aantal klokflanken later, ook terug in de vorm van het 'countx' signaal. Dit signaal gaat het snelheidsregister steeds invullen met de laatste waarde. Men zal verder in deze uitleg de waarde van dit signaal, countx, aldus beschouwen als de waarde van het snelheidsregister.



figuur 10 : Kwadratuur decoder snelheidsregister

Op te merken valt dat het resultaat pas 7 klokflanken na het einde van het interval in het snelheidsregister komt te staan. Dit is het gevolg van de gebruikte methode voor het bepalen van het aantal flanken. Er moet om de zeven klokflanken een bewerking plaatsvinden die de vorige positie van de teller (of dus de huidige waarde van de vorige bewerking) aftrekt van de huidige teller. De aldus bekomen waarde is een indicatie voor de snelheid.

$$snelheidsfactor = (huidige\ waarde - vorige\ waarde)\ \ in \left[\frac{flanken}{tijd} \right]$$

Eens dit werd uitgevoerd, wordt de waarde van de vorige positieteller door die van de huidige vervangen. Deze kan dan bij een volgende bewerking weer worden afgetrokken van de dan optredende positieteller-waarde.

3.2.6 Resultaat

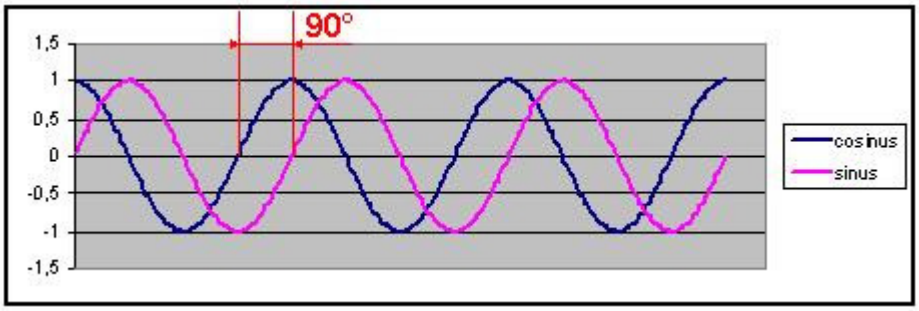
3.2.6.1 Positiebepaling

De positiebepaling verloopt naar wens evenals de ijking ervan. Tests werden uitgevoerd door middel van een slinger. Een sinus-vormige golf dient bekomen als men de posities, bepaald door steekproeven, uitzet in een grafiek. Deze vorm werd ook bekomen, doch met occasionele glitches.

3.2.6.2 Snelheidsbepaling

Ook de snelheidsbepaling verloopt naar wens. Tests werden hiervoor ook uitgevoerd door middel van een slinger. Als men hier steekproeven neemt, moet men eveneens een sinus-vormige golf bekomen. Dit is ook hiet het geval, maar weer met occasionele glitches.

Als we de steekproeven voor zowel de positie als de snelheid gelijktijdig nemen en uitzetten in de tijd verkrijgt men twee sinus-vormige golven. Deze twee golven zijn 90° in fase verschoven ten opzichte van elkaar.



figuur 11 : 90° verschoven sinusoïdale golven

Vanuit fysisch oogpunt weet men dat de snelheid kan opgevat worden als de afgeleide van de plaats naar de tijd.

$$\frac{dx}{dt} = v \quad met \qquad \begin{array}{l} x: positie \\ t: tijd \\ v: snelheid \end{array}$$

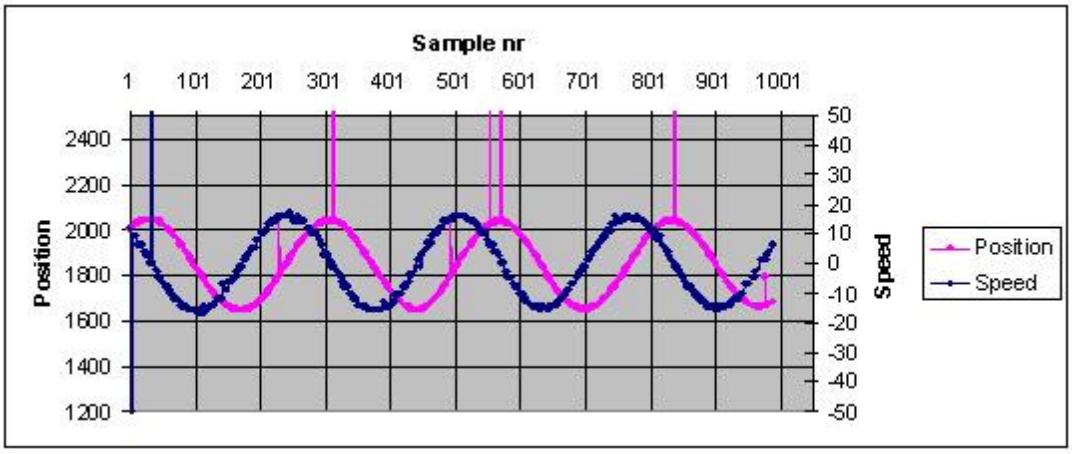
Als men dit nu benadert vanuit een wiskundig oogpunt en men aanneemt dat de positie in de tijd verloopt als een sinus, dan zou de snelheid moeten overeenkomen met een cosinus.

$$\frac{d \sin x}{dt} = \cos x$$

Wetende dat een sinus overeenkomt met een cosinus die 90° verschoven is,

$$\sin x = \cos(x - 90°)$$

bekomt men hier aldus hetzelfde geval als op de grafische weergave.



figuur 12 : Positie - snelheid grafiek

Om de echte waarde van de snelheid te kennen hoeven we slechts volgende bewerking uit te voeren met het uitgelezen getal :

$$v = \frac{2\pi}{resolutie} \times \frac{f}{2^{(tellerbreedte+1)}} \times v_f \quad \left[\frac{rad}{s}\right]$$

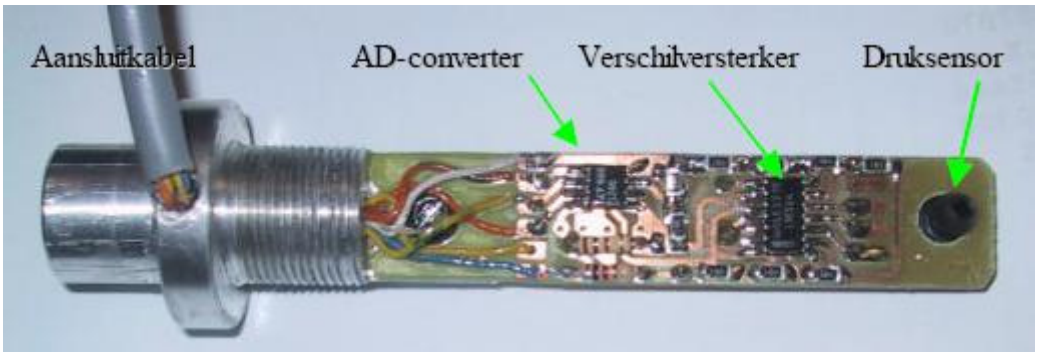
$$snelheid = \frac{2\pi}{resolutie} \times \frac{kloksnelheid}{tellerbereik} \times snelheidsfactor$$

De berekende waarde geeft de snelheid aan in radialen per seconde.

3.3 Drukmetingen

3.3.1 Werking

Druk drukmetingen gebeuren door middel van een absolute druksensor geplaatst in elke spier. Deze sensor genereert een spanning evenredig met de waargenomen druk. Deze spanning wordt vervolgens door een verschilversterker versterkt en onmiddellijk gedigitaliseerd met een AD-converter.



figuur 13 : Druksensor met AD-converter

Al deze componenten bevinden zich dicht bij de sensor om tijdsvertragingen op het gemeten signaal te mijden. Bij grote tijdsvertragingen zou de drukregeling instabiel worden. Naast die tijdsvertragingen leiden ladingsverliezen eveneens tot valse metingen en bijgevolg tot een instabiele drukregeling.

Deze gedigitaliseerde vorm van de aanwezige druk in de spier kan dan op elk moment worden opgevraagd. Het versturen van deze informatie gebeurt op een seriële wijze met het SPI (Motorola ®) protocol. Dit is een bekend en veelgebruikt communicatie protocol tussen chips en microcontrollers.

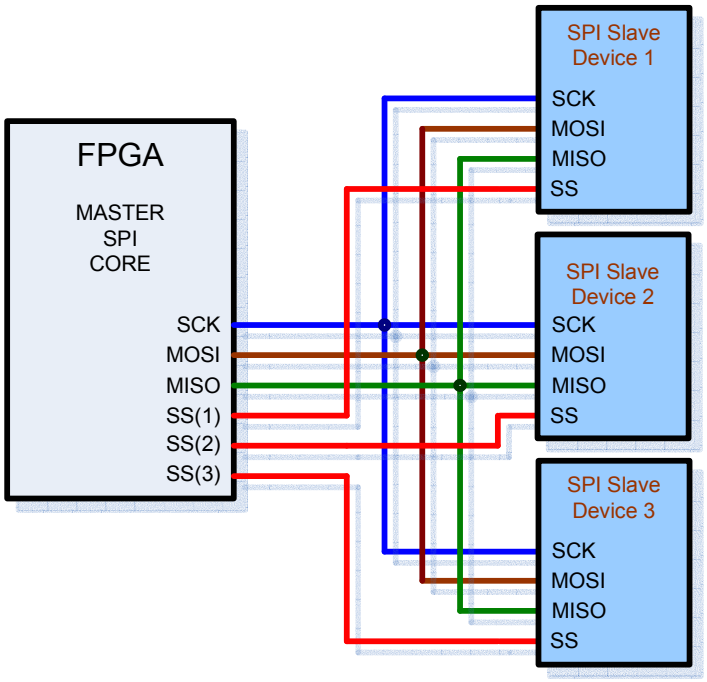
3.3.2 Huidige situatie

De communicatie met de AD-converter wordt momenteel verzorgt door de aanwezige microcontroller. Daar dit protocol vaak gebruikt wordt voor de communicatie tussen microcontroller en andere chips, bezit de microcontroller een reeds ingebouwde SPI controle. Men hoeft dus slechts de juiste parameters in te stellen en de communicatie verloopt naar behoren.

3.3.3 Analyse

Daar FPGA's niet met zo'n reeds ingebouwde SPI controle uitgerust zijn, moet deze geprogrammeerd worden. Dit was dan ook één van de vooropgestelde doelen van dit eindwerk.

Het SPI protocol is een eerder eenvoudig serieel communicatie middel dat gebruik maakt van vier signalen om met elke chip te communiceren, namelijk SCK, MISO, MOSI en SS. Drie van deze signalen zijn bussen waarmee alle chips verbonden zijn.



figuur 14 : SPI situatieschets

Het eerste signaal vertegenwoordigt een globaal signaal dat met alle chips, die het SPI protocol gebruiken, wordt verbonden. Dit globaal signaal, dat men de seriële klok (SCK) noemt, geeft de snelheid aan van de communicatie tussen de chips en de FPGA. Mits de juiste instellingen, zorgt de seriële klok er ook voor dat de verstuurder en ontvanger met elkaar gesynchroniseerd zijn.

Dit wil zeggen dat beide zijden, op het juiste ogenblik, hun data op de lijnen veranderen of behouden.

Het tweede signaal, evenals het eerste signaal een globaal signaal, is de MISO. Deze verbinding laat de Slave toe data naar de Master door te sturen.

Het derde signaal vervult een gelijkaardige functie. Dit signaal geeft de informatie door in de andere richting, dus van de Master naar de Slave. Vandaar ook de benaming MOSI.

Het vierde en laatste signaal, de Slave Select (SS), is een afzonderlijk signaal dat voor elke individuele chip moet worden aangelegd. Men heeft hier dus niet met een bus te maken. Dit signaal speelt de rol van Chip Select en bepaald aldus met welke chip of chips de communicatie moet gebeuren.

Een uitgebreide analyse (evenals een FSM) is terug te vinden in de Application notes van Xilinx over dit SPI protocol.

3.3.4 Programmering

Daar dit protocol een veel gebruikte standaard is, was het mogelijk een reeds bestaand design te vinden op het internet. Mits enkele code aanpassingen, werd het mogelijk dit design in te bouwen in het globale LCI design.

De voornaamste aanpassingen waren de uitbreiding van de byte registers tot 12 bit registers en het groeperen van alle registers en controle logica van zowel SPI als van alle andere onderdelen.

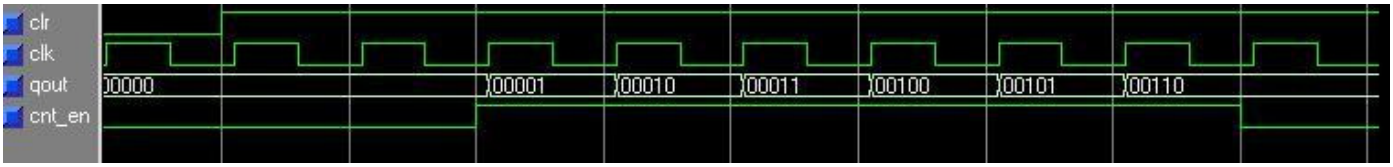
Buiten deze kleine aanpassingen werd niets extra geprogrammeerd om dit protocol te kunnen gebruiken.

Meer uitgebreide informatie kan u terug vinden in de application notes van Xilinx omtrent dit onderwerp[2].

3.3.5 Simulatie

Het SPI protocol werd ontworpen in vier functionele delen: de seriële klokgenerator, de transmitter, de receiver en tenslotte de controller.

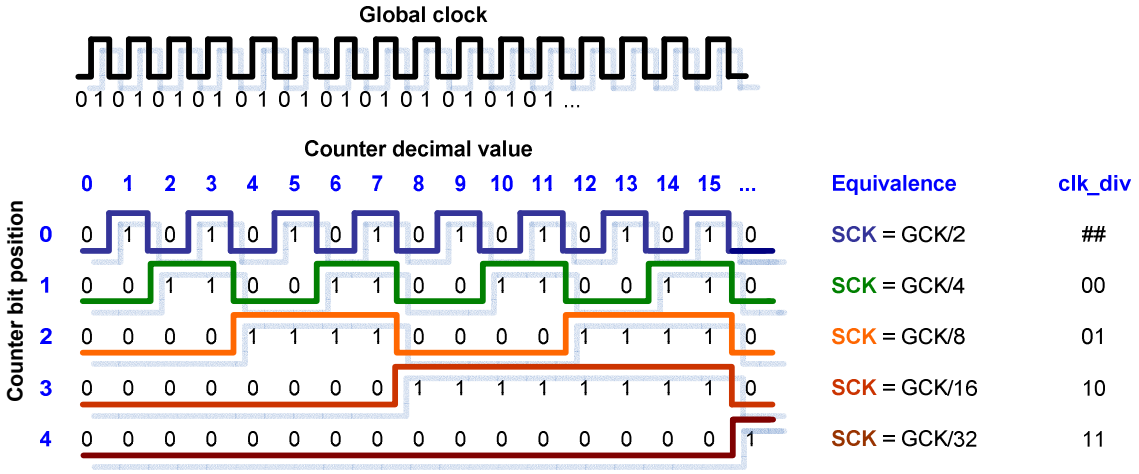
De seriële klokgenerator produceert een kloksignaal met een vier, acht, 16 of 32 maal kleinere frequentie dan de globale klok. Om deze deling te maken, wordt gebruik gemaakt van een teller. Telkens de globale klok en overgang maakt van nul naar één, wordt de teller met één geïncrementeerd. De binaire uitkomst van deze teller wordt dan gebruikt voor die deling.



figuur 15 : Seriële klokgenerator teller

Zo zal de op één na LSB (bit 1) de frequentie aangeven van de seriële klok als de klokdeler in het register op "00" staat. De frequentie van bit 1 is één vierde van deze van de globale klok. Iedere keer dat men een bit verder gaat (wat op figuur 16 overeenkomt met een lijn lager), heeft men twee maal meer globale klok pulsen nodig om deze bit van waarde te veranderen. Dit vertaalt zich in een twee maal tragere seriële kloksnelheid.

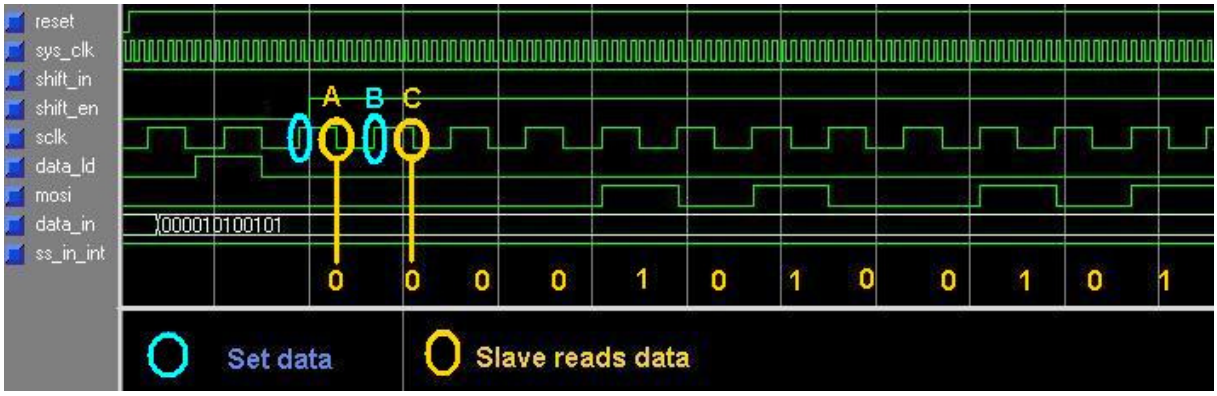
[2] Xilinx application note 386 (zie bijlage)



figuur 16 : Seriële klok generator

Naast het genereren van de seriële klok, brengt dit onderdeel eigenschappen van die seriële klok naar de andere functionele delen, onder de vorm van aparte signalen. Het zijn deze signalen die verder intern zullen gebruikt worden. De gegenereerde seriële klok wordt rechtstreeks naar buiten gebracht om met de chips verbonden te worden.

De transmitter moet de waarde die in zijn register aanwezig is, serieel versturen met behulp van de MOSI aansluiting. Voor de communicatie-vereisten van Lucy is dit onderdeel van het SPI protocol niet nodig, daar enkel informatie moet ontvangen worden van de sensoren.

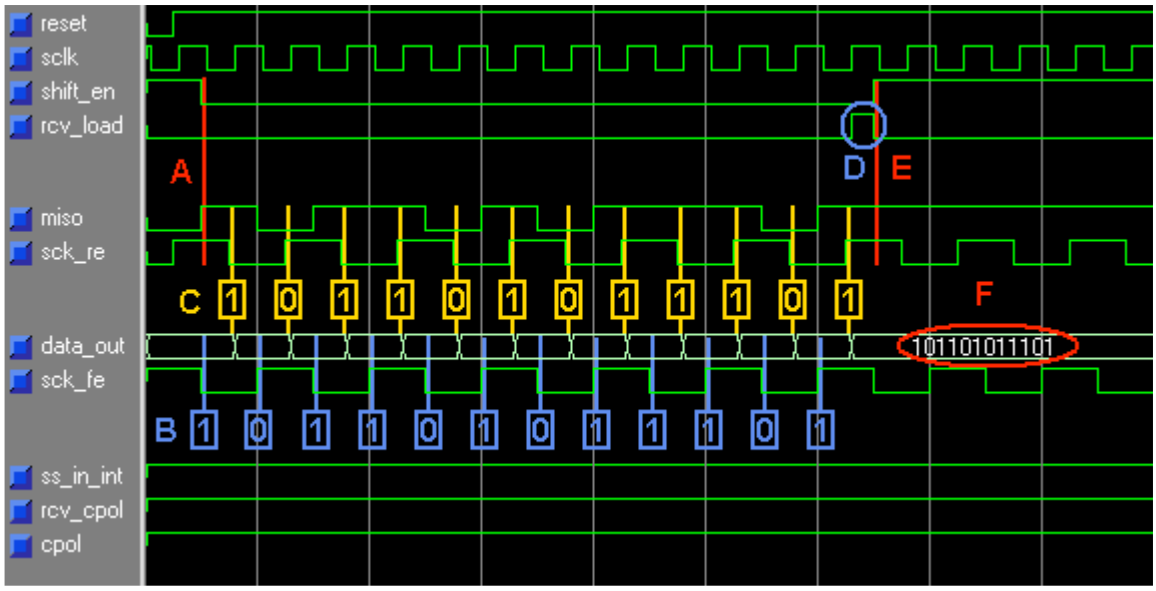


figuur 17 : SPI Transmitter component

De data wordt bij elke stijgende klokflank verandert naar de gewenste waarde. Deze waarde is steeds dezelfde (bit 11), tot en met de eerste klokperiode waarbij de dalende klokflank voorkomt bij een geasserteerde shift_en [A]. Bij de volgende stijgende flank [B] wordt dan de tweede bit (bit 10) uit het register op de MOSI lijn geplaatst. Die bit wordt dan op zijn beurt gelezen door de slave bij de daaropvolgende dalende flank [C]. Dit verloopt verder op dezelfde wijze tot de laatste bit (bit 0) verstuurd werd.

De receiver heeft als taak de waarden die hij detecteert op zijn MISO lijn bij te houden en dit op welbepaalde ogenblikken. Het moment waarop een waarde wordt vastgelegd is van groot belang. Indien de instellingen niet overeenkomen met de eigenschappen van de chip waarmee gecommuniceerd moet worden, zal de communicatie niet correct verlopen.

Bij correcte instelling zal de chip de waarde constant houden terwijl de FPGA de waarde opneemt [C]. Op de momenten ertussen zal de chip dan de waarde indien nodig veranderen [B]. Men merkt aldus dat, bij slechte instelling, de waardeverandering door de chip zou kunnen samenvallen met de opname van de gedetecteerde waarde op de MISO lijn van de FPGA. Wat zou kunnen resulteren in een foutieve waarde.



figuur 18 : SPI Receiver component

Het shift enable signaal moet laag zijn, opdat de opeenvolgende gedetecteerde bitwaarden zouden bijgehouden worden. Indien dit shift enable signaal hoog staat, wordt telkens de zelfde bit van waarde veranderd. Dit heeft als gevolg dat de ontvangen bitwaarden niet worden bijgehouden, maar telkens overschreven.

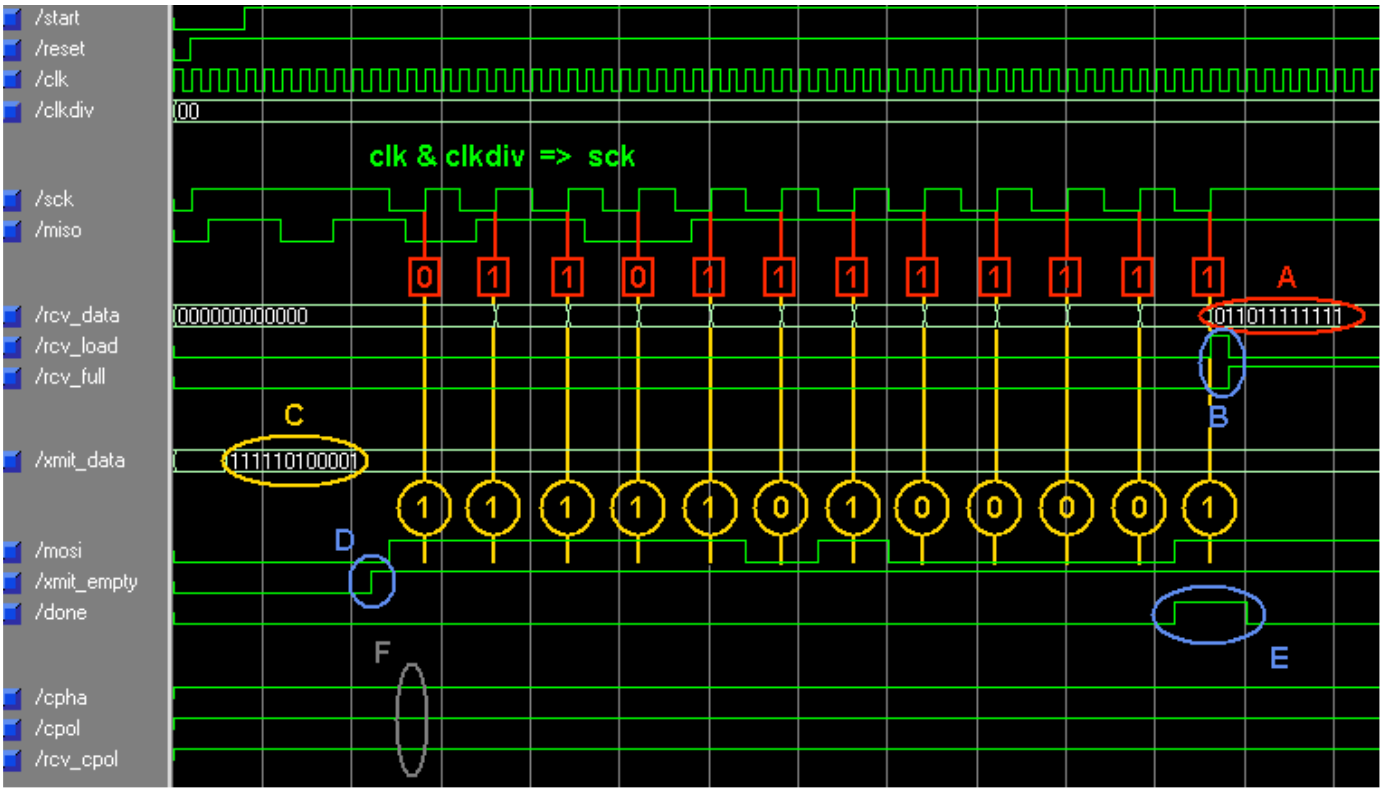
Op figuur 18 wordt het shift enable signaal laag [A] en 12 seriële klokperioden later terug hoog [E]. Tijdens die periode verandert de SPI slave chip de MISO naar de gewenste waarde bij elke dalende seriële klokflank[B]. Bij elke stijgende seriële klokflank wordt de MISO-waarde opgeslagen in een schuifregister [C]. Eens er 12 bits werden ontvangen, wordt een signaal hoog geplaatst[D]. Deze laat de overdracht toe van de ontvangen data [F] naar het receive-register, om later via EPP uitgelezen te kunne worden.

De controller is de component van het SPI schema dat het geheel bestuurd, afhankelijk van de ingegeven parameters in het SPI-Controle register van het GCI[3]. Dit onderdeel onderhoudt ook het SPI-Status register van het GCI, sluist de ontvangen gegevens door naar het SPI-Receive register en de te versturen gegevens van het SPI-Transmit register naar de SPI Transmitter component.

Zo zal de SPI-Controller bij het overbrengen van de te versturen gegevens van het SPI-Transmit register naar het Transmitter shift register, het xmit_empty-statussignaal hoog plaatsen [D,figuur 19]. Dit betekent dat nieuwe te versturen gegevens in dit register kunnen worden geplaatst. Eens naar dit register weg geschreven, komt dit signaal terug laag te staan. Het done-statussignaal geeft weer wanneer een gegevensuitwisseling werd volbracht [E].

[3] Global Communication Interface: zie 3.5 Interne Communicatie

Het geheel kan opgevat worden als een SPI component die, binnen de FPGA een aantal signalen ontvangt afkomstig van de registers van het GCI en met vier pinnen van de FPGA verbonden is voor rechtstreekse externe communicatie.



figuur 19 : Simulatie van het SPI protocol

figuur 19 geeft als het ware een samenvatting van alle vroeger besproken onderdelen van het SPI protocol. Hier ziet men het gedrag van alle componenten bij simultane werking.

3.3.6 Resultaat

3.3.6.1 Test metingen

Voor het uittesten van het protocol werd gebruik gemaakt van het uitbreidingsbordje[4]. Na de nodige instellingen werden testen uitgevoerd door de potentiometer in verschillende standen te brengen. De resultaten die hierbij werden verkregen varieerden tussen 1 en 3839 (0x001 tot 0xEFF in het hexadecimale talstelsel).

Bij de laagste stand wordt 1 verkregen waarbij men dus kan besluiten dat de LSB correct wordt ontvangen. Met de potentiometer op de andere uiterste stand, wordt de waarde 0xeff verkregen. Op te merken valt, dat dit niet de hoogste waarde is die men uit 12 bits kan verkrijgen (4095 of 0xfff).

Zou het kunnen dat de MSB niet goed wordt ontvangen tengevolge van een communicatiefout? Om hieruit conclusies te kunnen trekken zetten we best deze waarde om naar het binaire talstelsel, daar dit de wijze is waarop de informatie wordt doorgegeven. Omzetting van 0xeff in het binair talstelsel geeft aldus volgende waarde : 0x111011111111.

De reden dat de hoogste waarde niet wordt uitgelezen ligt dus niet bij een foutieve communicatie waarbij de MSB niet goed wordt ontvangen. Men kan dit afleiden uit de volgende beschouwing: als de potentiometer in laagste stand staat (en men dus 0x001 verkrijgt), is de MSB nul. Draait men nu aan de potentiometer dan zal deze MSB, vanaf een spanning overstemmende met een kwantisatie-waarde van 2303 (0x8ff), één worden. De MSB verandert over het hele bereik van waarde en wordt bijgevolg dus wel degelijk ontvangen.

Mits de juiste instellingen verloopt de SPI communicatie met de AD-converter bijgevolg naar wens.

3.3.6.2 Nodige instellingen

Om de SPI communicatie goed te laten verlopen, is een correcte instelling van de registers vereist. SPI heeft als het ware verschillende modes. De correcte instellingen zijn afhankelijk van de door de chip gebruikte SPI mode waarmee gecommuniceerd moet worden.

Met de huidige AD-converter chip (LTC1286CS8) moet de SPI enabled staan, daar anders dit protocol niet kan gebruikt worden. De klok fase, polariteit en ontvangpolariteit moeten allen als logische één worden geïnitialiseerd.

Meer algemene informatie over deze modes en de nodige instellingen kan u hier ook terug vinden op de Xilinx Application notes behorende bij de gebruikte code.

[4] Zie Appendix B

3.4 Externe Communicatie

3.4.1 EPP

3.4.1.1 Werking

De enhanced parallel port (EPP) is een parallelle communicatie protocol, die gebruik maakt van de parallelle poort, om data uit te wisselen. De EPP communicatie gebeurt steeds per acht bits, daar de parallelle poort slechts over acht datalijnen beschikt. De parallelle poort bezit ook een aantal controle- en statuslijnen die de handshaking verzorgen.

Door de adres- of datastrobe laag te trekken, weet de andere partij of men op dat moment respectievelijk een adres of data gaat doorsturen. De tegenpartij laat op zijn beurt weten of hij klaar is om informatie te ontvangen door de wachtlijn aan te sturen. Verder is er nog een resetlijn en een interuptlijn voorzien, die het goede verloop van de communicatie helpen verzekeren.

3.4.1.2 Analyse

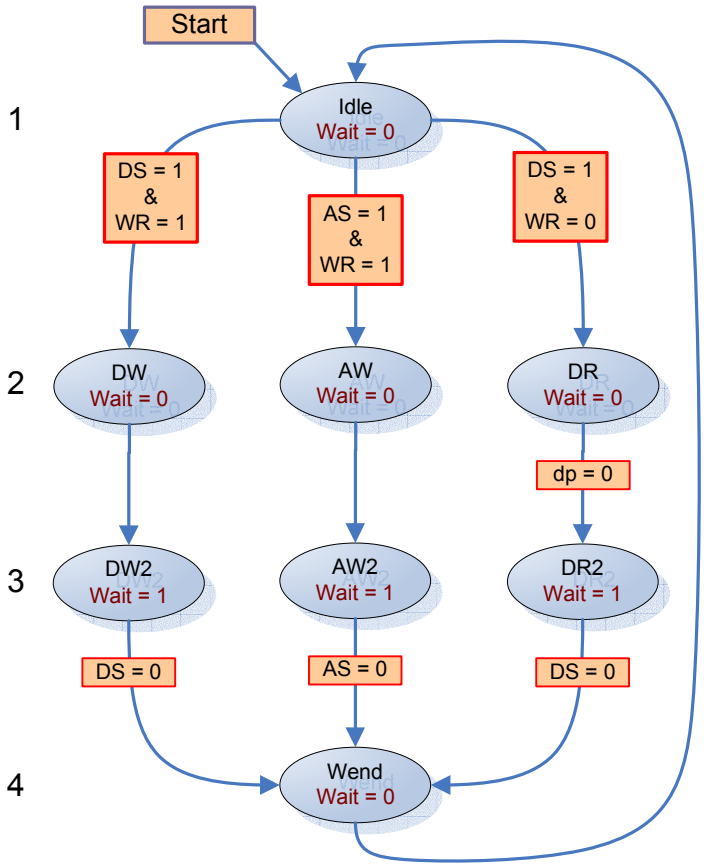
Het onderdeel dat hier geanalyseerd moet worden omvat de passieve zijde van de communicatie. Met de passieve zijde bedoelen we in dit geval de FPGA, daar de PC alle controlesignalen aanstuurt en dus bepaalt welke operatie er plaats vindt. Aan de PC-zijde worden alle signalen hardwarematig gegenereerd door middel van EPP. Men heeft hier dus geen nood aan softwarematige sturing, wat een analyse van deze zijde overbodig maakt.

Wat uiteindelijk in de FPGA moet terechtkomen dient grondig geanalyseerd te worden voor het bereiken van een vlotte communicatie. De manier waarop de signalen worden aangestuurd door de PC en welke signalen deze moet ontvangen, zijn zowat overal terug te vinden op het internet. Dit onderdeel van de FPGA moet dus correct reageren op de ontvangen controlesignalen, de nodige statussignalen terugsturen en de gegevensoverdrachten op de juiste momenten verwerken.

Op figuur 20 vindt u een eenvoudige weergave van de implementatie van dit protocol onder de vorm van een finite state machine. Enkel uitwendige signalen verbonden aan dit protocol zijn weergegeven. In de implementatie ervan moeten, in deze finite state machine, ook interne signalen worden aangebracht om de gegevensverwerking te kunnen verwezelijken.

De aanwezigheid van een voorwaarde (dp^5=0) tussen niveau 2 en 3 bij de lees operatie (DR^6), volgt uit de nood om bij een leesoperatie de nodige data nog te moeten opgehalen op het ontvangen adres. Dit is niet het geval bij een schrijfoperatie (DW^7/AW^8), daar de nodige data door de PC voor niveau 2 reeds op de datalijnen werden geplaatst.

[5] Data Prepared
[6] Data Read
[7] Data Write
[8] Address Write



figuur 20 : Finite state machine van het EPP protocol

Naast het beheer van de externe communicatie, moeten ook de ontvangen gegevens op de juiste wijze en op het gepaste moment verwerkt worden om binnen het globale design bruikbaar te zijn. Dit wordt verder besproken bij de uiteenzetting van de interne communicatie architectuur.

3.4.1.3 Programmering

3.4.1.3.1 VHDL

De programmatie van dit protocol in hardware, is eerder eenvoudig. De EPP handshaking vertegenwoordigt geen ingewikkelde procedure en kan dus snel omgezet worden naar de nodige hardware. De FSM die vroeger besproken werd in de analyse kon hier op een zelfde wijze geïmplementeerd worden. Enkel het op het juiste moment aansturen van de interne schrijf- en leessignalen, vergden wat meer werk, wanneer zulke operatie gedetecteerd werd.

3.4.1.3.2 Software

Het programma dat gebruikt werd om de parallelle poort aan te sturen werd geschreven in C. Het aanspreken van de parallelle poort gebeurt door de nodige data en adressen, via een eenvoudig commando, naar de juiste registers te sturen. De computer stuurt de parallelle poort dus aan, afhankelijk van wat er zich in de desbetreffende registers bevindt.

Met het EPP protocol, dient men normaal slechts de gewenste data naar het data register te schrijven. De handelingen nodig voor het versturen van wat er in het register staat, wordt verder verhandeld door de hardware. Zo heeft EPP een apart register voor data en adressen. Dit in tegenstelling tot SPP (standard parallel port), die hetzelfde register gebruikt voor data en adressen, maar waarbij de controlelijnen softwarematig moeten worden bediend.

De programmatie die hier werd uitgevoerd is eerder een implementatie van het EPP protocol in een SPP omgeving. Met andere woorden, het EPP protocol werd geëmuleerd met behulp van de SPP registers en softwarematige controle van de lijnen.

Dit maakte het debuggen van de hardware die moest geconstrueerd worden makkelijker. Voor de werkelijke implementatie van het EPP protocol (en dus geen emulatie ervan), was er onvoldoende tijd.

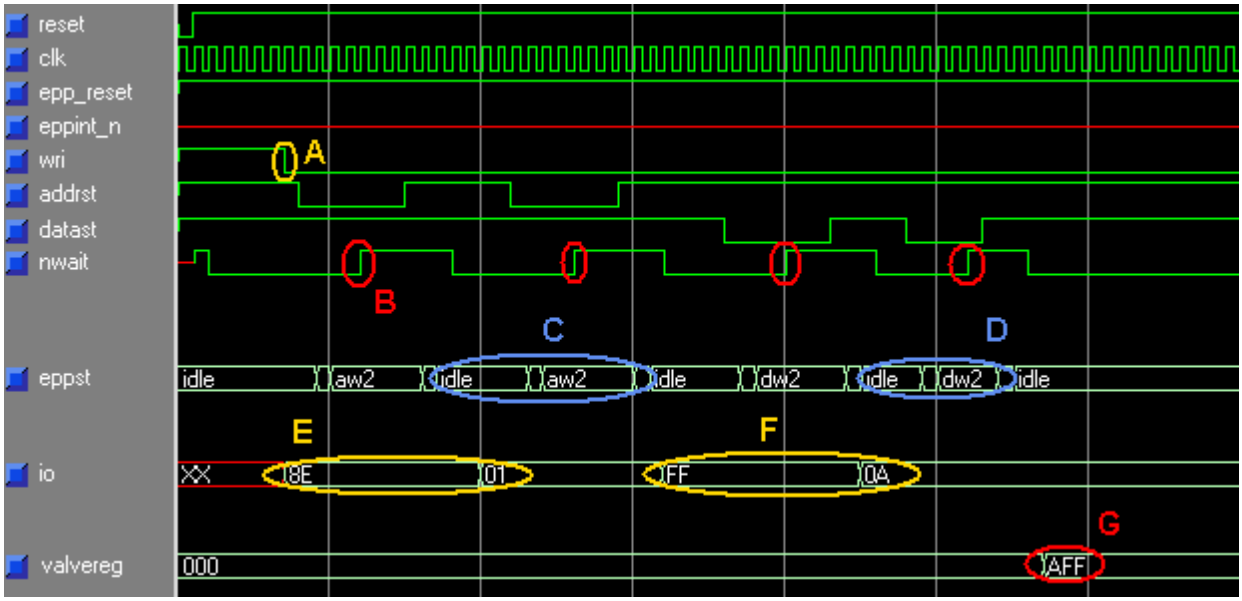
3.4.1.4 Simulatie

Om EPP te simuleren, moeten we een register kiezen waar we naar kunnen schrijven. Voor de eenvoud zullen we het kleppen-register nemen. Deze vraagt geen bijkomende verrichtingen in tegenstelling tot de SPI registers en kwadratuur decoder-registers.

In figuur 21 vindt men de relevante signalen terug van een schrijfoperatie [A] met behulp van het EPP protocol. De simulatie vertoont een correcte werking van zowel het EPP protocol als van de conversie van acht naar zestien bit, die verder in "3.5 Interne Communicatie" wordt besproken.

Bij het asserteren van het nwait-signaal [B], worden de datalijnen door de FPGA bemonsterd. Door dit hoge nwait-signaal weet de PC dat hij de schrijfoperatie mag afronden. Hij wacht dan ook tot deze terug laag komt te staan om een nieuwe operatie in te leiden. Bij een schrijfoperatie worden twee verschillende takken van de eerder besproken EPP finite state machine, namelijk address-write en data-write [C & D], steeds twee maal doorlopen. Deze zorgen ervoor dat de twee bytes van het adres [E] en de twee bytes van de data [F] correct worden ontvangen.

Zoals reeds eerder werd aangehaald, duidt de simulatie op een correcte werking van de EPP schrijfoperatie. De verzonden informatie [F] dat verstuurd werd naar adres dat overeenkomt met deze van het kleppenregister [E], is na verloop van tijd terug te vinden in dit register [G].

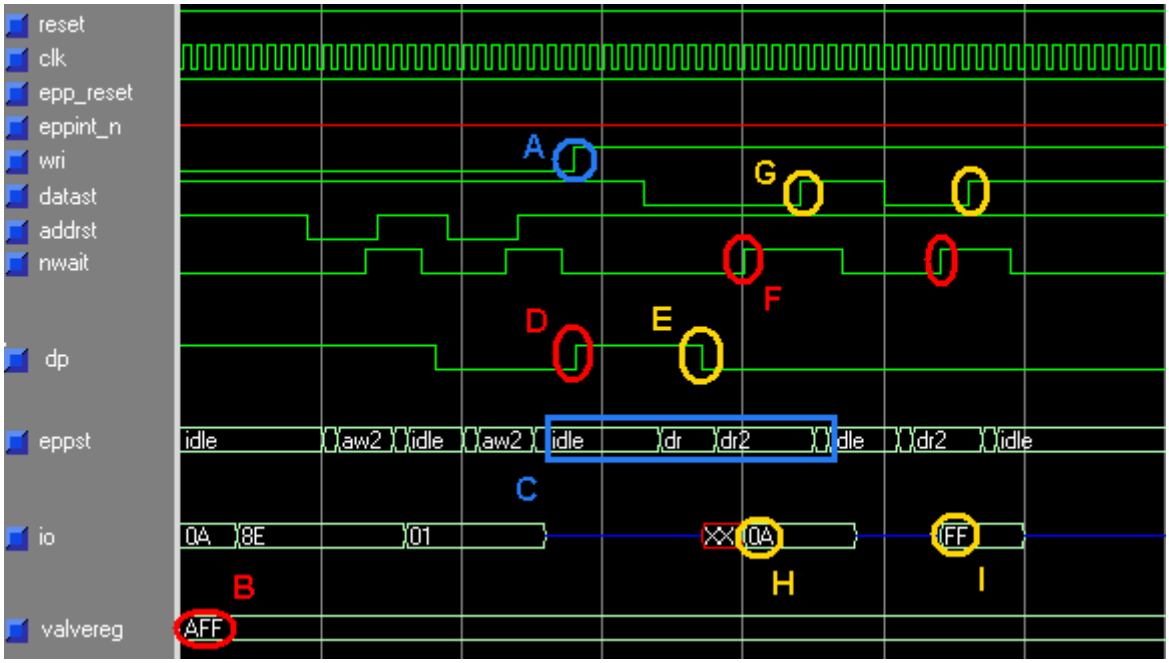


figuur 21 : GCI simulatie van een EPP schrijfoperatie

Voor een leesoperatie is de werkwijze gelijkaardig. Eerst moet het adres worden verstuurd zoals bij de schrijfoperatie. Vervolgens moet de PC het Write-signaal hoog plaatsen [A], zodat de FPGA kan uitgelezen worden. De waarde die we willen uitlezen, is deze die we zojuist bij de schijfoperatie naar het kleppenregister hebben geschreven.

Eens het adres door de FPGA werd ontvangen, moet geen tweede schijfoperatie gebeuren, maar een leesoperatie. Zodra de FPGA een leesoperatie waarneemt, zal deze op zoek gaan naar de data op het ontvangen adres [D]. Eens het adres in het extended-dataregister is geladen [E], kan de finite state machine voor een leesoperatie verder worden doorlopen [C].

Tijdens het hoog plaatsen van zijn nWait signaal [F], wordt de nodige data op de datalijnen geplaatst [H&I]. Op het ogenblik dat de PC zijn datastrobe terug ophoog trekt, bemonstert deze ook de stand van de datalijnen. Men merkt bijgevolg dat ook de leesoperatie correct verloopt.



figuur 22 : GCI simulatie van een EPP leesoperatie

3.4.1.5 Resultaten

De communicatie via de parallelle poort werkt naar behoren. Alle registers van de FPGA kunnen correct aangesproken worden. Ook alle functies die de FPGA moet kunnen uitvoeren, werden door middel van dit communicatiemiddel uitgetest.

Bij gebrek aan tijd, werkt men hier nog slechts met een EPP emulatie. Om het maximum uit dit communicatieprotocol te halen kan men best de hardware alle handshaking laten doen. Omwille van problemen bij de implementatie van deze hardwarematige handshaking werd geopteerd voor een softwarematige emulatie van dit protocol, zodat alle andere onderdelen zouden kunnen worden getest op hun werking. Het verschil met de hardwarematige handshaking laat zich merken in de snelheid van het data uitwisselingsproces.

Extra veel tijd investeren in de correcte uitwerking van dit protocol met hardwarematige handshaking leek overbodig. De kennis voor de correcte implementatie van dit protocol is reeds terug te vinden in een ander lopend eindwerk. Het gebruik van EPP als communicatievorm is ook slechts tijdelijk, daar het de bedoeling is over te gaan naar USB 2.0.

De tijd die hierbij werd bespaard, kon geïnvesteerd worden in het op punt stellen van de correcte werking van alle andere onderdelen, die van groter belang zijn voor toekomstige toepassingen.

3.4.2 USB 2.0

Een eindwerk handelende over de implementatie van USB voor gebruik met FPGA's werd reeds uitgevoerd op het departement, waardoor de kennis voor de implementatie van USB reeds aanwezig is. Zij het wel dat de toen geïmplementeerde standaard USB 1.1 niet voldoet aan de communicatie-eisen die hier gesteld worden. Daar de USB 1.1 standaard slechts in kleine mate verschilt van de USB 2.0 standaard, wordt de overdracht van de kennis naar USB 2.0 mogelijk geacht.

3.5 Interne Communicatie

De interne communicatie tussen de verschillende onderdelen gebeurt helemaal via het GCI (Global Communication Interface), door middel van 12 bit of 16 bit brede registers. Elk onderdeel geeft de nodige informatie door naar en krijgt informatie van zijn toegekend(e) register(s) in het GCI.

3.5.1 De registers

Het al of niet lees- of schrijfbaar zijn van een register heeft betrekking op een mogelijke externe communicatie via EPP. Hieronder wordt elk onderdeel van het LCI nagegaan naar zijn toegekende registers in het GCI. Naast de naam van de registers en hun doel, wordt ook de lees- en schrijfbaarheid ervan telkens toegelicht.

3.5.1.1 Kleppen aansturing

Het aansturen van de kleppen gebeurt door middel van slechts één register, het "Valve control register", waaraan het adres 0x8E werd toegekend. Dit register is verbonden met de Valve-controller component van het LCI. Om de kleppen aan te sturen moet naar dit register geschreven worden. De Valve-controller zal dan afhankelijk van de registerwaarde, de nodige aansluitingen aansturen. Er kan ook nagegaan worden welke kleppen open staan door dit register uit te lezen. Dit register is bijgevolg lees- en schrijfbaar.

3.5.1.2 Kwadratuur decoder

De kwadratuur decoder maakt gebruik van drie registers in het GCI, waarvan twee betrekking hebben op de positiebepaling en één op de snelheidsbepaling.

De snelheid die de kwadratuur decoder bepaalt, wordt telkens naar het overeenstemmende register van het GCI, het "Qdec speed register (Qdecs)" met adres 0x8C, geschreven. De waarde in het snelheidsregister is slechts van informatieve aard. Het is met andere woorden slechts een observatie van de huidige toestand en stuurt niets aan. Er moet enkel uit dit register gelezen worden en is bijgevolg niet schrijfbaar.

Het deel van de kwadratuur decoder die de positiebepaling verzorgt, gebruikt twee registers in het GCI. Het register die de huidige positie bijhoudt, , het "Qdec position register (Qdecp)", kan enkel gelezen worden omwille van dezelfde reden als het geval was voor het snelheidsregister. Maar de positie van de kwadratuur encoder kan ook geïnitialiseerd worden tot een bepaalde waarde.

De reden voor het niet lees- en schrijfbaar, maar enkel leesbaar maken van dit register volgt uit de volgende beschouwing : Het register moet steeds de huidige positie bevatten. Als een initialisatie waarde, ook de ijkingswaarde genoemd, wordt ingegeven, gaat deze nog niet onmiddelijk de plaats mogen innemen van deze huidige positie. De ijkingswaarde mag slechts de huidige waarde vervangen eens het referentiepunt voor de initialisatie van de kwadratuur encoder werd gedetecteerd. Tot men dit signaal ontvangt, mag de kwadratuur decoder geen rekening houden met de ijkingswaarde. Voor dit te verwezenlijken werd een tweede register aangemaakt, het "New Position Register".

Beide registers verbonden aan de positiebepaling worden om praktische redenen samengenomen als een virtueel lees- en schrijfbaar register voor de buitenwereld. Deze twee registers worden dus het "Quadrature decoder speed register" met adres 0x8A, die bijgevolg lees- en schrijfbaar is. Het "New Position Register" werd bijgevolg enkel schrijfbaar gemaakt.

3.5.1.3 SPI

Dit serieel protocol heeft heel wat meer registers nodig. Voor de correcte werking is een statusregister, een controleregister, een slave select register, een transmit register en een receive register nodig. Deze vijf registers bevatten alle nodige informatie voor het goede verloop van de seriële communicatie.

Opdat men de status van de SPI component zou kunnen in het oog gehouden, is het GCI voorzien van het "SPI Status Register [SPISR]" die kan aangesproken worden op het adres 0x80. Dit register bevat informatie als de toestand van registers "SPI Receive Register [SPIRR]" en "SPI Transmit Register [SPITR]" met de bits xmit_empty en rcv_full, de toestand van de bus met de bus busy [bb] & data transferring [dt] bits en de ontmoeting van problemen met de interrupt [int_n] en SPI error [spierr] bits. Hoewel dit register voor de gebruiker lees- en schrijfbaar lijkt, kan deze in werkelijkheid slechts gelezen worden. De schrijfoperatie is slechts bedoeld om de SPI error bit te wissen en kan dus niets rechtstreeks veranderen.

SPI status register operation										
	15	14	13	12	11	10	9	8	7	6	5 – 0
Read operation	dt	spierr	bb	int_n	xmit_empty	rcv_full	0	0	0	0	0
Write operation	#	#	#	#	#	#	#	#	#	spierr_reset	#

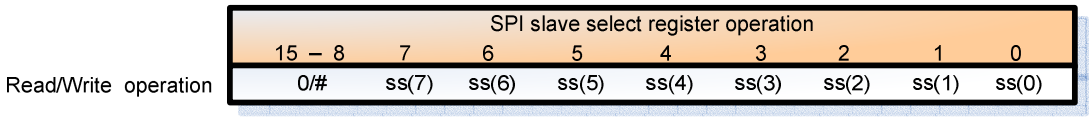
figuur 23 : SPI status register bit beschrijvingen

Het controleregister bevat de instelbare parameters die het verloop van de SPI communicatie in de hand houden. Deze parameters zijn de enable signalen, namelijk het "SPI enable" signaal (spien), die het mogelijk maakt SPI te gebruiken, en het "interrupt enable" signaal (inten), die de generatie van interrupts toelaat. De parameter "start" geeft aan dat een communicatie mag gestart worden. Tenslotte zijn er nog de parameters met betrekking tot de generatie en interpretatie van de seriële klok (clkdiv, cpha, cpol & rcv_cpol). Er moet logischer wijze naar dit register kunnen geschreven worden, maar ook de mogelijkheid om dit register uit te lezen werd voorzien. Dit "SPI Control Register [SPICR]" is dus lees- en schrijfbaar op het adres 0x82.

SPI control register operation								
	15	14	13	12 – 11	10	9	8	7 – 0
Read operation	spien	inten	start	clkdiv	cpha	cpol	rcv_cpol	0
Write operation	spien	inten	start	clkdiv	cpha	cpol	rcv_cpol	#

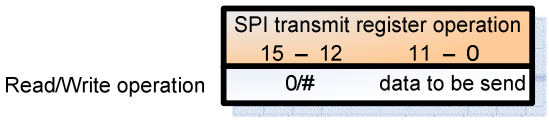
figuur 24 : SPI control register bit beschrijvingen

Voor de chips te selecteren waarmee men via SPI wil communiceren maakt men gebruik van het "SPI Slave Select [SPISS]" van het GCI op het adres 0x84. Dit register moet dus zeker kunnen beschreven worden. Ook hier werd de mogelijkheid tot het lezen van dit register ingebouwd om te kunnen nagaan welke chips geselecteerd zijn. Dit register is dus lees- en schrijfbaar.



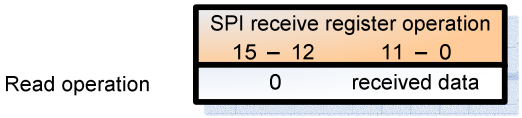
figuur 25 : SPI slave select register bit beschrijvingen

Indien men data wil versturen met het SPI protocol, moet deze naar dit "SPI Tansmit Register [SPITR]" worden geschreven. Eens een gegevensuitwisseling met SPI wordt geïnitialiseerd, vindt een overplaatsing van data van dit SPITR naar de SPI Transmitter component, met de nodige aanpassingen van de statussignalen. Dit register is lees- en schrijfbaar, daar ook hier de mogelijkheid tot uitlezing van de te verzenden data is voorzien.



figuur 26 : SPI transmit register bit beschrijvingen

Tenslotte is er nog het "SPI Receive Register [SPIRR]" op het adres 0x88. Dit register kan enkel uitgelezen worden, daar een schrijfoperatie naar dit register niet nuttig zou zijn. Dit register bevat de data die in de vorige SPI gegevensuitwisseling werd ontvangen.



figuur 27 : SPI receive register bit beschrijvingen

3.5.2 Het gegevensformaat

3.5.2.1 Analyse

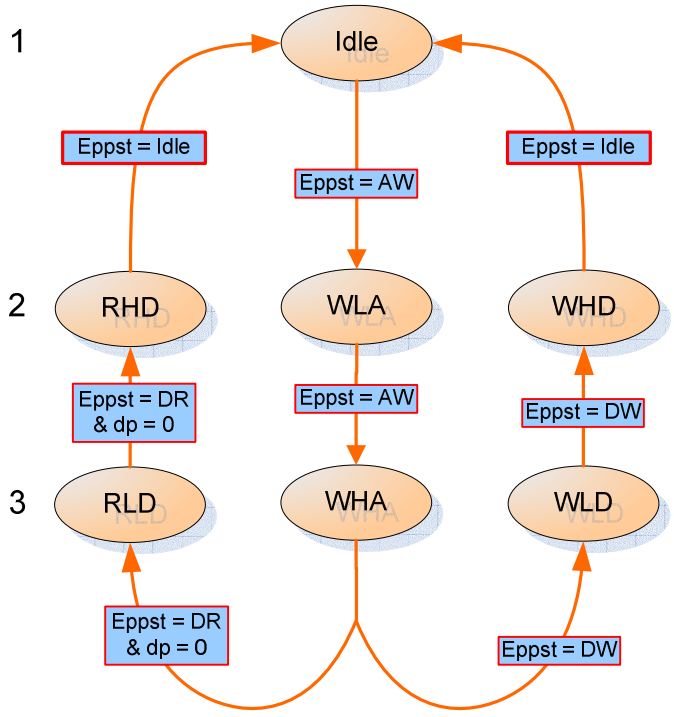
Zoals we reeds vroeger hebben aangehaald is de communicatie via de parallelle poort, een acht bit parallelle communicatie. De registers zijn echter telkens groter dan acht bit en kleiner of gelijk aan 16 bit. De beste aanpak is aldus de communicatie over 16 bit te laten plaatsvinden of telkens 2 transmissies van de parallelle communicatie samen te nemen.

Indien een transactie moet plaatsvinden, zij het een lees- of een schrijf operatie, is een adres telkens het eerste dat verstuurd moet worden. Eerst wordt de lage byte van het adres ontvangen, gevolgd door de hoge byte.

Het schrijven van gegevens naar een register gebeurt op gelijkaardige wijze, maar dan via een tijdelijk dataregister. Eens de 16 bits ontvangen zijn, moeten deze naar het geadresseerde register weggeschreven worden.

Leesoperaties gebeuren lichtjes anders daar er data uit de registers moet worden gehaald en naar de pc verstuurd. Hier moet weer eerst een adres ontvangen worden, daar men moet weten welk register men wenst uit te lezen. Eens een leesoperatie is waargenomen, wordt het geadresseerde register uitgelezen en in een tijdelijk dataregister bewaard. Deze data worden dan ook op hun beurt in 2 transmissies van telkens een byte doorgestuurd. Eerst wordt opnieuw de lage byte verzonden, gevolgd door de hoge byte.

In schematische vorm ziet de gegevensformaat-omzetting van acht naar 16 bits er als volgt uit:



figuur 28 : Finite state machine van de gegevensformaat-omzetting

3.5.2.2 Programmering

Het combineren van telkens twee 8 bit transmissies hangt nauw samen met de EPP FSM. Dit kon dus slechts geprogrammeerd worden nadat het EPP protocol reeds geprogrammeerd was en naar behoren werkte. Telkens een transmissie met het EPP protocol is voltooid, wordt de huidige toestand van de conversie in het FSM nagegaan.

Als deze zich bijvoorbeeld in Idle toestand bevindt, is de pas ontvangen byte de lage byte van het adres [WLA]. Op dezelfde wijze zal de volgende ontvangen byte bijgevolg in de hoge adres byte van het adresregister terecht komen [WHA]. De operatie die hierop volgt is weer afhankelijk van de toestand waarin de EPP FMS komt te staan.

Is de die toestand een leesoperatie, Data Read [DR], dan zal de data die zich op het ontvangen adres bevond, worden verstuurd op dezelfde wijze als het adres werd ontvangen: eerst de lage byte [RLD] gevolgd door de hoge byte [RHD].

Heeft men daarentegen te maken met een schrijfoperatie, Data Write [DW], dan wordt exact dezelfde werkwijze gevolgd als voor het ontvangen van het adres. Het enig verschil is de plaats waar de ontvangen gegevens in komen te staan. Aangezien men hier data ontvangt komen de ontvangen data in een tijdelijk data register in plaats van een adres register.

Hou er rekening mee dat twee opeenvolgende identieke voorwaarden niet juist na elkaar worden uitgevoerd, daar de EPP toestand zich slechts gedurende 1 klokperiode in de toestanden DR, AW en DW bevindt. De EPP finite state machine moet dus een tweede keer worden doorlopen om in dit finite state machine de tweede identieke voorwaarde te doorlopen.

Naast de gegevensformaat-omzetting, moeten ook andere signalen worden aangestuurd die aan de rest van het GCI laat weten dat respectievelijk het adres en de data volledig is ontvangen. Deze signalen zijn ingebed in deze FSM, evenals deze FSM ingebed is in de EPP FSM.

3.5.2.3 Simulatie

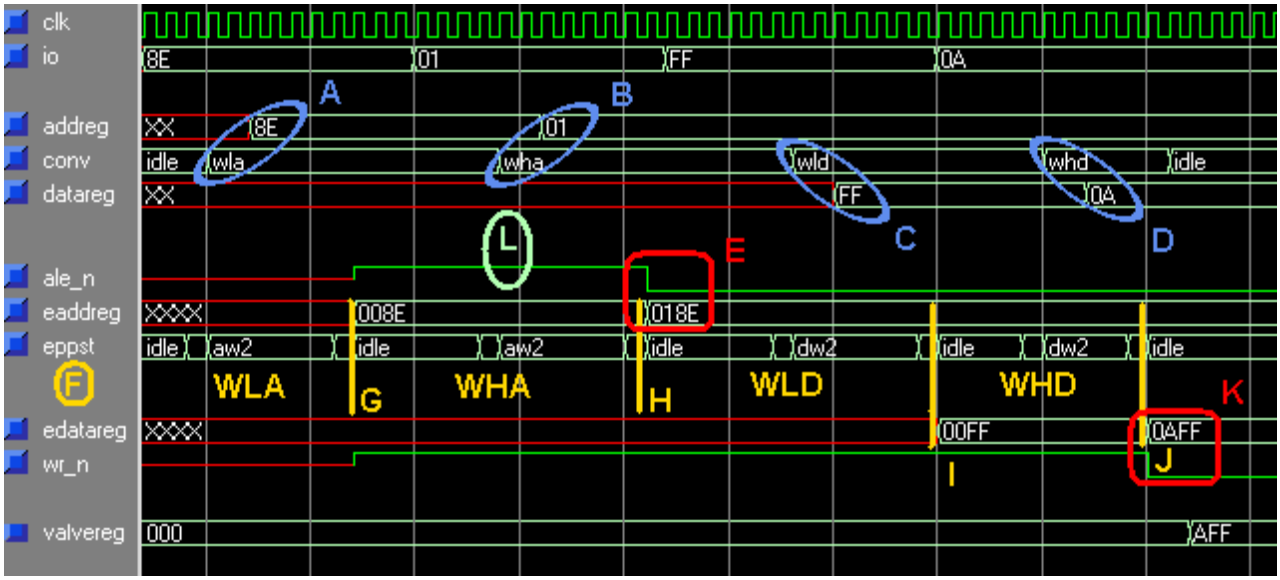
De simulatie van de gegevensformaat-omzetting gebeurt met dezelfde testbench[9] als bij het simuleren van het EPP protocol.Het verschil tussen de figuren ligt bij de getoonde signalen. Bij alle simulaties worden slechts de relevante signalen getoond. Voor de simulatie van het EPP protocol werden vooral de positie van de controle signalen met hun weerslag op EPP FSM besproken. Met de gegevensformaat omzetting zijn het eerder interne signalen die aan bod komen, daar deze omzetting ook volledig intern plaats vindt.

Aangezien men dezelfde testbench overneemt, heeft men ook hier te maken met twee verschillende operaties: de schrijfoperatie en de leesoperatie.

[9] Om het model van één of ander ontwerp te testen moet een ontwerper testpatronen toepassen op de inputpoorten en het gedrag van de outputpoorten in tijd waarnemen om te besluiten of de inputs naar de verwachte outputs werden omgezet. Het model wordt over het algemeen naar verwezen als een ontwerp in onderzoek (Design Under Test - DUT). Met het testen, ogen we het verifiërenen van de functionele juistheid en niet het vinden van productiefouten. De meest gebruikte methode om dit te verwezenlijken bestaat erin de DUT fysisch met een laag van code te omringen, die de stimulusgeneratie en output vergelijking uitvoert. Deze laag code, ook gekend als een testbench, kan zeer eenvoudig of zo complex zijn als nodig voor de betreffende toepassing.

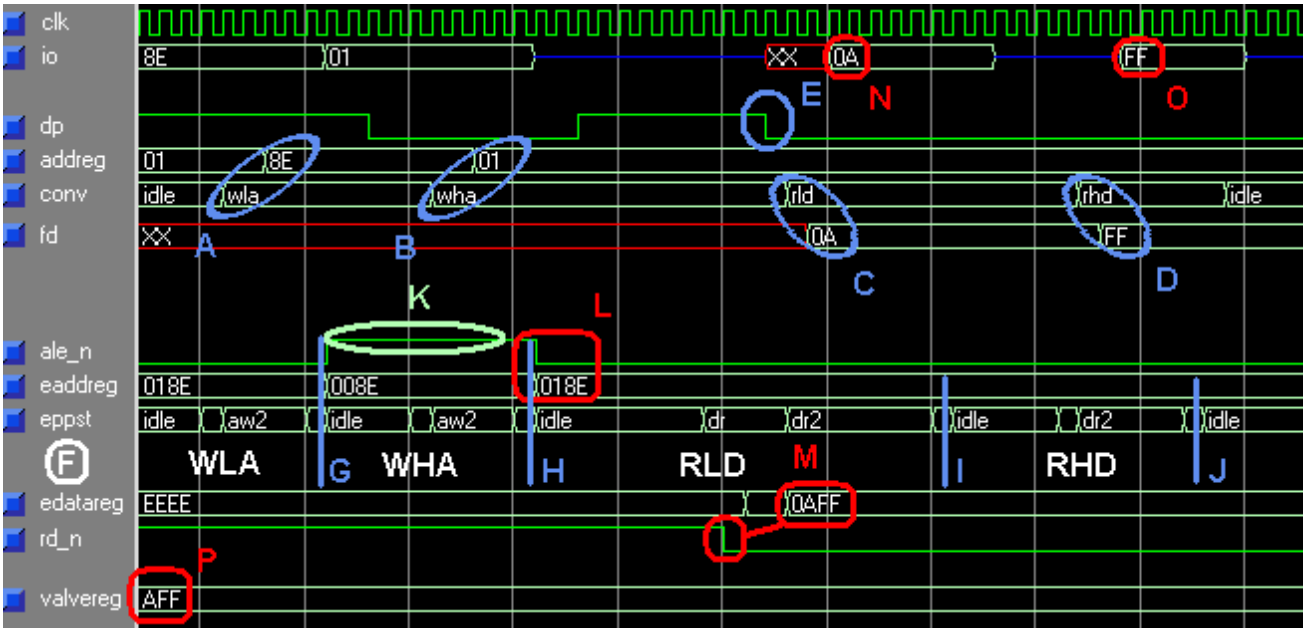
Op figuur 29 vindt men de relevante signalen terug bij een schrijfoperatie. Ter hoogt van [F] is weergegeven wat elke EPP transmissie doorgestuurd. Zoals reeds eerder werd vermeld, is hier ook te zien dat er vier schrijfoperaties gebeuren met behulp van het EPP protocol: twee voor het adres en twee voor de data.

Tijdens de eerste EPP transmissie, wordt de lage byte van het adres ontvangen [F-WLA]. Dit zorgt dat de conversie toestand van idle naar WLA verandert, waardoor de lage adresbyte in het tijdelijke adresregister opgenomen wordt [A]. Eens deze eerste EPP transmissie is voltooid [G], volgt een tweede transmissie [F-WHA], maar deze keer voor de hoge adresbyte [B]. Eens zowel de lage als de hoge adresbytes werden ontvangen, komt het Address Latch Enable (ale_n) signaal laag te staan [H]. Dit is een actief laag signaal en laat dus weten aan de rest van het GCI dat het adres volledig werd ontvangen [E]. De hoge adresbyte wordt dan vergeleken met het adres van de FPGA (base address). Als deze overeenkomen, wordt het register aangesproken die met de lage adresbyte overeenkomt. De operatie die hierop volgt is afhankelijk van de eerstvolgende EPP operatie.



figuur 29 : GCI simulatie van de gegevensformaat omzetting bij een schrijfopdracht

Indien het EPP write-signaal laag blijft, volgt een schrijfoperatie. Vanaf hier verlopen de zaken dus anders dan bij een EPP leesoperatie. De data wordt opnieuw in twee EPP transmissies ontvangen [F-WLD,C & F-WHD,D]. Eens beide databytes werden ontvangen [J], komt het CGI write-signaal laag te staan. De data die zich in het "extended dataregister" bevindt, kan op dat ogenblik naar het eerder aangesproken register worden geschreven.



figuur 30 : GCI simulatie van de gegevensformaat omzetting bij een leesopdracht

Heeft men echter een EPP write-signaal dat hoog wordt, dan volgt een leesoperatie. Hier wordt geen data ontvangen van de PC, maar ernaar verstuurd. Bij de detectie van een leesoperatie wordt in de EPP-DR toestand het, actief laag, read signaal (rd_n) laag geplaatst [M]. Dit laat toe om de data die zich bevindt in het register op adres [L] in te laden in het "extended data register" [M]. In de EPP-DR2 toestand, wordt de conversie FSM respectievelijk ingesteld op de RLD [C] en RHD [D] toestanden. In deze toestanden komen de respectievelijke bytes om beurt in het fd register te staan [C & D]. Dit register wordt bij leesoperaties met de input-output pinnen verbonden. De data aanwezig op de io pinnen [N & O] tijdens de twee EPP transmities komen overeen met de data die zich in het aangesproken register [P] bevindt. Men kan hier uit besluiten dat de gegevensformaat-omzetting ook bij een leesoperatie correct verloopt.

3.5.2.4 Resultaat

De verstuurde bytes komen wel degelijk terecht in de registers en dit in de juiste volgorde. Ook het uitlezen van de registers verloopt correct.

Nu zowel het EPP protocol als de gegevensformaat-omzetting werden besproken, heeft men een volledig zicht op de communicatiewijze tussen de PC en de GCI registers. Dit synthetiseert de volledige code die vereist is om de communicatie tussen de PC en de registers in de FPGA, via de parallelle poort, correct te laten verlopen.

4. Besluit

4.1 Beoordeling van de studie

De doelen van deze studie werden succesvol bereikt. De communicatie met de kwadratuur encoder, de kleppen en het seriële protocol gebruikt voor de communicatie met de ADC van de druksensoren, verlopen naar wens. Een deel van de bestaande architectuur is bijgevolg voortaan beschikbaar onder de vorm van een FPGA.

4.2 Sterke punten

Dit eindwerk had tot doel een eerste stap te zetten naar het vervangen van de hardware die de huidige communicatie verzorgt van de tweepotige robot Lucy. Elk van de onderdelen die hiervoor werden ontworpen kunnen ook voor andere toepassingen dienen, die gebruik maken van deze communicatiemiddelen. Door de indeling van de interne communicatie in drie entiteiten, kunnen deze gemakkelijk worden overgedragen naar andere designs die hiervan gebruik wensen te maken.

4.3 Zwakke punten

De uitlezing van de kwadratuur-positie bevat nog steeds occasionele fouten. Hoewel deze fouten gemakkelijk kunnen gedetecteerd worden, moet de oorzaak hiervan worden achterhaald. Een volledig correcte werking vermijdt extra code voor deze opvang, die het geheel vertraagt en een toekomstige overdracht van rekenwerk van de PC naar het sturingsplatform lastiger maakt.

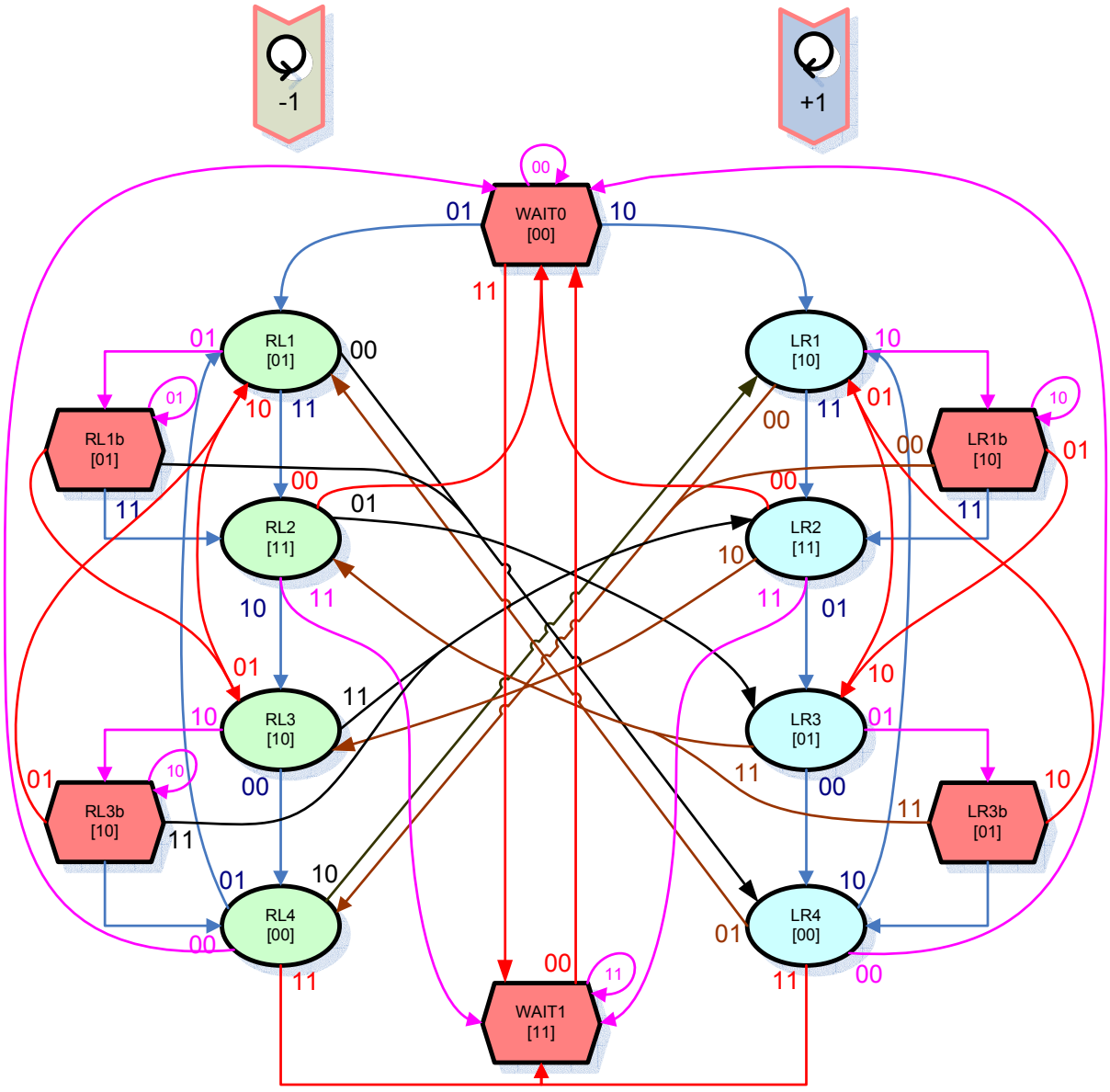
De communicatie met de PC die werd ontworpen is slechts een debug communicatie. Deze is geenszins bruikbaar voor de werkelijke implementatie van het design. Normaal moet, mits een aanpassing van de emulatie van het EPP protocol naar het echte EPP protocol, dit communicatie-protocol volstaan wat de snelheid betreft. Seriële communicatie is echter beter bestand tegen lange communicatiekanalen. Het is bijgevolg beter over te schakelen naar een USB communicatie. Deze is reeds aanwezig op de huidige communicatie hardware, maar kon bij gebrek aan tijd hier niet meer geïmplementeerd worden.

4.4 Toekomstig onderzoek

Nu de eerste stap werd gezet naar het werken met FPGA's voor de communicatie van Lucy, zou men de ontworpen componenten kunnen integreren in een groter design. De voorbije jaren werd reeds onderzoek gedaan naar dynamisch herconfigureerbare FPGA's. Deze techniek zou het mogelijk kunnen maken om met behulp van de geïntegreerde PowerPC's, sommige berekeningen over te brengen van de computer naar het sturingsplatform. Door deze dynamische herconfiguratie is op ieder moment slechts de logica aanwezig, nodig voor de uit te voeren processen. Dit vertaalt zich naar een compactere architectuur.

Appendix A. Volledige Kwadratuur decoder FSM

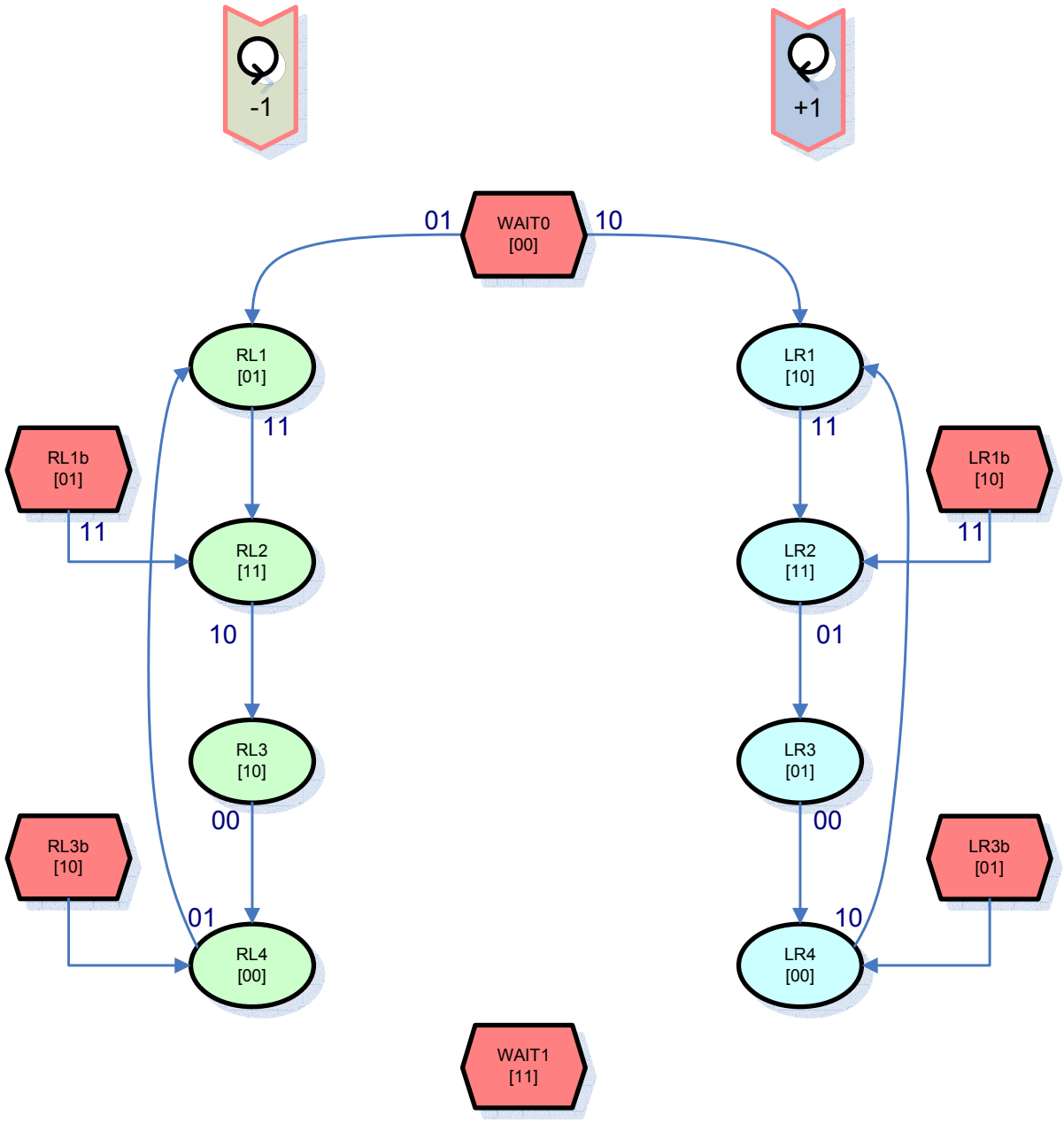
In deze appendix vindt u de volledige finite state machine die werd opgesteld om tot een correcte werking te komen van de Kwadratuur decoder. In Paragraaf 3.2 werden reeds de voornaamste eigenschappen verduidelijkt.



figuur 31 : Volledige Kwadratuur decoder FSM

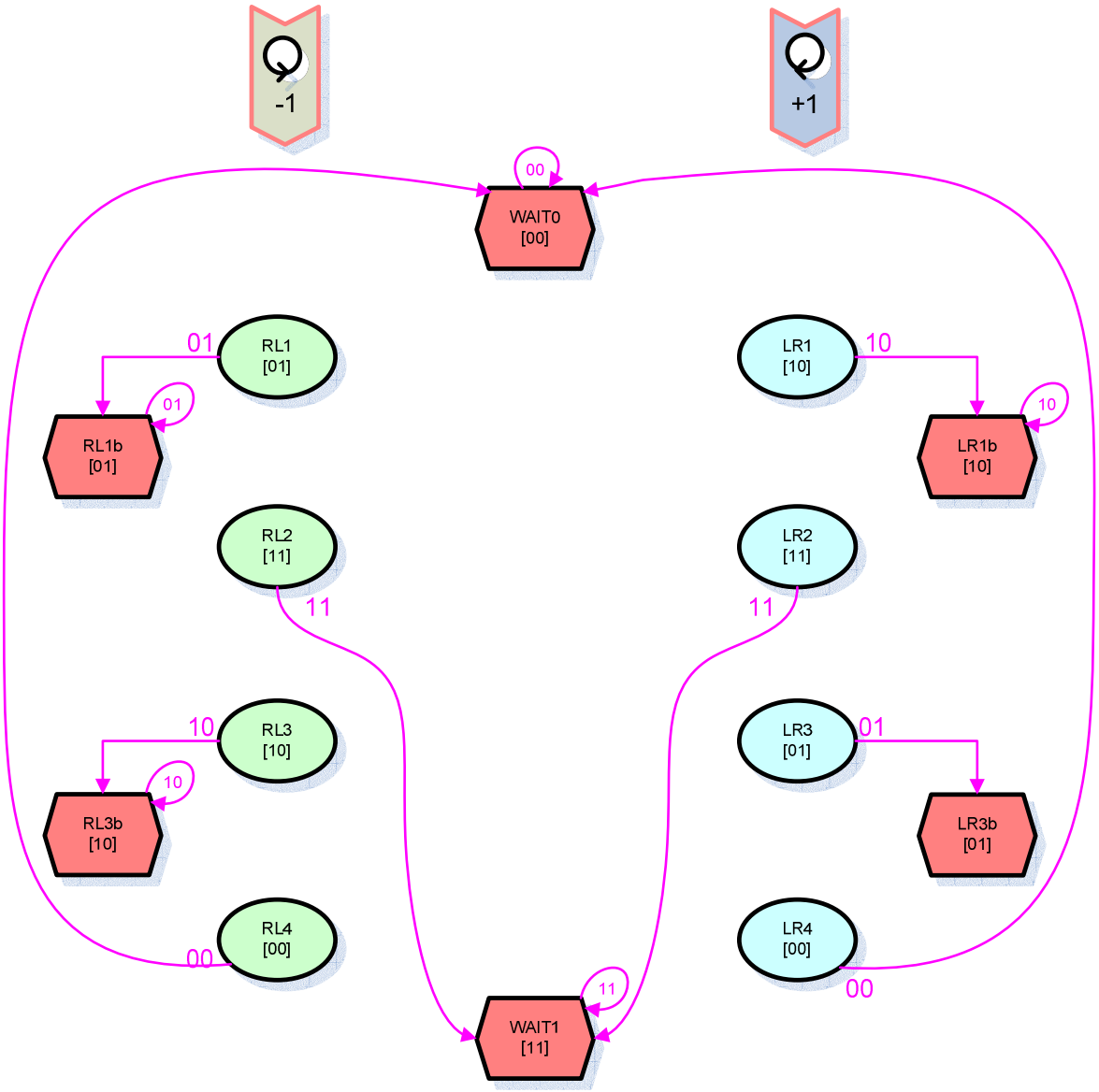
Op figuur 31 vindt u deze volledige FSM terug. Voor de duidelijkheid werden de pijlen van deze FSM onderverdeeld in verschillende categoriën. Elk van deze categoriën worden in de figuren hieronder apart afgebeeld.

Op figuur 32 vindt u de eerste categorie terug. Deze signalen vormen de normale overgangen tussen de verschillende toestanden zoals deze reeds terug te vinden zijn op het vereenvoudigde FSM. Daar voor één toestand van deze laatste er soms twee toestanden zijn in de uitgebreidere FSM, vormt dit schema geen FSM op zichzelf. Sommige toestanden zijn niet verbonden of niet voldoende.



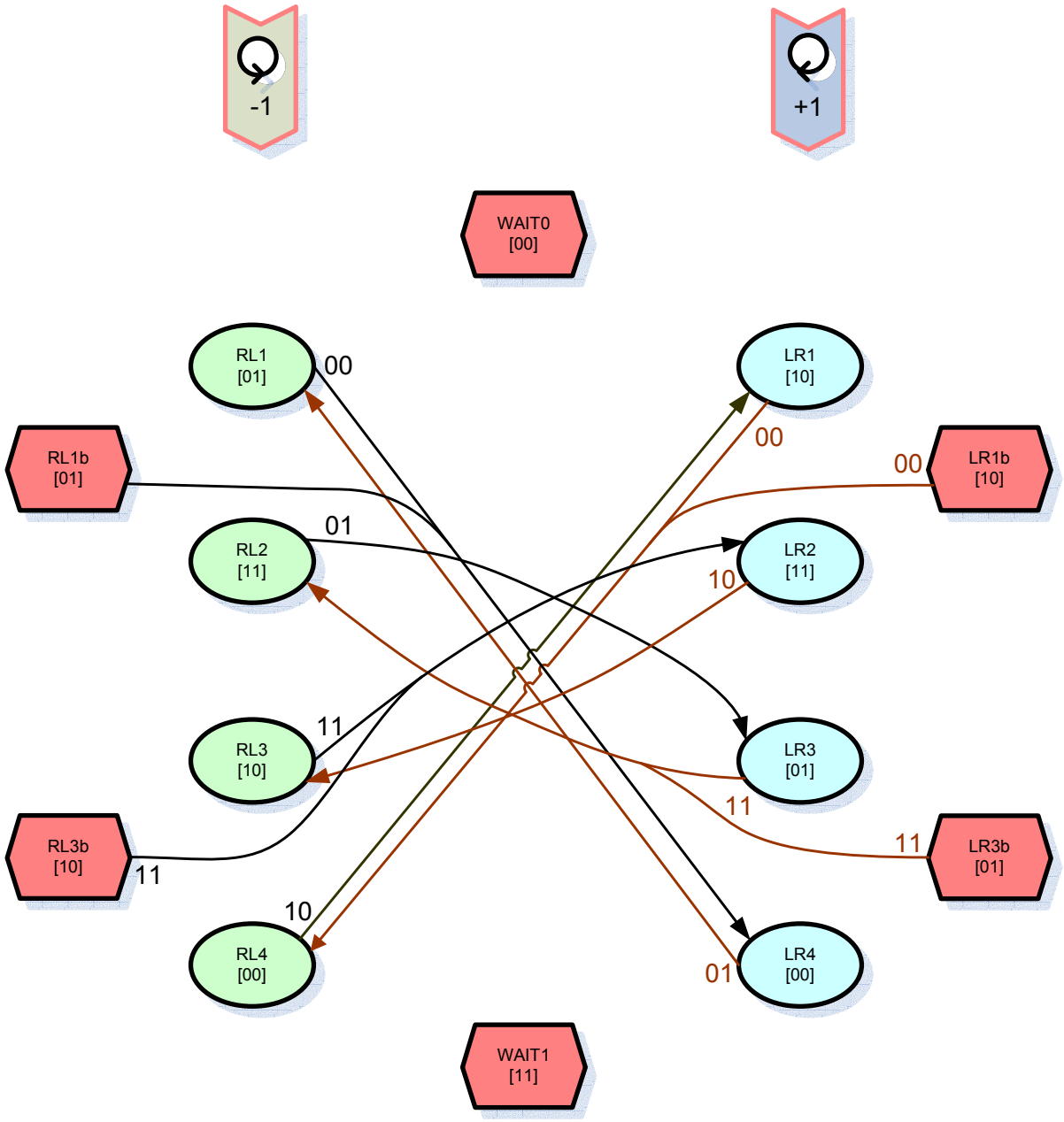
figuur 32 : FSM Kwadratuur decoder - Normaal verloop

Op figuur 33 zijn de overgangen terug te vinden waarbij twee opeenvolgende klokflanken dezelfde kwadratuur-waarde bezitten. Dit kan zich vertalen naar een overgang tussen twee toestanden waarbij de kwadratuur-waarde ook identiek is, of naar een overgang van een toestand naar zichzelf. De reden voor de aanwezigheid van toestanden met dezelfde kwadratuur-waarde ligt in volgende beschouwing: Als we aannemen dat bij elke klokflank de kwadratuur-waarde met één verschilt, moet bij elke klokflank de kwadratuur-teller geïncrementeerd of decrementeerd worden. In elk van deze toestanden moet dus de teller verhoogd of verlaagd worden. Neem nu dat de kwadratuur-waarde niet bij elke klokflank verandert. Bijgevolg zou dezelfde toestand nog eens doorlopen worden en de teller nog eens geïncrementeerd worden, hoewel de kwadratuur-waarde dezelfde is gebleven.



figuur 33 : FSM Kwadratuur decoder - zelfde toestand

Om dit te vermijden werden extra toestanden toegevoegd, die als twee of meer opeenvolgende klokflanken met dezelfde kwadratuur-waarde worden aangesproken. Deze toestanden doen op zich niets en wachten op een kwadratuur-waarde verandering.

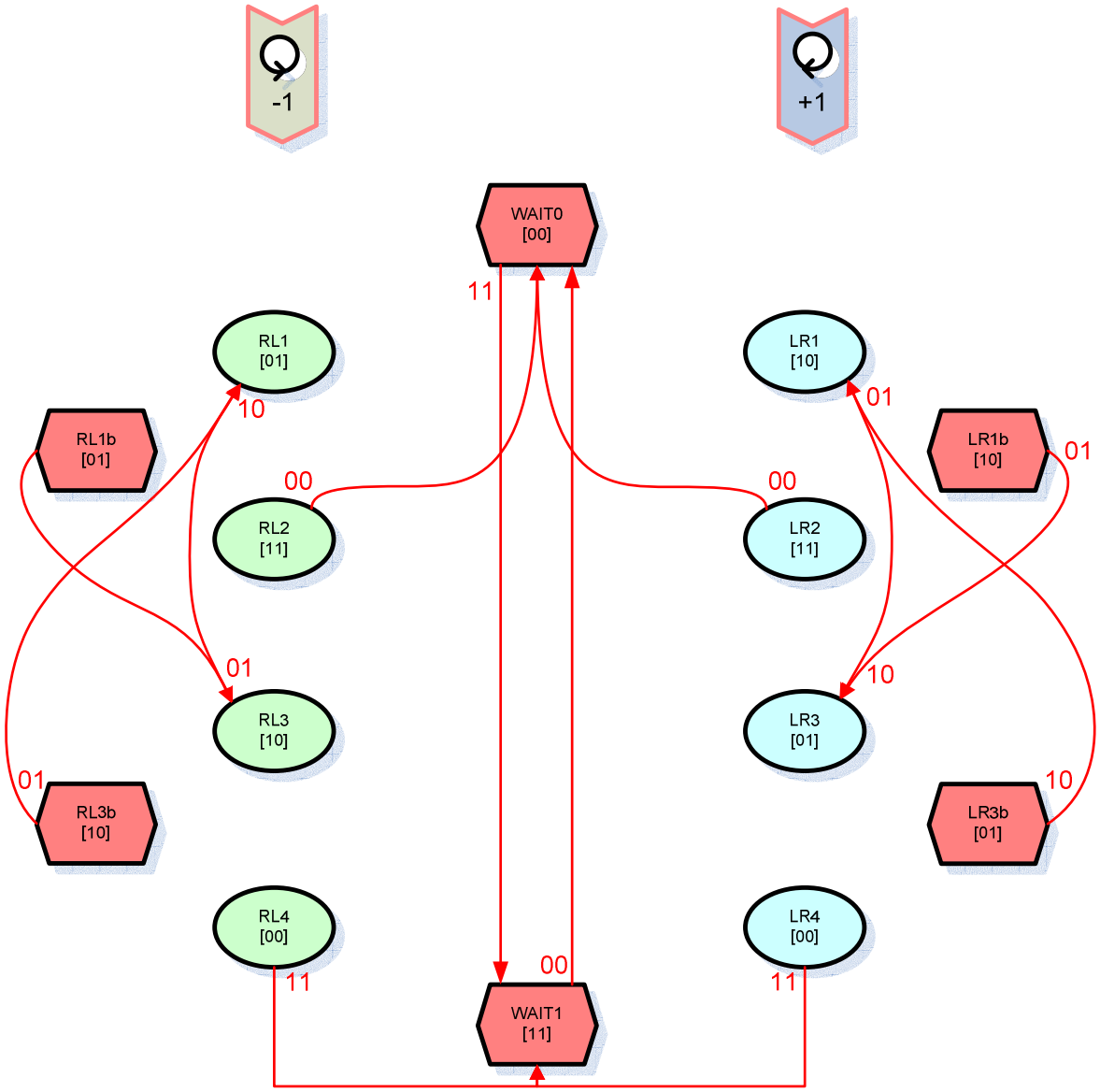


figuur 34 : FSM Kwadratuur decoder - Draaizin wijziging

figuur 34 toont de overgangen tussen de toestanden bij een draaizin-wijziging. Hier valt de kwadratuur-waarde dus terug op zijn vorige waarde. Daar men bij een tegengestelde draaizin, een tegengestelde operatie moet uitvoeren, mag hier niet gewoon naar de vorig toestand worden gegaan. Zou men dit doen, dan wordt de kwadratuur-teller bijvoorbeeld nog eens verhoogd in plaats van één maal verlaagd.

De laatste overgangen die men op de FSM terugvindt zijn de signalen ter vervollediging van de mogelijkheden bij elke toestandsovergang. Ook onmogelijke, lees niet gewenste, overgangen moeten worden opgenomen in het FSM. Op figuur 35 vinden we deze signalen terug.

De kwadratuur-waarde verandert bij elke overgang telkens slechts in één van zijn bits. Bij een sprong waarbij de twee bits veranderen, is er iets misgelopen. Een overgang werd gemist. Deze voorvallen moeten ook voorzien worden. De getroffen maatregelen zijn niet telkens correct, maar minimizeren de fout die wordt opgelopen.



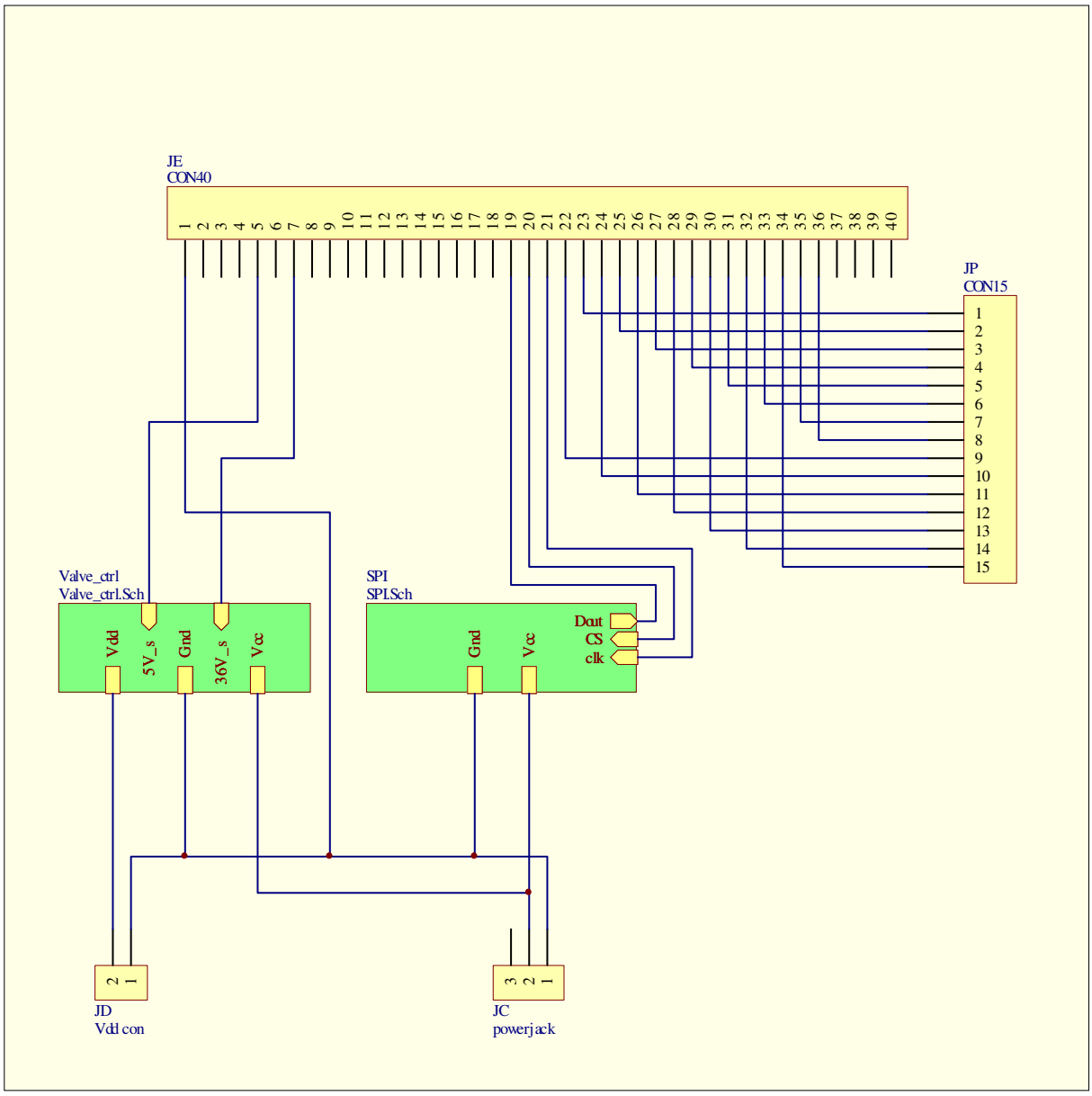
figuur 35 : FSM Kwadratuur decoder - Opvang van foutieve gebeutenissen

Appendix B. Uitbreidingsbordje

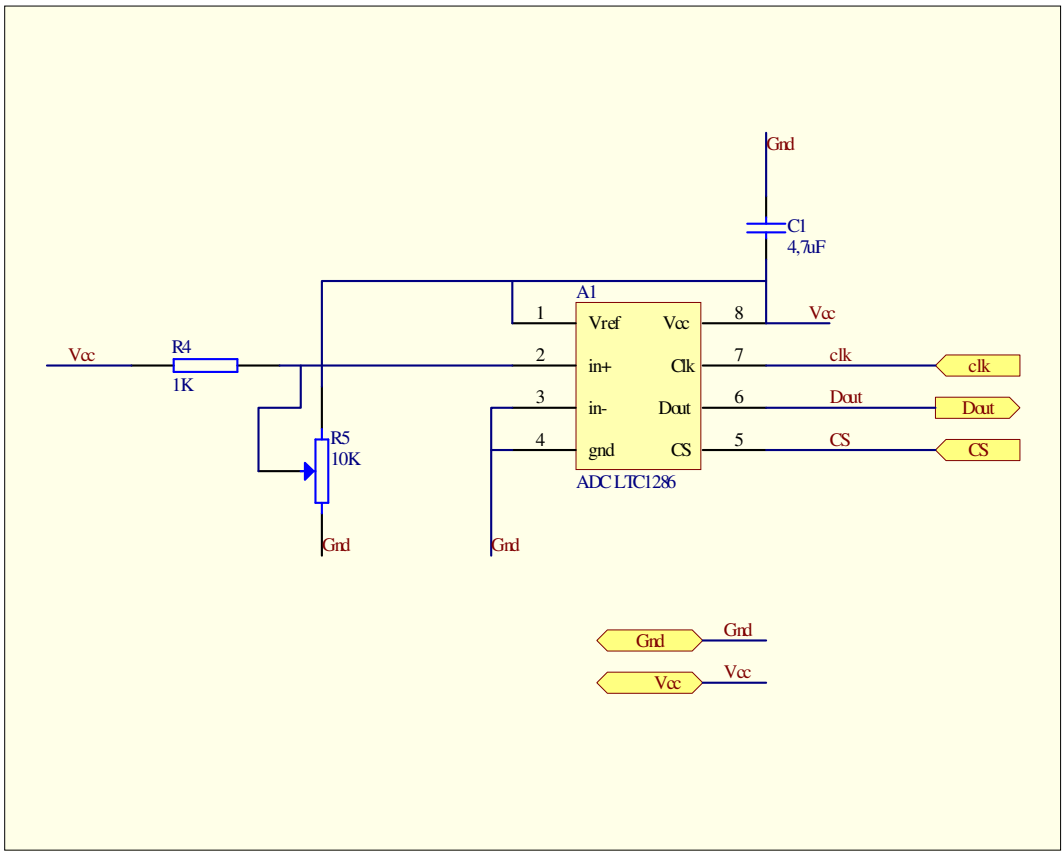
Dit bordje werd, zoals de naam laat blijken, ontworpen als uitbreiding voor het gebruikte testbordje van Digilent. Deze uitbreiding moest het mogelijk maken de parallelle poort van de PC te verbinden met het testbordje, zodat de verschillende onderdelen van de communicatie zouden kunnen worden getest.

Dit uitbreidingsbordje werd ook voorzien van een schakeling die de verbinding met de druksensoren kon nabootsen. Om praktische redenen werden de sensor en verschilversterker vervangen door een potentiometer. Met deze potentiometer kon men, evenals bij het gebruik van de originele schakeling, de spanning aan de ingang van de ADC, dewelke representatief is voor de druk, wijzigen. Door deze wijziging zou de informatie afkomstig van deze ADC verschillen, zodat de goede werking van de code kon worden nagegaan. Ten laatste werd het uitbreidingsbordje ook van de speed-up schakeling voorzien, dewelke niet in gebruik werd genomen.

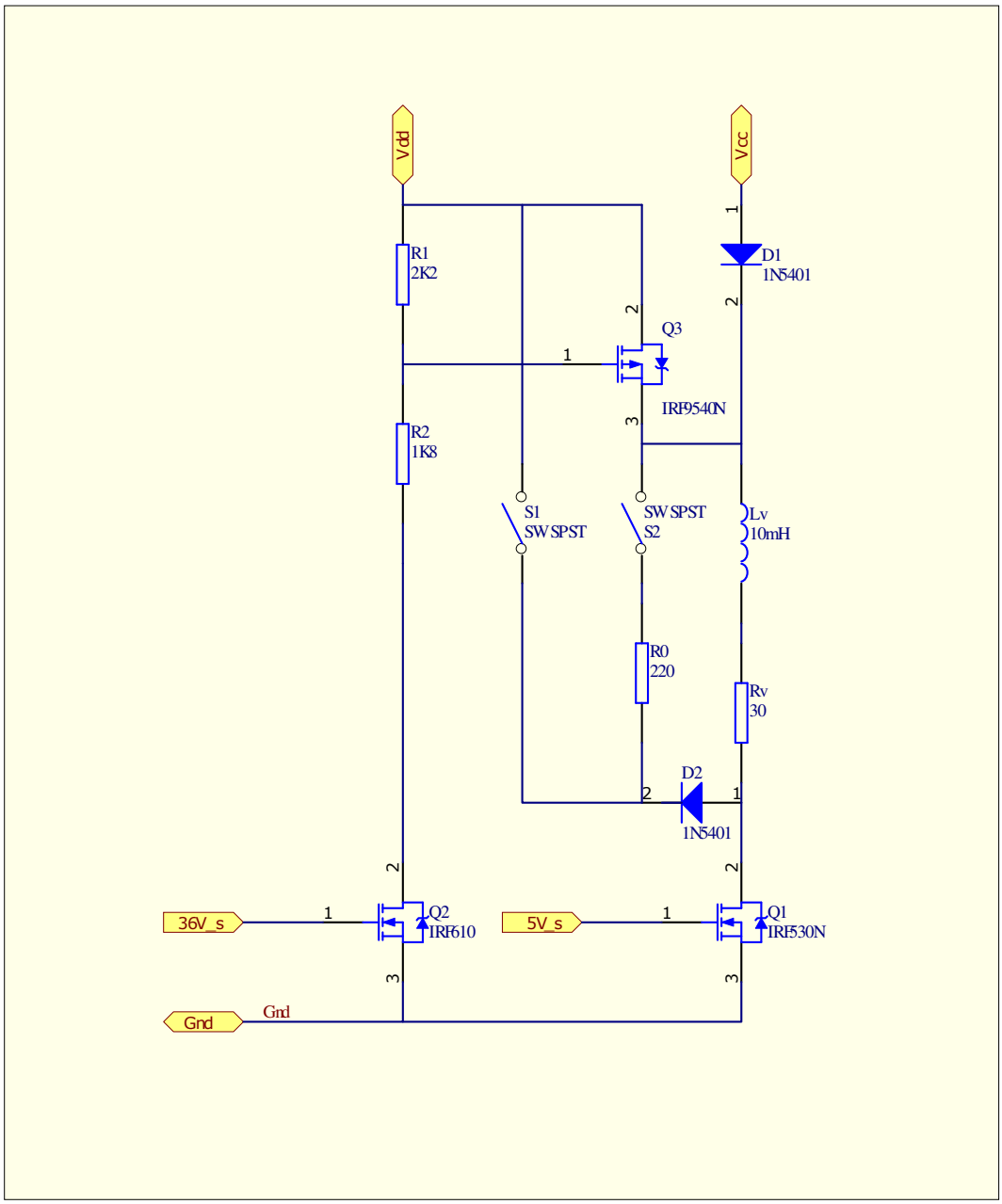
Op de volgende bladzijden vindt u deze verschillende schakelingen terug evenals de PCB afdruk van het volledige uitbreidingsbordje.



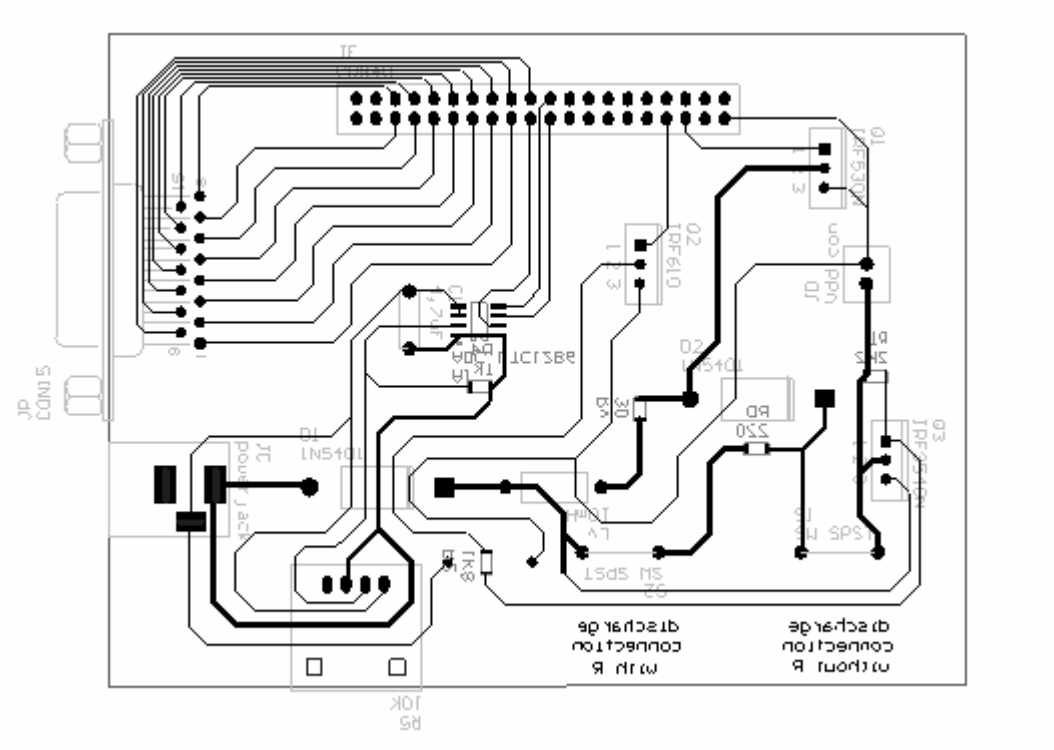
figuur 36 : Uitbreidingsbordje - Toplevel



figuur 37 : Uitbreidingsbordje - ADC schakeling



figuur 38 : Uitbreidingsbordje - Speed-up schakeling



figuur 39 : Uitbreidingsbordje - PCB

Bibliografie

Vanderborght B. [2003], *Quasi-statische controle van een stappende robot aangedreven door actuatoren met regelbare stijfheid*, Master thesis, Vrije Universiteit Brussel.

Verrelst B. [2005], A dynamic walking biped actuated by pleated pneumatic artificial muscles: Basic concepts and control issues, PhD thesis, Vrije Universiteit Brussel.

Vandenhoudt J. [2002], *PWM-sturing van een antagonistisch paar pneumatische artificiële spieren*, Master thesis, Vrije Universiteit Brussel.

Craig Peacock [1998], *Interfacing the standard parallel port,* Beyond logic, http://www.beyondlogic.org.

Craig Peacock [2002], *Interfacing the enhanced parallel port,* Beyond logic, http://www.beyondlogic.org.

Digilent [2004], *Pegasus board reference manual*, http://www.digilentinc.com

Xilinx [2002], *Serial Peripheral interface master*, Application note, http://www.xilinx.com

Xilinx [2000], *Implementing HDL with WebPACK ECS Schematic Editor*, Application note, http://www.xilinx.com

Xilinx [1995], *Quadrature Phase decoder,* Application note, http://www.xilinx.com

Motorola [2004], *SPI Block guide*, http://e-www.motorola.com

Motorola [1996], *QSM reference manual*, http://www.freescale.com

MEER OVER LUCY :

Vanderborght B., Verrelst B., Van Ham R., Vermeulen J., Naudet J. & Lefeber D. [2004], *Control architecture of LUCY, a Biped with Pneumatic Artificial Muscles.*, CLAWAR 2004 7th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines Madrid, Spain.

Verrelst B., Daerden F., Lefeber D., Van Damme M., Vanderborght B., Van Ham R. & Vermeulen J. [2003], *Pleated Pneumatic Artificial Muscles for Robotic Applications,* 1st Flanders Engineering PhD Symposium, Brussels.

Van Ham, R., Verrelst, B., Daerden, F. & Lefeber, D.[2003], *Pressure control with on-off valves of Pleated Pneumatic Artificial Muscles in a modular one-dimensional rotational joint,* International Conference on Humanoid Robots, Karlsruhe and Munich.

Verrelst, B., Van Ham, R., Vermeulen, J., Lefeber, D. & Daerden, F. [2003], *Concept of combining adaptable passive behaviour with an active control structure using Pleated Pneumatic Artificial Muscles for the bipedal robot LUCY*, International Conference on Humanoid Robots, Karlsruhe and Munich.

Van Ham, R., Daerden, F., Verrelst, B.& Lefeber, D. [2003], *Control of a joint actuated by two pneumatic artificial muscles with fast switching on-off valves*, 6 th National congress on theoretical and applied mechanics, Ghent.

Verrelst, B.& , Van Ham, R., Vanderborght, B., Lefeber, D. & Daerden, F. [2003], *Lucy: a walking robot*, 6th National congress on theoretical and applied mechanics, Ghent.

Verrelst, B., Van Ham, R., Daerden, F.& Lefeber, D. [2002], *Design of a biped actuated by pleated pneumatic artificial muscles*, 5th International conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Paris.

Van Ham, R., Daerden, F., Verrelst, B., Lefeber, D. & Vandenhoudt, J. [2002], *Control of pneumatic artificial muscles with enhanced speed up circuitry*, Proceedings of the 5th International conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Paris.

Daerden, F., Lefeber, D., Verrelst, B., & Van Ham, R. [2001], *Pleated pneumatic artificial muscles: actuators for automation and robotics,* 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Como.

Daerden, F., Lefeber, D., Verrelst, B., & Van Ham, R. [2001], *Pleated pneumatic artificial muscles: compliant robotic actuators,* 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, Hawaii.

Verrelst, B., Daerden, F., Lefeber, D., Van Ham, R., Fabri, T. [2000], *Introducing Pleated Pneumatic Artificial Muscles for the actuation of legged robots : a one-dimensional setup*, CLAWAR 2000: 3rd International Conference, Madrid.