KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Ontwerp van een quadcoptersturing in VHDL

Andries Gert-Jan

Master of Science in de industriële wetenschappen: elektronica-ICT, optie Elektronica

Promotor:
Ing. Sammy Verslype

Academiejaar 2014 – 2015

Inhoudsopgave

Samenvatting

In dit projectlab wordt een quadcoptersturing ontworpen op een FGPA. Aan de hand van een PID regeling wordt de stabilisatie voorzien rond de drie bewegingsassen van de quadcopter. Vanwege de beperkte hoeveelheid resources die op de FPGA aanwezig zijn, dient er in het hardware ontwerp rekening gehouden te worden met deze beperking. Verschillende iteraties en methodieken worden ondernomen om de benodigde resources voor deze PID regeling tot een minima te beperken. Zo werd onderzocht of er gebruik gemaakt kan worden van de snelle DSP48 slices. Daarnaast wordt ook een implementatie gemaakt aan de hand van de IP CoreGenerator multiplier.

Naast het ontwerpen van een PID controller worden ook verschillende andere systeemcomponenten ontworpen. Een PWM generator zal de BLDC motoren voorzien van een gepast PWM signaal dat voldoet aan de frequentievoorschriften van de motorfabrikant. Om het juiste signaal voor elke motor te genereren, wordt een motormixing unit ontworpen die het motormixingalgoritme implementeert.

Communicatie met de buitenwereld wordt voorzien door middel van een SPI slave interface. Deze interface wordt gebruikt om sensorwaarden te ontvangen, maar laat de gebruiker ook toe bepaalde registers uit te lezen of in te stellen.

Alle hardware wordt gesimuleerd en vervolgens getest op een FPGA. Via Matlab en Simulink worden vooraf de nodige berekeningen en simulaties uitgevoerd die als basis voor het FPGA design dienen.

Naast de hardware wordt ook het quadcopterframe volledig ontworpen. Dit frame wordt zodanig ontworpen dat het uit verschillende onderdelen bestaat die door een 3D printer vervaardigd kunnen worden.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen

ADC	Analog to Digital Converter
ARM	Advanced RISC Machines
AHRS	Attitude and Heading Reference System
BLDC	Brushless DC
CPHA	Phase Polarity
CPOL	Clock Polarity
DC	Direct Current
DMP	Digital Motion Processing
DSP	Digital Signal Processing
ESC	Engine Speed Controller
FPGA	Field Programmable Gate Array
I^2C	Inter-IC bus
IC	Integrated circuit
IMU	Inertial Measurement Unit
IO	Input Output
IP	Intellectual Property
KU	Katholieke Universiteit
KUL	Katholieke Universiteit Leuven
MAC	Multiply and Accumulate
MAVLink	Micro Air Vehicle Communication Protocol
MHz	Megahertz
MISO	Master In Slave Out
MOSI	Master Out Slave In
PCB	Printed Circuit Board
PID	Proportioneel Integrerend Differentiërend
PWM	Pulse Width Modulation
RF	Radio Frequency
SCLK	Serial Clock Line
SM	Sliding Mode
SPI	Serial Peripheral Interface
SS	Slave Select
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Hoofdstuk 1

Inleiding

In dit verslag worden de resultaten van het projectlab toegelicht. Elke stap die werd genomen om het eindresultaat te bekomen wordt kort besproken, alsook de moeilijkheden die zich hebben voorgedaan tijdens de ontwikkeling. Tenslotte wordt ook een besluit gevormd over het gehele project en worden de toekomstmogelijkheden kort toegelicht.

1.1 Omschrijving van de opdracht

Het project is onderverdeeld in twee grote onderdelen. Enerzijds is er het stabilisatiealgoritme voor de quadcopter en anderzijds is er de sturing, hardware design en optical flowintegratie. De focus van het project zal gelegd worden op de stabiliteitsnelheid van het eindresultaat. Het onderdeel met het stabilisatie-algoritme omvat het genereren van PWM signalen die de motoren aansturen. Hiervoor is sensordata nodig van bijvoorbeeld een gyroscoop, accelerometer, magnetometer en barometer. Om de ruis uit deze metingen weg te filteren zal gebruik gemaakt worden van een filter, bijvoorbeeld Kalman, die ook in de FPGA zal geïntegreerd worden. Naast data van sensoren zal de FPGA ook informatie ontvangen van een extra processor. Op deze extra processor wordt de communicatie met de grond voorzien (besturing), alsook een optical flow algoritme.

De stabilisatie zelf zal uitgevoerd worden door een PID of SM (sliding mode) dat ook op de FPGA zal geïntegreerd worden. De instellingen van het stabilisatie-algoritme zullen gemaakt worden door middel van registers die door de externe processor kunnen ingesteld worden via I^2C of SPI. Via deze registers kan ook de actuele sensor- en systeeminformatie opgevraagd worden. De hardware bestaat uit een zelf te ontwikkelen printplaat gebaseerd op een FPGA (Spartan 6) die de stabilisatie en verwerking van de inputs verzorgt. Daarnaast bevat het ook een microcontroller die de communicatie voorziet, alsook de beeldverwerking voor de optical flow en het batterijbeheer. De communicatie zou verlopen volgens het MAVLink protocol, een gestandaardiseerd protocol dat communicatie voorziet onafhankelijk van de fysische laag waarop dit gebeurt. Een keuze voor het beste kanaal maakt ook deel uit van dit project.

Optical flow detectie voorziet de mogelijkheid om de horizontale ongewenste bewegingen tegen te werken en dus de quadcopter op een vaste positie te houden. De sturing voor de BLDC motoren wordt zelf geïmplementeerd en vindt plaats op externe processoren die elk instaan voor één enkele motor. Het FPGA ontwikkelbord waarmee gewerkt zal worden is het Mojo bord met een Spartan 6 processor. Dit bord zal zelf aangekocht worden. Verder zal ook een MPU9250 gebruikt worden om accelero-, magneto- en gyro-data te verzamelen. De andere sensoren die gebruikt zullen worden dienen te worden bepaald tijdens de looptijd van het project.

Wanneer alle sensoren getest zijn, worden ze samen met de FPGA (niet het ontwikkelbord) op de zelf ontworpen printplaat geplaatst om zo een quadcoptersturing te verkrijgen die onafhankelijk is van een ontwikkelbord of opgelegde pinout. Deze wordt gemonteerd op de quadcopter structuur die gemaakt zal worden door middel van een 3D printer. Dit maakt het nadien ook mogelijk om bij een crash een bepaald onderdeel snel te reproduceren.

Dit projectlab kan worden opgesplitst in verschillende milestones.

- Stabilisatie en sturing (Gert-Jan)
  - Uitlezen sensoren
  - Uitfilteren van ruis op sensordata
  - Stabilisatie-algoritme PID/SM
  - Instellingsmogelijkheid van registers
  - Ontwerp van quadcopterframe
- Hardware en optical flow (Nick)
  - Ontwerpen printplaat
  - Implementeren MAVLink
  - Selecteren communicatiekanaal
  - Optical flow
  - BLDC motor sturing
  - Return to home functie

1.2 Planning en werkwijze

De algemene werkwijze van het project kan onderverdeeld worden in vier grote blokken.

Onderzoek: In deze fase van het project wordt een onderzoek uitgevoerd met betrekking tot quadcopters. Er wordt gekeken naar hoe een quadcopter bestuurd wordt en via welke algoritmen dit kan gebeuren. Daarnaast zal ook een studie gemaakt worden naar het implementeren van een PID-controller in een FPGA. Via simulaties in het programma simulink kunnen de algoritmen geëvalueerd worden.

Implementatie: In deze fase worden de onderzoeksresultaten omgezet in code. Via het programma Xilinx ISE en Vivado wordt de implementatie voorzien op een FPGA. Aan de hand van het programma Modelsim worden de hardwaredesigns geëvalueerd alvorens deze op de FPGA worden getest.

Testen: Wanneer de hardware gevalideerd is door middel van een testbench, kan deze getest worden op de FPGA zelf. Blok per blok zullen de nodige testen uitgevoerd worden en bijgestuurd waar nodig.

Reflectie: Elk ontworpen blok zal onderworpen worden aan een kritische evaluatie. Hierbij wordt gekeken naar performantie, accuraatheid en werking binnen het geheel. Waar nodig wordt bijgestuurd. Regelmatige communicatie met medestudent Nick, die de hardware (PCB) ontwerpt is hierbij nodig.

Een beknopte projectplanning wordt hieronder weergegeven. Een uitgebreide versie hiervan is terug te vinden in appendix A op pagina 30. Deze uitgebreide versie houdt rekening met de initiële planning. Hieronder is het werkelijke verloop van het project weergegeven.

Week 1:
- Keuze maken van het project uit de aangeboden projecten.
- Registreren van het gekozen project.

Week 2:
- Indienen van de projectplanning.
- Onderzoek naar quadcopters en besturing.
- Studie PID en digitalisering.
- Studie IMU sensoren.

Week 3:
- Simulatie van PID algoritme in Matlab en Simulink.
- Theoretisch ontwerp van PID door middel van DSP48 slices.
- Kennismaking Mojo ontwikelbord en Xilinx ISE.

Week 4:
- Tussentijdse presentatie van het project.
- Implementatie van PID door middel van DSP48 slices .
- Evaluatie van PID implementatie en bijsturing.
- Ontwerp van quadcopter frame in Inventor.

Week 5:
- Ontwerp SPI slave interface in VHDL.
- Bijsturen PID ontwerp, implementatie met behulp van IP core multiplier.
- Validatie van ontwerp door middel van testbenches.

Week 6:
- Bijsturen PID ontwerp, implementatie met behulp van IP core multiplier.
- Validatie van ontwerp door middel van testbenches.
- Ontwerp SPI slave interface aanpassen aan Mojo SPI.
- Testprint van quadcopter arm en stevigheidstest.

Week 7:
- Bijsturen ontwerp van quadcopter arm.
- Printen van vier quadcopterarmen.
- Ontwerp van PWM generator in VHDL.

Week 8:
- Tussentijdse presentatie van het project.
- Testen PWM generator op Mojo.
- Testen van SPI slave interface op Mojo.
- Ontwerp bijsturen waar nodig.

Week 9:
- PID regeling aanpassen naar multiplexing om resources te besparen op de FPGA.
- Valideren van het nieuwe PID ontwerp.
- Testen van PWM en SPI samen op Mojo bord.

Week 10:
- Ontwerpen van motormixingalgoritme in VHDL.
- Optimalisatie van PID controller.
- Implementatie van registers.

Week 11:
- Labobad Gent.

Week 12:
- Eindpresentatie en demo van het labproject.

Week 14:
- Indienen van het eindverslag.

Hoofdstuk 2

Vooronderzoek

In dit hoofdstuk zullen de verschillende resultaten van het vooronderzoek besproken worden. Ook de simulaties in het programma simulink worden kort toegelicht. Op het einde van dit hoofdstuk wordt een compleet systeemoverzicht weergegeven.

2.1 Quadcopters

2.1.1 Wat is een quadcopter

Een quadcopter, ook wel quadrocopter genoemd, is een hefschroefvliegtuig dat, zoals de naam al doet vermoeden, aangedreven wordt door vier motoren. In tegenstelling tot een conventionele helikopter maakt een quadcopter gebruik van symmetrisch hellende propellors die in groepen van twee dezelfde draairichting hebben. Op deze manier wordt de tolbeweging die ontstaat door de krachtwerking van de motoren tegengewerkt en gereduceerd tot nul. Dit zorgt ervoor dat de quadcopter tijdens de vlucht niet rondtolt. Bij een klassieke helikopter wordt het opheffen van deze kracht voorzien door de staartrotor.

Figuur 2.1: Quadcopter in actie

Een quadcopter is opgebouwd uit een frame dat vier armen bevat. Op het uiteinde van deze armen zijn de motoren (brushless DC) bevestigd. Om de motoren aan te sturen wordt gebruik gemaakt van een ‘engine speed controller ’(ESC). Deze zorgt voor de aansturing van de motoren. Een centrale eenheid regelt zowel de stabilisatie als de invoer van de piloot.

Een quadcopter is van nature onstabiel. Een regelsysteem dat stabilisatie voorziet tijdens de vlucht is nodig. Dit systeem zal ten minste een stabilisatie voorzien rond de roll-, pitch- en yaw-as. Optioneel kan ook een stabilisatie voor drift en hoogte voorzien worden. Deze stabilisatie wordt meestal voorzien door middel van een PID controller voor elke as.

Naast het voorzien van stabilisatie dient er ook gereageerd te worden op de invoer van de piloot. De centrale eenheid (autopilot) implementeert typisch deze functies. Naast het voorzien van deze basisfuncties kunnen nog verschillende opties toegevoegd worden zoals een return-to-home functie, een autoland functie...

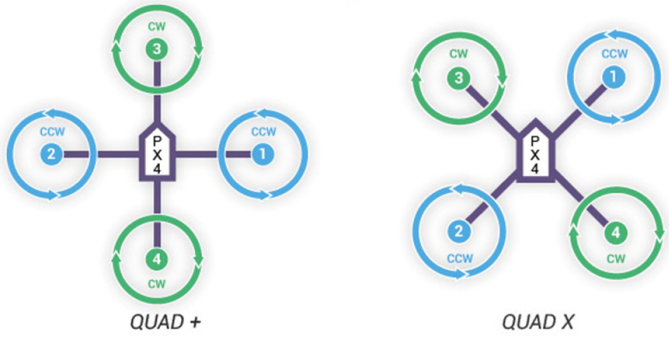
Als basis voor deze autopilot wordt een IMU unit voorzien. Deze beschikt aan de hand van een accelerometer-, magnetometer- en gyroscoop-informatie over de actuele attitude van de quadcopter. Op basis van deze gegevens kan een stabilisatie uitgevoerd worden.

Hiervoor zijn reeds verschillende systemen op de markt. Een populair systeem binnen de amateurpiloten is ArduPilot. Deze autopilot is volledig gebaseerd op Arduino en compatibel met zowat alle quadcopters en kleine vliegtuigen.

Net zoals ArduPilot zijn zowat alle autopilot systemen momenteel op de markt, gebaseerd op een microcontroller. Het doel van dit projectlab is het onderzoeken of de basisfuncties van een autopilot ook geïmplementeerd kunnen worden op een FPGA.

2.1.2 Configuraties

Een quadcopter kan opgebouwd worden op twee verschillende manieren. Afhankelijk van hoe de armen aan het centrale stuk geplaatst worden, verkrijgt men ofwel een '+'ofwel een 'x'configuratie. De configuratie wordt dus bepaald door de vliegrichting van de quadcopter ten opzichte van de propellers. In figuur 2.2 is een overzicht van beide configuraties weergegeven.



Figuur 2.2: Verschillende configuratiemogelijkheden van een quadcopter

De keuze van een bepaalde configuratie heeft een invloed op de manier van besturing, maar ook op het algoritme waarmee de verschillende motoren worden aangestuurd. Wanneer men gebruikt maakt van de 'x'configuratie, zijn er twee motoren nodig om een beweging om een bepaalde as uit te voeren.

Het spreekt voor zich dat het algoritme dat nodig is om de motoren aan te sturen bij een 'x'configuratie complexer is dan bij de keuze van een '+'configuratie. Er wordt dus gekozen voor de '+'configuratie binnen dit projectlab.

2.1.3 Besturing van een quadcopter

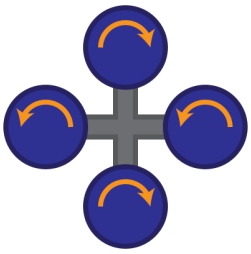
Om een quadcopter te besturen dienen de motoren op een correcte wijze aangestuurd te worden. Afhankelijk van hoe de motoren aangestuurd worden zal de quadcopter stijgen, dalen, gieren, of stampen. Daarnaast is het ook mogelijk een combinatie van de verschillende bewegingen samen uit te voeren.

Om een quadcopter te laten stijgen of dalen, dienen de vier motoren gelijktijdig versneld of vertraagd te worden.

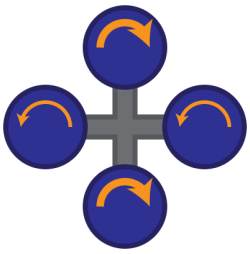
Om een quadcopter te laten bewegen om zijn verticale as moet de snelheid van twee motoren die éénzelfde draairichting hebben verhoogd worden. Hierdoor zal het koppel op de quadcopter toenemen waardoor een gierbeweging zal ontstaan.

Om een quadcopter vooruit, achteruit, link of rechts te laten bewegen in de '+ 'configuratie volstaat het om één motor sneller te laten bewegen dan de anderen. Hierdoor zal de quadcopter kantelen in een bepaalde richting, met een verplaatsing tot gevolg.

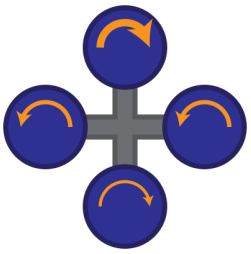
In figuren 2.3, 2.4 en 2.5 is een overzicht weergegeven van de besturing van een quadcopter in de '+ 'configuratie.



Figuur 2.3: Omhoog en omlaag



Figuur 2.4: Gieren



Figuur 2.5: Kantelen

2.2 PID regelaar

Een PID regelaar is een veel voorkomende regelaar binnen de procesregeling. De afkorting PID staat voor Proportioneel, Integrerend en Differentiërend. Dit zijn ook de acties waaruit de regelaar is opgebouwd.

2.2.1 Werkingsprincipe

De regeling gaat uit van het verschil tussen de gewenste waarde en de actuele gemeten waarde. Dit wordt omschreven als het foutsignaal. Het bijsturen van dit foutsignaal gebeurt in drie parallelle stappen.

P-actie: De proportionele actie zal het foutsignaal versterken met een factor $\mathrm{K_p}$.

I-actie: De integrerende actie zorgt voor een constante sommatie van het foutsignaal. Afhankelijk van hoe lang er een fout is tussen de gemeten en gewenste waarde, zal de integrerende actie meer of minder signaal uitsturen. De $\mathrm{K_i}$ term, ook wel nasteltijd genoemd, bepaald het effect van deze actie. Hoe kleiner deze waarde, hoe krachtiger de actie.

D-actie: De differentiërende actie zal reageren op de snelheid van verandering van het foutsignaal. Hoe hoger de factor $\mathrm{K_d}$ wordt ingesteld, hoe sneller de wenswaarde bereikt zal worden.

Wiskundig kan men een PID regelaar als volgt omschrijven:

$$u(t) = K_p \cdot (e(t) + \frac{\int e(t)dt}{T_i} + T_d \cdot \frac{de(t))}{dt})$$

Hierin is

$U(t)$: De uitgang van de regelaar
K_p : De proportionele actie van de regelaar
T_i : De integratietijd van de regelaar
T_d : De differentiatietijd van de regelaar
$E(t)$: Het errorsignaal

Het bepalen van de juiste constanten is een werk van trial-and-error. Dit zal dan ook experimenteel moeten vastgesteld worden op de quadcopter.

2.2.2 Digitalisatie van een PID

De formule die hierboven weergegeven is, kan als volgt vertaald worden naar een digitaal systeem.

$$U_{\mathrm{k}} = U_{\mathrm{k\text{-}1}} \cdot e_{\mathrm{k}} + b_0 \cdot + b_1 \cdot e_{(\mathrm{k\text{-}1})} + b_2 \cdot e_{(\mathrm{k\text{-}2})}$$

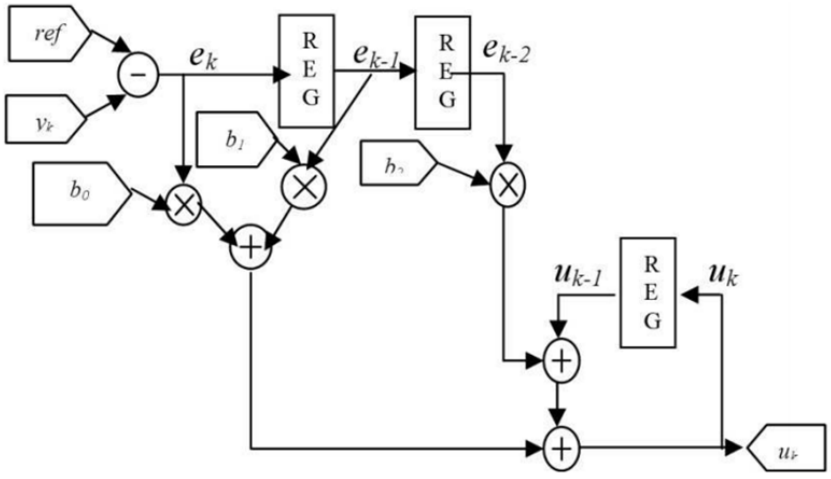
Hierbij is

$$b_0 = K_{\mathrm{p}}(1 + \tfrac{T_{\mathrm{d}}}{T})$$

$$b_1 = K_{\mathrm{i}}(-1 + \tfrac{T}{T_{\mathrm{i}}} - 2\tfrac{T_{\mathrm{d}}}{T})$$

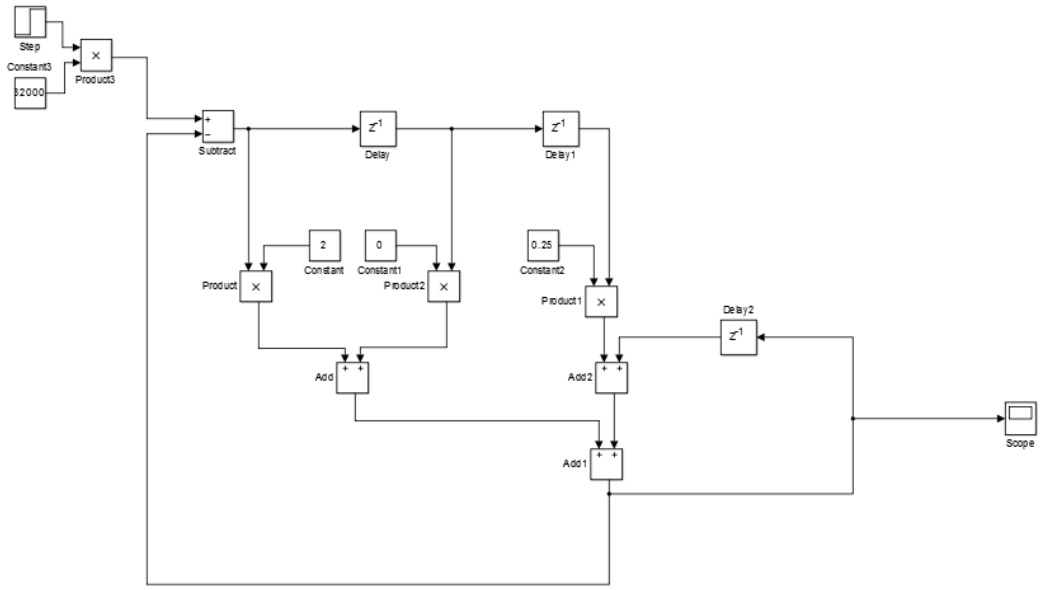
$$b_2 = K_{\mathrm{d}}(\tfrac{T_{\mathrm{d}}}{T})$$

Deze formule kan weergegeven worden in het onderstaande schema.



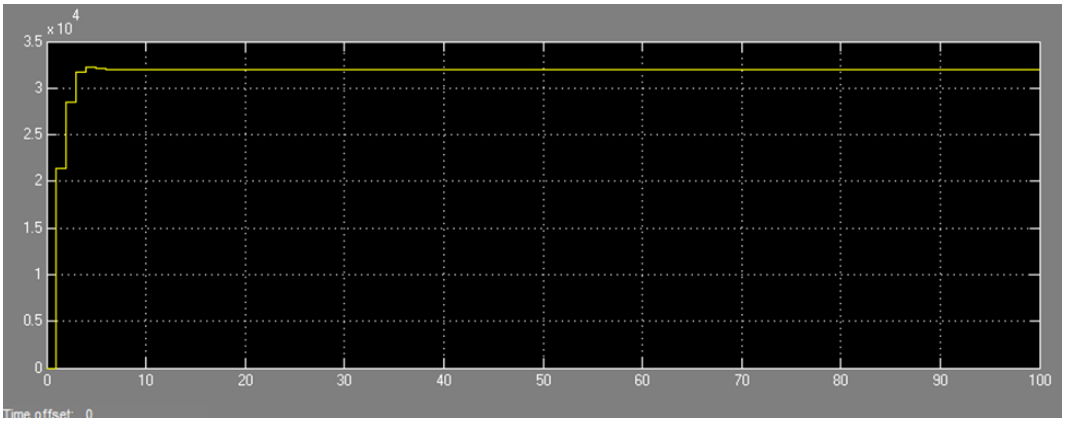
Figuur 2.6: Schematische voorstelling van een PID regelaar

De werking van de in figuur 2.6 weergegeven regelaar wordt gevalideerd via het programma Simulink. Op deze manier kan reeds voor de FPGA implementatie gekeken worden of de regelaar zal doen wat er verwacht wordt. Daarnaast vormt deze simulatie een goede basis om het FPGA design te starten. In figuur 2.7 is de Simulink versie van de PID regelaar weergegeven.



Figuur 2.7: Simulink voorstelling van een PID regelaar

Wanneer men de uitgang aan de ingang koppelt van bovenstaande PID wordt er aangenomen dat de nieuwe gemeten waarde gelijk is aan de uitgang van de PID. Dit levert het volgend resultaat op na simulatie.

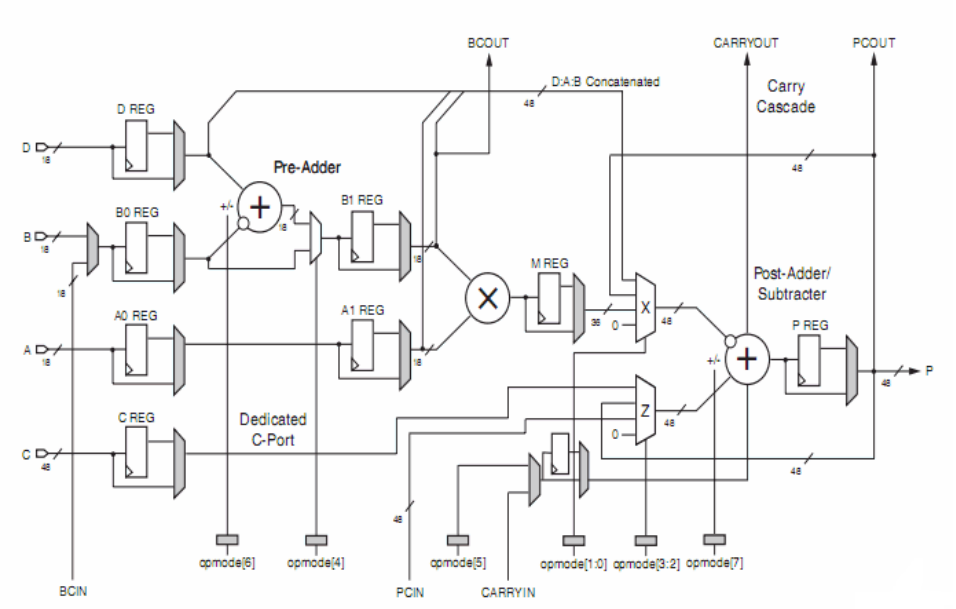


Figuur 2.8: Resultaat van simulatie PID in Simulink

In figuur 2.9 is duidelijk de werking van een PID waar te nemen. Het signaal klimt naar de gewenste waarde toe, maakt een kleine overshoot en stabiliseert vervolgens op het gewenste signaal. Dit is ook wat men verwacht van een PID. Men kan dus besluiten dat het schema in figuur 2.6 voldoet aan de eisen en geïmplementeerd kan worden in de FPGA.

2.3 Xilinx DSP48 slices

De Spartan 6 reeks bevat verschillende DSP48A slices. Deze MAC bouwstenen zijn door Xilinx speciaal ontworpen om DSP toepassingen te realiseren.



Figuur 2.9: Interne structuur van een DSP48A slice

Het doel van een DSP48 slice is het zo performant mogelijk uitvoeren van MAC operaties. Hiervoor zijn verschillende adders en een multiplier binnen het DSP48 slice aanwezig. Als invoer kan een signaal van maximaal 18 bits aangelegd worden. De uitgang is maximaal 48 bit. Hiervan is ook de naam DSP48 afgeleid.

De DSP48A slices kunnen binnen de Xilinx ISE ontwikkelomgeving op verschillende manieren worden geïnstantieerd. De eenvoudigste methode is via de IP Coregen. De gebruiker maakt stapsgewijs de instellingen op basis van een aantal selecties. Visueel wordt weergegeven welke instellingen gemaakt worden. Een andere manier om een DSP48 Slice te implementeren is door gebruik te maken van een zogenaamde Port Map in de code. Hierbij worden alle instellingen handmatig gemaakt en dient goed rekening gehouden te worden met de datasheet. Een derde manier om DSP48 slices te gebruiken is via de Xilinx System Generator. Dit is een pakket dat in samenwerking met Matlab en Simulink de mogelijkheid voorziet om visueel, door middel van blokken, een systeem samen te stellen en de nodige VHDL code te genereren.

De Spartan 6SLX9 beschikt over 12 DSP48A slices. Binnen het FPGA zullen deze gebruikt worden waar mogelijk om het gehele proces te versnellen.

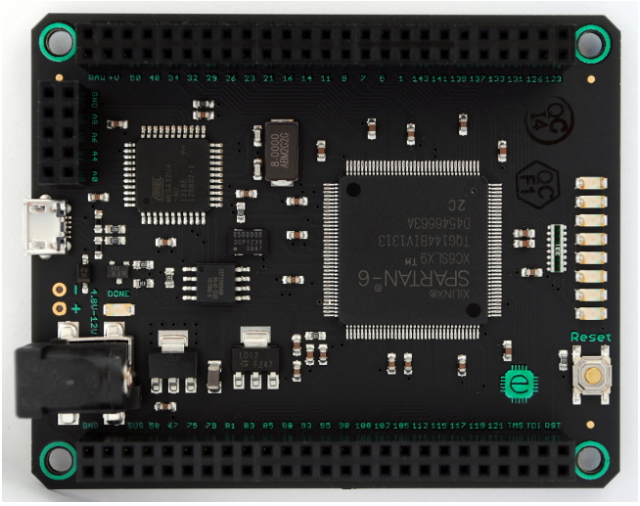
2.4 Mojo FPGA ontwikkelbord

Tijdens het projectlab zal gebruik gemaakt worden van het Mojo FPGA ontwikkelbord. Dit bord heeft een Spartan 6 XC6SLX9 FPGA aan boord waarbij de gebruiker beschikt over 84 digitale IO pinnen. Naast de digitale IO pinnen van het Mojo bord zijn er ook 8 analoge inputs voorzien. Deze bevinden zich op een externe ATmega controller die via een SPI interface communiceert met de FPGA.

Het programmeren van het Mojo bord gebeurt op eenvoudige wijze. Via het programma Mojo loader kan een `.bit` bestand naar de externe ATmega geladen. Telkens wanneer de FPGA opgestart wordt zal het laatst opgeslagen `.bit` bestand geladen worden op de FPGA. Op deze manier dient na stroomuitval niet steeds opnieuw het `.bit` bestand geladen te worden.

Er zijn reeds verschillende shields op de markt die het mogelijk maken om het Mojo board verder uit te breiden. Daarnaast is het project volledig open source waardoor er zelf ook makkelijk extra hardware ontwikkeld kan worden. Alle schema's en PCB layouts zijn via de website van de makers vrij verkrijgbaar.

In afbeelding 2.10 is het Mojo bord weergegeven. Naast de 84 digitale en 8 analoge IO pinnen beschikt het Mojo bord ook over 8 leds die verbonden zijn met de FPGA.



Figuur 2.10: Overzicht van het Mojo FPGA ontwikkelbord

2.5 Sensoren

Om een stabilisatie te voorzien van de quadcopter is het noodzakelijk een inzicht te krijgen van zijn actuele attitude. Hiervoor wordt gebruik gemaakt van een IMU sensor. IMU staat voor inertial measurement unit. Deze unit bevat een combinatie van accelerometers, magnetometers en gyroscopen om zo informatie te verwerven over de huidige attitude van de quadcopter in de ruimte. Om daarbij ook informatie te verschaffen over de hoogte, wordt gebruik gemaakt van de combinatie van een ultrasone sensor en een barometer.

2.5.1 Opmeten van de attitude

Principe

Een AHRS, attitude and heading reference system, voorziet de gebruiker van informatie over de stand van het vliegtuig. Om dit te berekenen wordt gebruik gemaakt van de data die afkomstig is van een IMU sensor. Zoals reeds hierboven beschreven werd, is een IMU sensor een combinatie van volgende drie sensoren.

Accelerometer: Een accelerometer meet de versnelling van de quadcopter, vaak uitgedrukt in g-krachten. Aan de hand van deze versnelling kan bepaald worden of de quadcopter in een bepaalde richting beweegt of in vrije val is.

Gyroscoop: Een gyroscoop meet de hoekverdraaiing op, uitgedrukt in radialen per seconde. Zo kan men de draairichting van de quadcopter bepalen. Deze hoekverdraaiing kan op zijn beurt gebruikt worden om de gemeten accelerometerwaarden te corrigeren. Indien de accelerometer namelijk niet in de correcte positie gehouden wordt, is het assenstelsel van de quadcopter verdraaid en zijn de gemeten x, y en z componenten niet correct ten opzichte van het assenstelsel in de reële wereld.

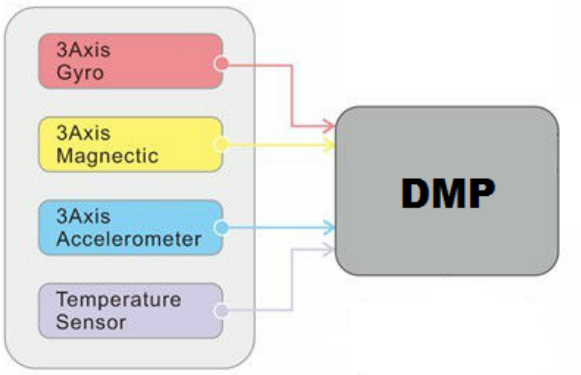
Magnetometer: Een magnetometer meet storingen in het aardmagnetisch veld op. Deze data wordt gebruikt om de performatie van de AHRS berekeningen op te voeren.

Aangezien men met de gyroscoop kan opmeten hoe groot de verdraaiing ten opzichte van het originele assenstelsel is, kunnen we het assenstelsel van de accelerometer corrigeren door matrixbewerkingen uit te voeren. Op die manier zijn de gemeten accelerometerwaarden toch te beschouwen vanuit het wereldse assenstelsel.

Uit deze data wordt vervolgens de attitude van de quadcopter berekend. Deze bestaat uit de heading, de pitch-rate en de yaw-rate.

Vele sensoren gebruiken naast de boven beschreven parameters ook de temperatuur om een kalibratie uit te voeren en de nauwkeurigheid van het systeem te verbeteren.

In figuur 2.11 is het principe van een AHRS op basis van IMU data weergegeven.



Figuur 2.11: Overzicht van een AHRS systeem

Sensoren

Vandaag de dag zijn er verschillende sensoren op de markt te verkrijgen. Onderzoek heeft uitgewezen dat binnen de UAV wereld de MPU reeks van InvenSense vaak gebruikt worden als IMU. Binnen deze reeks van sensoren komen twee sensoren in aanmerking.

MPU 6050: Deze IMU omvat heel wat mogelijkheden. Zo is er een ingebouwde processor, de DMP (Digital Motion Processor), die reeds heel wat berekeningen op zich kan nemen. Deze DMP is in staat simpele gestures te detecteren, stappen te tellen, filtering uit te voeren van de sensorwaarden en meer aangezien deze volledig naar wens kan ingesteld worden. De nauwkeurigheid van deze IMU is uitstekend, alsook de mogelijkheid tot het herconfigureren van de DMP. De MPU 6050 bevat een 3-assige accelerometer en een 3-assige gyroscoop. Via een I^2C is het mogelijk een externe magnetometer aan te sluiten. De data hiervan wordt opgenomen in de DMP bewerkingen.

MPU 9250: Deze IMU beschikt over exact dezelfde mogelijkheden als de MPU 6050, maar heeft reeds een 3-assige magnetometer aan boord. Er dient dus geen externe magnetometer aangesloten te worden.

Er werd gekozen om de MPU9250 sensor te gebruiken bij het ontwerp van de quadcopter. Door de interne filter binnen de DMP wordt een Kalman filter overbodig in het FPGA ontwerp.

2.6 Serial peripheral interface

SPI is een veelgebruikt protocol voor communicatie tussen twee apparaten. Een SPI-bus bestaat uit vier signalen.

MOSI: (Master Out Slave In) Dit signaal is de uitgang van de master en vormt de ingang van het slave apparaat. Via dit kanaal wordt informatie van de master naar de slave verstuurd.

MISO: (Master In Slave Out) Dit signaal is de uitgang van de slave en vormt de ingang van het master apparaat. Via dit kanaal wordt informatie van de slave naar de master verstuurd.

SCLK: (Serial Clock Line) Dit kanaal wordt gebruikt als klok. Het is het master apparaat dat de klok voorziet voor het slave apparaat. Data wordt enkel verstuurd en ontvangen als er een kloksignaal is.

SS: (Slave Select) Via een slave select kan de master kiezen uit verschillende slave apparaten waarmee hij wil communiceren. Op deze manier kan een communicatie opgezet worden met meerdere apparaten door middel van één master apparaat.

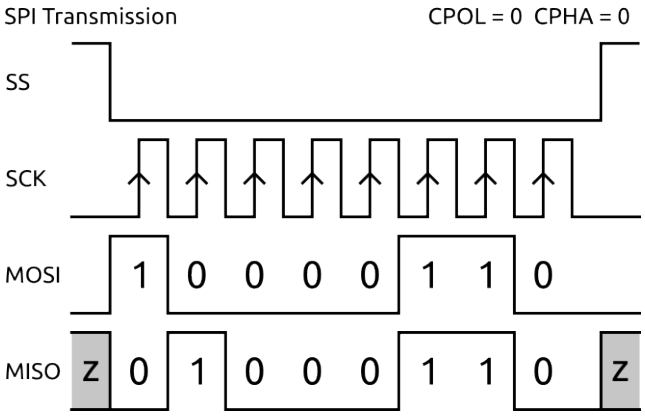
Zoals reeds eerder werd aangehaald, bestaat een SPI interface uit één master apparaat en verschillende slave apparaten. Hierbij kan de master communiceren met elke slave, maar een slave kan enkel communiceren met de master. Aan de hand van een uniek slave select signaal voor elke slave wordt de selectie gemaakt. De communicatie gebeurt full duplex.

De vereiste voor de klok is dat zijn frequentie lager moet zijn dan de maximale frequentie van de betrokken toestellen. Binnen SPI zijn er twee parameters waarbij rekening gehouden moet worden.

CPOL: De Clock polarity geeft aan wat de standaard waarde is van het kloksignaal wanneer de SPI-bus zich in de idle toestand bevindt.

CPHA: De Clock phase geeft aan op welke edge van het signaal gesampled zal worden. Een 0 geeft aan dat er op de rising edge gesampled wordt. Een 1 geeft aan dat er op de falling edge gesampled wordt.

Door het combineren van bovenstaande parameters kan men vier verschillende modi onderscheiden binnen de SPI interface. In figuur 2.12 is een overzicht weergegeven van een SPI communicatie tussen twee apparaten. Hierbij zijn de CPOL en CPHA beide ingesteld op 0.



Figuur 2.12: Overzicht van SPI communicatie tussen master en slave

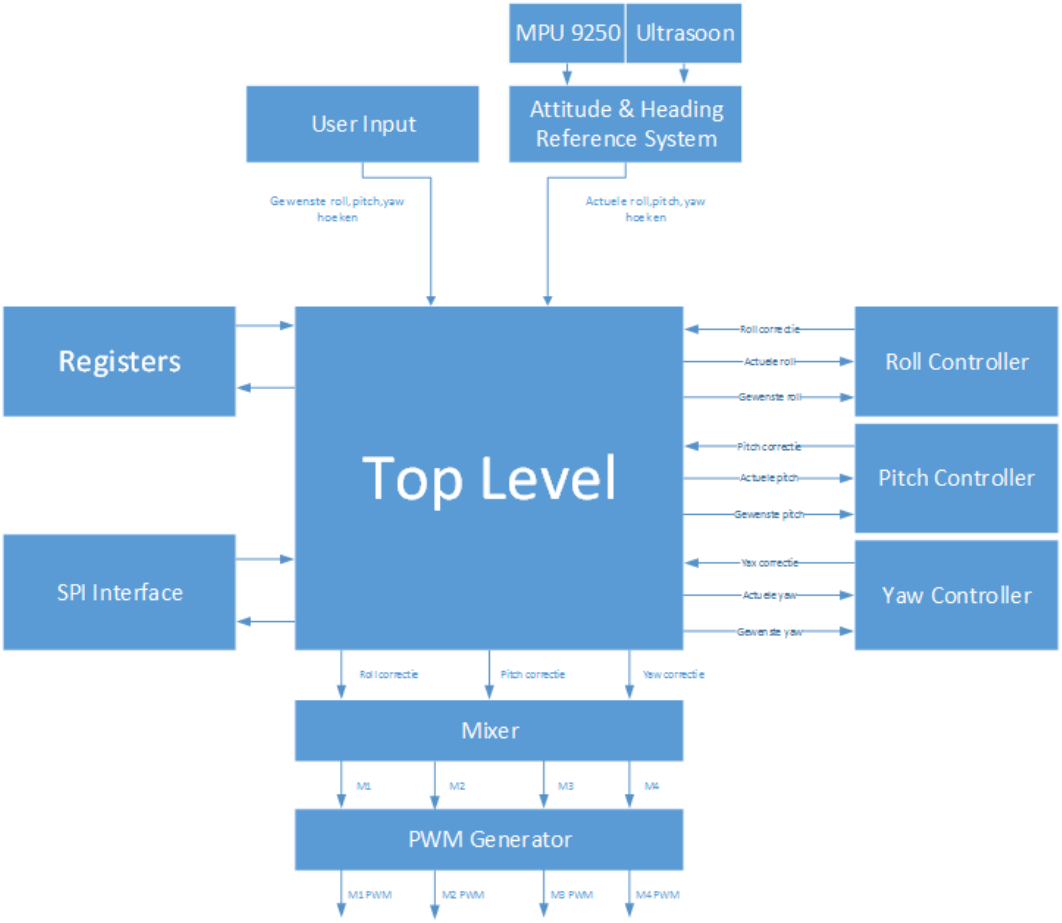
Op het moment dat de master een clock genereert, zal de communicatie aanvatten. Zowel de master als de slave zullen gelijktijdig data uitwisselen.

Wanneer de SS naar een lage toestand gebracht wordt, zal de slave geselecteerd worden. Vervolgens genereert de master het kloksignaal en start daarbij ook met versturen van data. Het slave apparaat zal ook starten met sturen van data op de wijzigende flanken van de klok. Wanneer de communciatie voltooid is, stopt de klok en wordt de SS weer naar hoge toestand gebracht.

Om data van de slave op te vragen, moet dus eerst een bericht verzonden worden van welke data men wil opvragen. Daarna moet men nog een bericht versturen om de data te ontvangen. Dit is nodig aangezien het zenden en ontvangen steeds gelijktijdig gebeurt.

2.7 Systeemoverzicht

De resultaten van dit vooronderzoek hebben geleid tot een systeem dat in staat is om de attitude van de quadcopter te controleren. Dit systeem dient in de volgende fase van het project geïmplementeerd te worden in een FPGA. In figuur 2.13 is een overzicht van dit systeem weergegeven.



Figuur 2.13: Schematische voorstelling van de quadcopter autopilot

In de figuur kan men volgende blokken onderscheiden.

User input: Deze blok is verantwoordelijk voor het afhandelen van de gebruikersinvoer. Deze is afkomstig van de RF antenne. Een PWM signaal per kanaal dient gelezen te worden. Dit PWM signaal wordt door het toplevel doorgegeven aan de PID controllers om zo de gewenste actie te ondernemen.

Registers: In de registers worden alle verschillende instellingen bewaard. Naast deze instellingen worden ook waarden aangeboden aan de gebruiker. Via een SPI interface kan het register gelezen en beschreven worden.

SPI interface: De SPI slave interface maakt het mogelijk om zowel te communiceren met een externe microcontroller als de MPU9250 die dienst doet als IMU uit te lezen.

Mixer: De mixer is verantwoordelijk voor de verwerking van alle data die afkomstig is van de PID controllers voor rol, pitch en yaw. Aan de hand van deze informatie zal de mixer de geschikte PWM waarden berekenen voor de motoren. Hierbij zal rekening gehouden worden met de maxima en minima die ingesteld zijn in de registers.

PWM generator: De BLDC motoren worden aangestuurd door middel van een PWM signaal. De PWM generator zal voor elke motor een geschikt PWM signaal genereren op basis van de uitvoer van de mixer.

Roll controller: De roll-controller stabiliseert het rollkanaal van de quadcopter. Naast de gebruikersinvoer vormt ook de roll-info van het AHRS een input voor deze controller.

Pitch controller: De pitch-controller stabiliseert het pitchkanaal van de quadcopter. Naast de gebruikersinvoer vormt ook de pitch-info van het AHRS een input voor deze controller.

Yaw controller: De yaw-controller stabiliseert het yawkanaal van de quadcopter. Naast de gebruikersinvoer vormt ook de yaw-info van het AHRS een input voor deze controller.

Attitude & heading reference system: Het AHRS geeft meer informatie over de attitude en heading van de quadcopter. Hiervoor gebruikt dit systeem verschillende sensoren. Dit AHRS systeem zal uitgevoerd worden op een aparte microcontroller en via een SPI interface waarden doorgeven aan het autopilot systeem.

In het volgende hoofdstuk zullen alle stappen besproken worden die nodig zijn om bovenstaande blokken te implementeren.

Hoofdstuk 3

Implementatie

In dit hoofdstuk zal de implementatie van het volledige systeem besproken worden. Er zal vooral de nadruk gelegd worden op de moeilijkheden en problemen die zich voordeden tijdens het ontwikkelproces.

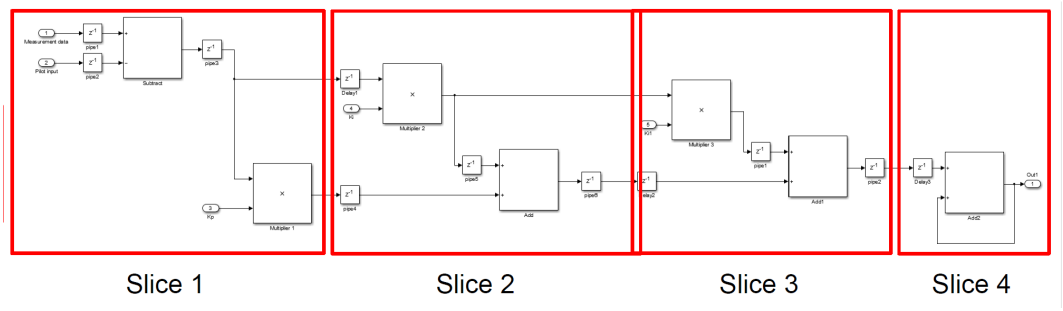
3.1 PID regelaar

De PID regelaar vormt het belangrijkste onderdeel van de quadcoptersturing. Deze component zal per as geïmplementeerd moeten worden. Daarnaast kan de PID in een verder stadia ook gebruikt worden om de hoogte te stabiliseren. Er dient dus een flexibele component ontworpen te worden die meerdere malen ingezet kan worden. Hierbij is het belangrijk dat de waarden voor K_p, K_i en K_d per controller ingesteld kunnen worden. Daarnaast is het ook mooi meegenomen om deze real-time aanpasbaar te maken.

De PID controller werd ontwikkeld in drie iteraties. Vanwege de hardware beperkingen van de FPGA was het nodig om een volledig re-design uit te voeren van de component. Deze stappen zullen hieronder verder toegelicht worden.

3.1.1 PID controller door middel van DSP48A slices

Vanuit theoretisch oogpunt zijn DSP48 slices ideaal voor het implementeren van een PID regelaar. De bewerkingen die uitgevoerd dienen te worden beperken zich tot optellen en vermenigvuldigen (MAC). Op deze manier wordt een hoge snelheid van de component mogelijk. Per PID regelaar zijn vier DSP48 slices nodig. In figuur 3.1 is de onderverdeling weergegeven van DSP48 slices binnen de PID regelaar. Hierbij zijn de functies van slice 2 en 3 exact hetzelfde. Bij de laatste DSP48 slice is een terugkoppeling voorzien. Om ervoor te zorgen dat het resultaat nooit overflowt, dient er een downscaling te gebeuren bij de terugkoppeling. Deze wordt in 'gewone'logica uitgevoerd en zal het proces vertragen. Door het introduceren van de nodige 'pipes'wordt de maximale snelheid van implementatie bereikt.



Figuur 3.1: Overzicht van een PID controller op basis van DSP48 slices

Bovenstaande code werd in een eerste fase geïmplementeerd door middel van de Xilinx System Generator. Wanneer de VHDL code gegenereerd diende te worden, doken er steeds fouten op. Opzoekwerk heeft uitgewezen dat de System Generator niet compatibel is met Windows 8. Een Windows 7 apparaat met alle nodige software en licenties was helaas niet beschikbaar. Er werd overgegaan naar het implementeren van de PID in Xilinx ISE. Hierbij werd gebruik gemaakt van de IP Coregen voor het genereren van de juiste instellingen binnen de DSP48 slices.

Door eerste synthese resultaten bleek dat het niet mogelijk was om drie of vier PID's te implementeren op de Spartan 6 FPGA wegens gebrek aan resources. Een mogelijke oplossing was het multiplexen van de PID regeling. Op die manier is er slechts één PID regelaar nodig die afwisselend de berekeningen uitvoert voor een bepaald kanaal. De implementatie hiervan bleek echter te complex om te realiseren omdat de PID steeds de vorige resultaten nodig heeft om het volgende te berekenen. Deze data zou dus moeten opgeslagen worden. Daarnaast was het ook niet steeds mogelijk om deze waarden terug op de juiste plaats in het DSP48 slice te laden vanwege de beperkingen van ingangssignalen.

Er dient dus een andere oplossing gevonden te worden om de PID in de Spartan 6 FPGA te implementeren.

3.1.2 PID controller door middel van IPCore multiplier

Een tweede aanpak was om de PID te implementeren aan de hand van 'gewone'logica. Om de vermenigvuldigingen uit te voeren wordt gebruik gemaakt van de IP Core multiplier. Er zijn dus drie van deze multipliers nodig om de volledige component te realiseren.

Bij het aanmaken van deze multipliers zijn verschillende keuzes mogelijk. Er kan gekozen worden tussen een implementatie die gebruik maakt van ExtremeDSP (DSP48A slices) of een implementatie die gebruik maakt van LUT's en FF. Afhankelijk van de gekozen optie zullen de timings verschillen. Er is ook een mogelijk om te

optimaliseren naar area (minimaal gebruik aan resources) of naar snelheid. Er werd gekozen om te minimaliseren naar area. Hierbij maakte de multiplier enkel gebruik van LUT's en FF.

Via Xilinx Vivado werd een eerste versie van deze PID geïmplementeerd en gesimuleerd. De keuze voor Vivado werd gemaakt omwille van problemen met de ISE simulator op Windows 8. Daarnaast was het ook niet mogelijk om CoreIP componenten te simuleren in Modelsim vanwege een probleem met de bibliotheken. Xilinx biedt deze niet meer aan, omdat zij een eigen simulator ontwikkeld hebben.

De simulaties leverden dezelfde resultaten op als bekomen in Simulink. Er kon dus besloten worden dat de implementatie van de PID gelijkaardig is als de versie in Simulink. Bij het verder onderzoeken van de synthese resultaten bleek dat er op het eerste zicht voldoende resources beschikbaar waren om vier PID controllers te implementeren. Omdat er niet enkel een PID op de FPGA geïmplementeerd moet worden, maar ook nog een reeks andere componenten zou het toch kunnen dat men niet voldoende resources ter beschikking heeft.

Er dient dus een andere oplossing gevonden te worden om de PID in de Spartan 6 FPGA te implementeren met een minimaal gebruik aan resources.

3.1.3 PID controller door middel van multiplexed IPCore multiplier

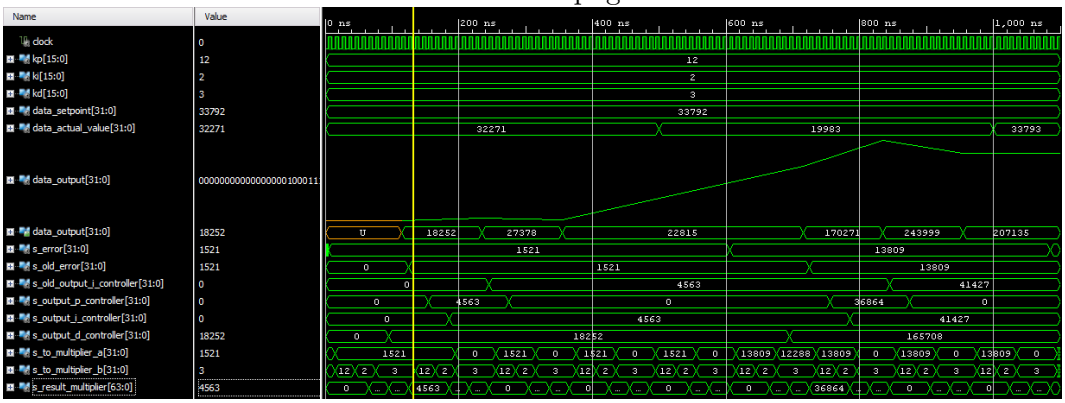
Een oplossing om de resources binnen een PID controller aanzienlijk te beperken is door in plaats van drie multipliers slechts één multiplier te gebruiken. Hierbij zal telkens de P, I en D actie uitgevoerd worden. De snelheid van het component zal hierdoor dalen met ongeveer een facor drie, maar dit zal weinig invloed hebben op de werking van de autopilot. Aangezien de sensordata binnenkomt met een snelheid van enkele kHz en de PID component geklokt kan worden aan een snelheid van enkele MHz zal hier geen effect van opgemerkt kunnen worden.

De implementatie gebeurt door middel van een state machine. In een eerste state zullen de K_p, K_i en K_d waarden ingelezen worden. Op deze manier zijn ze per berekening real-time aanpasbaar. In de volgende states zullen de P, I en D acties uitgevoerd worden. Dit houdt in dat de data die aan de multiplier aangelegd wordt, wijzigt per state. In de laatste state wordt een vlag kort hoog geplaatst om aan te geven dat de berekening klaar is. Er is dus maar één multiplier nodig die gemultiplext wordt.

Door het beperken van het aantal multipliers kon de snelheid van de multiplier geoptimaliseerd worden door toch gebruik te maken van DSP48 slices binnen deze multiplier. Deze wijziging werd dan ook doorgevoerd in de component.

Simulaties van de bovenstaande PID leverden exact hetzelfde resultaat op als de PID beschreven in paragraaf 3.1.2. Een overzicht van deze testbench is weergegeven in figuur 3.2.

testbench.png



Figuur 3.2: Simulatieresultaat van PID regelaar met gemultiplext multiplier

Wanneer men figuur 3.2 nader bekijkt, kan men opmerken dat wanneer de wenswaarde bereikt is, en bijgevolg het errorsignaal 0 is, de uitvoer niet terugvalt naar 0. De reden hiervoor is eenvoudig te verklaren. Aangezien de testbench werkt door middel van het aanleggen van bepaalde signalen voor gemeten waarde op bepaalde tijdstippen, heeft men hier niet te maken met een echt systeem. De I actie van de PID wordt berekend door middel van volgende formule.

$$I_{\text{regelaar}} = (Error \cdot K_{\text{i}} + Old_{K\text{i}})$$

Er wordt dus steeds een berekening gemaakt op basis van de oude waarde van K_{i} waardoor de waarde nooit 0 wordt in de simulatie. In een echt systeem wordt dit probleem onmiddellijk opgelost omdat de regelaar zal blijven bijregelen. De sensor zal opmerken dat de waarde de wens overstijgt en een tegenactie ondernemen.

3.1.4 Samenvattig PID controller

In listing 3.1 is de componentdefinitie van de PID controller weergegeven. Als invoer zijn zowel de sensordata als de gewenste waarde nodig. Daarnaast moeten ook de coëfficiënten voor K_{p}, K_{i} en K_{d} aangelegd worden. Deze coëfficiënten zijn real-time aanpasbaar.

Door de PID regelaar in verschillende stappen te ontwerpen konden steeds resources bespaard worden. Dit ging wel steeds ten koste van de snelheid van de component. Hoge snelheid bij deze component is niet zo belangrijk vanwege de relatief trage snelheid weermee de invoerdata kan aangeleverd worden.

Listing 3.1: Overzicht van de PID component

```
ENTITY PID_CONTROLLER IS
   GENERIC(
      data_width           : INTEGER := 32;
      internal_data_width  : INTEGER := 16
    );

   PORT(
       clock   :  IN  STD_LOGIC;
       reset   :  IN  STD_LOGIC;

       kp      :  IN STD_LOGIC_VECTOR(internal_data_width-1 DOWNTO 0);
       ki      :  IN STD_LOGIC_VECTOR(internal_data_width-1 DOWNTO 0);
       kd      :  IN STD_LOGIC_VECTOR(internal_data_width-1 DOWNTO 0);

       data_setpoint     :  IN STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
       data_actual_value :  IN STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
       data_output       :  OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0)
   );
END PID_CONTROLLER;
```

3.2 PWM generator

Per motor is er telkens één PWM generator nodig bij de quadcopter. De PWM generator zal een PWM signaal genereren op basis van een invoerwaarde die afkomstig is van de motormixing component. Om flexibiliteit te garanderen is de component volledig generiek gemaakt. Op deze manier kan zowel de resolutie als PWM frequentie gewijzigd worden. Een overzicht van de PWM component is weergegeven in listing 3.2.

Listing 3.2: Overzicht van de PWM component

```
ENTITY PWM_GENERATOR IS
  GENERIC (
         base_clock     : INTEGER := 100000000;
         pwm_frequency  : INTEGER := 200000;
         resolution     : INTEGER := 8);
  PORT (
         clock       : IN STD_LOGIC;
         reset       : IN STD_LOGIC;
         enable      : IN STD_LOGIC;
         duty_cycle: IN STD_LOGIC_VECTOR(resolution - 1 DOWNTO 0);
         pwm_out     : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
         pwm_inverse_out : OUT STD_LOGIC_VECTOR(0 DOWNTO 0));
END PWM_GENERATOR;
```

3.3 Motor Mixing

Het motormixing gedeelte van de quadcopter is verantwoordelijk voor het genereren van de signalen die naar elke motor gestuurd moeten worden. Bij de opbouw van de quadcopter werd gekozen voor de '+'configuraties. Het mixen van de motors gebeurt dan via het volgend algoritme.

$$Motor_{\text{front}} = baseSnelheid + pitchOutput - yawOutput$$
$$Motor_{\text{left}} = baseSnelheid + rollOutput + yawOutput$$
$$Motor_{\text{back}} = baseSnelheid - pitchOutput - yawOutput$$
$$Motor_{\text{right}} = baseSnelheid - rollOutput + yawOutput$$

Hierbij is de baseSnelheid gelijk aan de idle snelheid van de motor vermeerderd met de throttle.

Net als bij de PWM generator is ook deze component volledig generiek opgebouwd zodat deze makkelijk kan aangepast worden. Een overzicht van deze component is weergegeven in listing 3.3.

Listing 3.3: Overzicht van de motormixing component

```
entity MOTOR_MIXING is
   GENERIC(
      data_width_in  : INTEGER := 8;
      output_data_width : INTEGER := 8
   );
   PORT(
      clock: in STD_LOGIC;
      reset: in STD_LOGIC;
      roll_correction : IN STD_LOGIC_VECTOR(data_width_in-1 DOWNTO 0);
      pitch_corrention : IN STD_LOGIC_VECTOR(data_width_in-1 DOWNTO 0);
      yaw_correction : IN STD_LOGIC_VECTOR(data_width_in-1 DOWNTO 0);
      height_correction : IN STD_LOGIC_VECTOR(data_width_in-1 DOWNTO 0);
      throttle_value : in STD_LOGIC_VECTOR (data_width_in-1 DOWNTO 0);
      motor_front_out : OUT STD_LOGIC_VECTOR(output_data_width DOWNTO 0);
      motor_left_out : OUT STD_LOGIC_VECTOR(output_data_width DOWNTO 0);
      motor_back_out : OUT STD_LOGIC_VECTOR(output_data_width DOWNTO 0);
      motor_right_out : OUT STD_LOGIC_VECTOR(output_data_width DOWNTO 0);
      motor_max : IN (STD_LOGIC_VECTOR(output_data_width-1 DOWNTO 0))
    );
end MOTOR_MIXING;
```

Er zal ook rekening gehouden worden met de maxima waarde van het systeem. Wanneer een bepaalde waarde het maximum overschrijdt zal deze waarde automatisch worden teruggebracht naar de maximum waarde. De corrigerende actie zal daarom wat minder zijn dan gevraagd, maar de quadcopter blijft op deze manier wel binnen zijn limieten.

3.4 SPI slave interface

Zoals reeds eerder vermeld werd, voorziet de SPI interface de mogelijkheid om de autopilot te laten communiceren met de buitenwereld. In het vooronderzoek werd reeds de werking van een SPI slave interface aangehaald. Deze is in VHDL geïmplementeerd voor CPOL en CPHA waarbij beiden gelijk zijn aan 0. Als databreedte werd gekozen voor 8 bits, ofwel 1 byte.

In listing 3.4 is een overzicht weergegeven van het SPI slave component. Er kan data ingevoerd worden die verstuurd moet worden. Daarnaast wordt er ook kort een vlag hoog geplaatst wanneer de data ontvangen is.

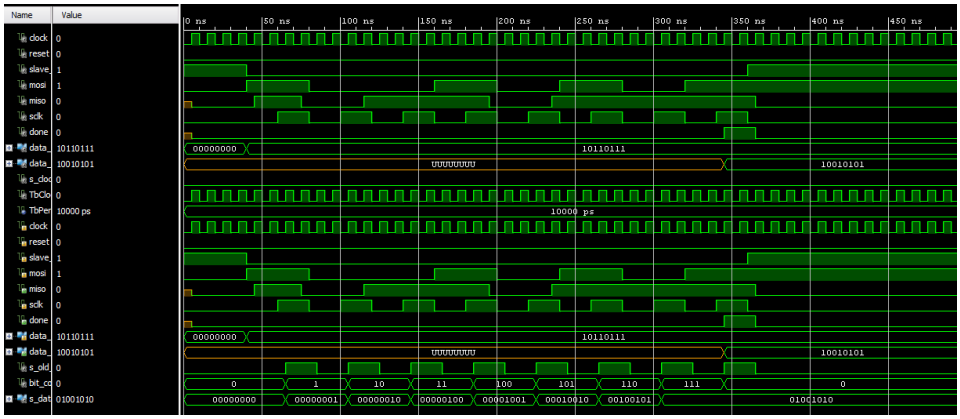
Listing 3.4: Overzicht van de SPI slave component

```
ENTITY spi_slave IS
  PORT(
        clock : in std_logic;
        reset : in std_logic;

        slave_select : in std_logic;
        mosi : in std_logic;
        miso : out std_logic;
        sclk : in std_logic;

        done : out std_logic;
        data_in : in std_logic_vector(7 downto 0);
        data_out : out std_logic_vector(7 downto 0)
    );
END spi_slave;
```

Om de werking van deze component te valideren, is er een testbench voor deze component ontworpen. Deze wordt geanalyseerd alvorens de code te testen op het Mojo board. Het resultaat van deze testbench is weergegeven in figuur 3.3.



Figuur 3.3: Simulatieresultaat van SPI slave interface

3.5 Top level

Het toplevel verbindt alle systeemcomponenten met elkaar. Daarnaast is ook op dit niveau de register geïmplementeerd aan de hand van de IP Coregen. Er wordt gebruik gemaakt van een block memory met 255 registers die elk een grootte hebben van 1 byte. Via de SPI interface kunnen deze registers zowel gelezen als ingesteld worden.

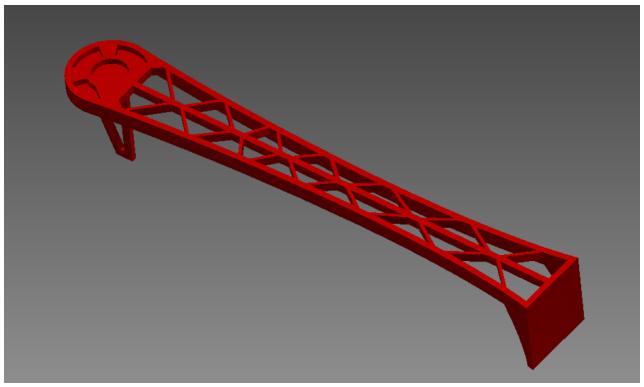
Het toplevel zal ook de nodige signalen naar de buitenwereld brengen. Zo worden de signalen naar de motoren, SPI-apparaten en sensoren naar buiten gebracht.

Via de IP Core clocking wizard worden de verschillende klokken aangemaakt waarmee de componenten werken.

3.6 Quadcopter Frame

Niet alleen de hardware componenten werden ontwikkeld tijdens dit projectlab, ook de quadcopter zelf werd ontworpen. Hierbij wordt gebruik gemaakt van een 3D printer die de verschillende onderdelen zal vervaardigen. In het computerprogramma Inventor worden alle onderdelen getekend en klaargemaakt om te kunnen worden geprint.

Een quadcopter is opgebouwd uit vier armen die gepositioneerd zijn in een 'x'. Zo'n arm is weergegeven in figuur 3.4.



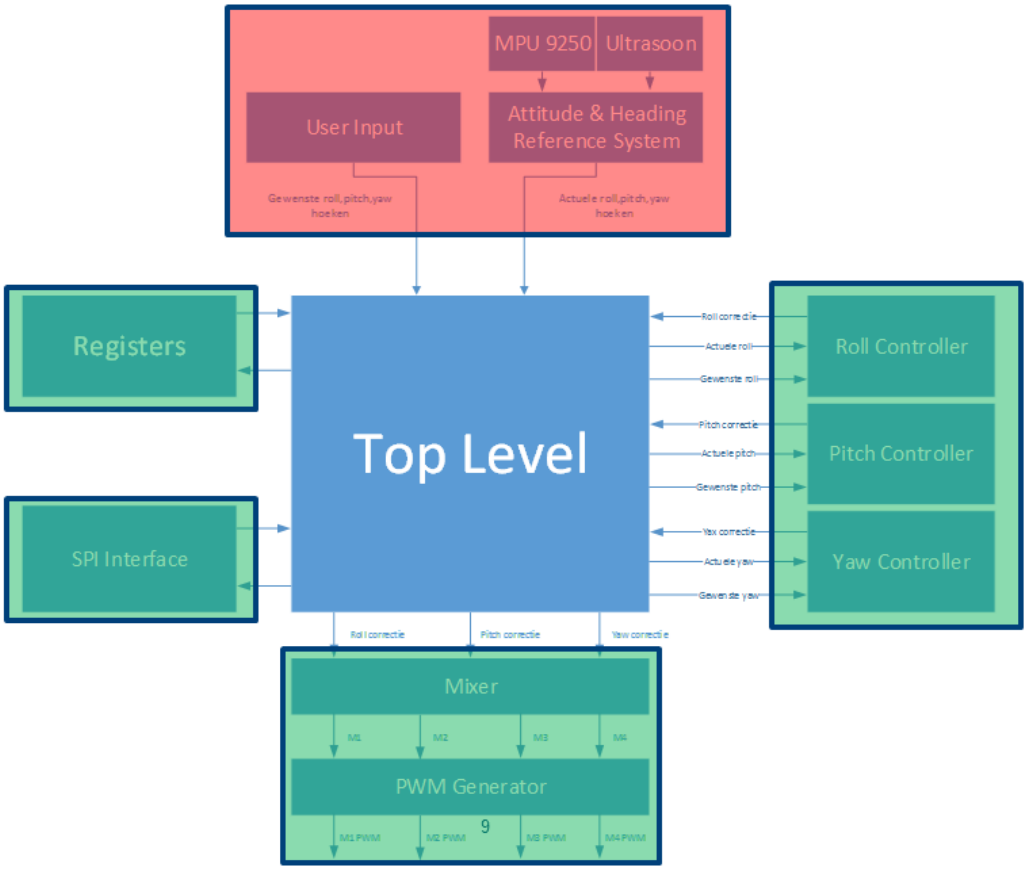
Figuur 3.4: Weergave van een quadcopter arm

De frameonderdelen werden vervaardigd door middel van een Ultimaker 3D printer. Indien er tijdens de vlucht een bepaald onderdeel beschadigd raakt, kan dit makkelijk opnieuw geprint worden en is quadcopter snel weer operatief. Het printen van een arm neemt ongeveer 4 uur in beslag.

Hoofdstuk 4

Besluit

Het project heeft een zeer grote omvang en kon tijdens de duurtijd van het projectlab niet volledig voltooid worden. Een overzicht van welke items voltooid werden en welke niet, is weergegeven in figuur 4.1.



Figuur 4.1: Overzicht van de projectstatus

De PID controller is volledig geïmplementeerd en geoptimaliseerd naar resources zodat deze past in de Spartan 6 FPGA. Hierbij werden verschillende iteraties doorlopen die telkens een andere methodiek gebruikten. Aan de hand van Simulink en testbenches werd de werking van de PID controller gevalideerd. De PID controller kon omwille van het ontbreken van sensordata nog niet getest worden op de FPGA zelf. Wel

werd al een synthese uitgevoerd om na te gaan of de verschillende PID controllers pasten in het FPGA ontwerp.

De SPI slave interface werd ontwikkeld aan de hand van dezelfde parameters als het Mojo board gebruikt om te communiceren met de analoge inputs. Op deze manier kan de SPI interface veelzijdig ingezet worden. Via een testbench werd de werking gevalideerd alvorens de interface uit te testen op een FPGA. Wanneer de testbench het gewenste resultaat gaf, werd een test uitgevoerd op de FPGA zelf. Ook hier was de werking perfect en kwamen geen problemen aan het licht.

De PWM generator en motormixer zijn volledig geïmplementeerd in VHDL. De werking van de PWM generator is reeds getest op een FPGA door het aansturen van de vier verschillende motoren. Hierbij werd ook gebruik gemaakt van de SPI interface om de gewenste PWM instelling te maken. Alles werkt naar behoren. Er dient wel rekening gehouden te worden dat bij het aanleggen van het systeem een vaste PWM waarde aan de motoren wordt aangelegd om deze te activeren. Na ongeveer 1 á 2 seconden (beepgeluid bij de motor) zijn de motoren actief en kunnen ze gebruikt worden.

Het quadcopterframe is vollledig geprint, maar dient nog geassambleerd te worden. Door de relatief lange printtijd was er onvoldoende tijd om deze assemblage uit te voeren gedurende het projectlab. De laatste prints waren helaas pas de dag zelf klaar.

Het globale autopilot systeem kon niet getest worden omdat nog niet alle componenten voltooid zijn. De delen die wel getest konden worden, zijn uitvoerig getest en gaven ook het verwachte resultaat.

In het verdere verloop van het project dienen volgende taken nog uitgevoerd te worden.

- Assamblage van het frame.
- Alle systeemcomponenten met elkaar verbinden.
- Implementeren van AHRS door middel van SPI slave connectie.
- Alle data tussen de verschillende componenten in het juiste formaat omzetten.
- Bepalen van de K_p, K_i en K_d parameters.
- Uitbreiden van de registers.
- Globale systeemtest.

Bijlagen

Bijlage A

Bijlage 1 - Projectplanning

% Work Complete

0%

Mon 23/02/15 - Mon 18/05/15

UPCOMING TASKS

TASKS QUADCOPTER PROJECT

Overview of the upcomming task

Name	Start	Finish	Work	Executor	Notes
Project management_2	Mon 23/02/15	Mon 18/05/15	338 hrs		
Theoretische voorstudie	Tue 24/02/15	Mon 2/03/15	28 hrs		
Opzoekwerk Quadcopters	Tue 24/02/15	Mon 2/03/15	5 hrs	Gert-Jan	In dit deel van het vooronderzoek wordt een theoretische voorstudie gemaakt over quadcopter. Er wordt gekeken naar hoe deze bestuurd worden, welke technieken er bestaan en hoe men een goede stabiliteit kan bekomen....
Studie PID implementatie	Tue 24/02/15	Mon 2/03/15	5 hrs	Gert-Jan	In deze subtaak wordt een theoretische studie verricht naar hoe een PID regelaar kan geïmplementeerd worden op een FPGA. Hiernaast zal ook de algemene werking van een PID regalaar bestudeerd worden....
Kalman filter	Tue 24/02/15	Mon 2/03/15	4 hrs	Gert-Jan	In deze subtaak zal de Kalman filter van naderbij bekeken worden. Er zal onderzocht worden of het mogelijk is deze ook in de FPGA te integreren. Er wordt zowel gekeken naar footprint als haalbaarheid binnen het gegeven tijdsdomein. Ook de theorie achte...
Studie naar BLDC sturing	Tue 24/02/15	Mon 2/03/15	3 hrs	Nick	Brushless DC motoren zijn betrouwbaar en licht, waardoor dit de optimale keuze is voor multicopters. De sturing hiervan is echter complexer dan brushed DC motoren. Hiervoor dient dus een stuurkring ontwikkeld te worden die de 4 motoren kan bedienen.
Optical Flow	Tue 24/02/15	Mon 2/03/15	5 hrs	Nick	Er zijn reeds verschillende optical flow sensoren en kits beschikbaar. Het is de bedoeling een beste mogelijkheid te zoeken en deze te implementeren.
Mavlink protocol	Tue 24/02/15	Mon 2/03/15	2 hrs	Nick	MAVlink is een communicatieprotocol dat geen fysische transportlaag gedefinieerd heeft. Het bevat enkel de data die moet overgedragen worden. Dit protocol moet dus geïmplementeerd worden op een zelf te selecteren fysische laag.

Antenne design	Tue 24/02/15	Mon 2/03/15	4 hrs	Nick	Voor de gekozen fysische laag voor het MAVlink protocol dient een correcte antenne ontworpen te worden. Hierbij spelen interefentie en antennelengte een grote rol.
FPGA Implementatie	Mon 23/02/15	Mon 4/05/15	103 hrs	Gert-Jan	
Register implementatie	Mon 2/03/15	Mon 27/04/15	33 hrs	Gert-Jan	
SPI Interface - Externe communicatie	Mon 2/03/15	Mon 9/03/15	5 hrs	Gert-Jan	Om te communiceren met een externe microcontroller zal het SPI protocol gebruikt worden. Hiervoor is het noodzakelijk om een SPI controller te implementeren op de FPGA die het mogelijk maakt bepaalde registers in te stellen en uit te lezen. Ook de beri...
Register block ram	Mon 2/03/15	Mon 9/03/15	3 hrs	Gert-Jan	Het register block ram bevat alle instellingen van de FPGA autopilot. Deze registers kunnen gelezen en beschreven worden door verschillende componenten binnen de FPGA. In deze subtaak worden het register opgebouwd en voorzien van de nodige documentatie.
PID Regeling	Tue 10/03/15	Mon 27/04/15	25 hrs	Gert-Jan	
PID Ontwerp	Tue 10/03/15	Mon 23/03/15	15 hrs	Gert-Jan	In deze subtaak wordt een PID regelaar ontwerpen op een FPGA. Deze regelaar ontvangt inputs van de MPU9250, Optical Flow en andere aangesloten sensoren. Eerst zal er een theoretisch model in Matlab gemaakt worden, waarna het geïmplementeerd wordt in VH...
P,I,D parameterbepaling	Mon 13/04/15	Mon 27/04/15	10 hrs	Gert-Jan	Het 'tunen' van de P, I, D parameters wordt aan het einde van het project vervolledigd. Hiervoor is het noodzakelijk dat alle systemen 'up and running' zijn. In dit process zullen de P,I en D parameters experimenteel bijgeregeld worden.
Uitlezen Sensoren	Mon 23/02/15	Mon 30/03/15	30 hrs	Gert-Jan	
I2C interface	Mon 9/03/15	Mon 16/03/15	10 hrs	Gert-Jan	De sensoren die aangsloten zijn op de FPGA communiceren met de FPGA door middel van een I2C inteface. Alvorens communicatie mogelijk is, is het alvast nodig om een I2C driver te implementeren in de FPGA. Deze driver zal dienst doen als I2C (single) mas...
MPU9250	Mon 16/03/15	Mon 23/03/15	8 hrs	Gert-Jan	De MPU6050 is de sensor bevat een 3-assige gyroscoop, accelerometer en magnetometer. Hij is de belangrijkste sensor van de autopilot. Deze sensor zal uitgelezen worden door middel van de I2C interface. Hiervoor dienen verschillende instellingen gemaak...
Optical Flow	Mon 23/03/15	Mon 30/03/15	5 hrs	Gert-Jan	In deze subtaak worden de gegevens van de optical flow sensor binnengelezen dit zal afhankelijk van het beschikbare communicatieprotocol ofwel via SPA ofwel via I2C gebeuren. De ingelezen data zal mee opgenomen worden in de PID regelaar.
RF Ontvanger - Sturing	Mon 23/02/15	Mon 30/03/15	7 hrs	Gert-Jan	In deze subtaak wordt de data voor de besturing van de quadcopter binnengelezen. Deze data is afkomstig van een externe micrcontroller die in contact staat met de grond. Communicatie kan gebeuren door middel van SPI of I2C.

	Motor Mixing	Mon 20/04/15	Mon 4/05/15	20 hrs	Gert-Jan	
	PWM Generatie	Mon 20/04/15	Mon 27/04/15	5 hrs	Gert-Jan	Om de BLDC motoren aan te sturen is een PWM of PPM signaal nodig. Er werdt gekozen voor een PWM signaal. Hiervoor dient een interface aangemaakt te worden die het mogelijk maakt om deze PWM signalen te genereren.
	Motor Mixing Algoritme	Mon 20/04/15	Mon 4/05/15	15 hrs	Gert-Jan	De motors van de quadcopter staan in voor zowel besturing als hoogte. Elk van de 4 motoren moet op een correcte manier aangestuurd worden om de gewenste attitude en altitude te verkrijgen. De uitvoer van de PID regelaar zal als invoer dienen voor het m...
	Kalman filter implementatie	Mon 23/03/15	Mon 27/04/15	20 hrs		Het inlezen van sensoren brengt vaak veel ruis met zich mee waardoor de berekeningen (PID) gebeuren met niet correcte waarden. Om deze ruis deels weg te filteren zal getracht worden een Kalman filter te implementeren die het ruisgehalt in sensordate ee...
Quadcopter Frame		Tue 3/03/15	Tue 5/05/15	56 hrs		
	Frame Design	Tue 3/03/15	Tue 10/03/15	6 hrs	Gert-Jan	Deze sub-taak omvat het ontwerpen van een quadcopterframe dat geprint kan worden door middel van een 3D-printer. Er dient dus rekening gehouden te worden met de maximale grootte van ieder onderdeel.
	Frame Printing	Tue 10/03/15	Tue 24/03/15	40 hrs	Gert-Jan	
	Quadcopter assembly prototype	Tue 24/03/15	Tue 31/03/15	6 hrs	Gert-Jan & Nick	
	Quadcopter final assembly	Tue 28/04/15	Tue 5/05/15	4 hrs	Gert-Jan & Nick	
Brushless DC sturing		Tue 24/02/15	Mon 2/03/15	12 hrs	Nick	
	Hardware implemetatie	Tue 24/02/15	Mon 2/03/15	8 hrs	Nick	
	Software implementatie	Tue 24/02/15	Mon 2/03/15	4 hrs	Nick	
Battery management		Tue 3/03/15	Mon 9/03/15	14 hrs	Nick	
	Spanningsniveau detectie	Tue 3/03/15	Mon 9/03/15	10 hrs	Nick	Het bijhouden van het batterijniveau is van belang voor de return home functionaliteit en de algemene betrouwbaarheid van de quadcopter. Het spanningsniveau staat in rechtstreeks verband met de nog resterende capaciteit van de batterij, door het spanni...
	Controller overwrite (return home)	Tue 3/03/15	Mon 9/03/15	4 hrs	Nick	Indien de batterijspanning onder een bepaald niveau daalt, dient de quadcopter terug te keren naar de home positie. Dit niveau is dynamisch te bepalen, en is afhankelijk van de nog te vliegen weg en het gemiddelde verbruikte vermogen. Zo kan er gegaran...

MAVlink protocol	Tue 10/03/15	Mon 16/03/15	29 hrs	Nick	
Physical link (RF-WiFi-BT-…)	Tue 10/03/15	Mon 16/03/15	14 hrs	Nick	Het bepalen van het ideale fysische protocol is afhankelijk van verschillende factoren. De mogelijkheden zijn 433MHz, 2.4GHz, Bluetooth (2.4GHz) of WiFi (2.4GHz). Lagere frequenties hebben een kleiner verlies dan hogere frequenties over dezelfde afstan...
Antenna design	Tue 10/03/15	Mon 16/03/15	10 hrs	Nick	Het bekijken en berekenen van het optimale antenne-ontwerp voor deze specifieke toepassing. Dit kan een pcb-antenne zijn, of een externe antenne. De keuze is afhankelijk van het gekozen protocol en kan dan ook pas na deze keuze bepaald worden.
Communicatie test	Tue 10/03/15	Mon 16/03/15	5 hrs	Nick	Het testen van de communicatie met behulp van een simpele opstelling. Zenden en ontvangen kan getest worden met behulp van een opstelling van twee arduinos of mbeds om zo te controlleren dat het MAVlink protocol correct is geïmplementeerd.
Ontwerpen PCB	Tue 17/03/15	Mon 30/03/15	36 hrs	Nick	
PCB design	Tue 17/03/15	Tue 24/03/15	30 hrs	Nick	Het tekenen van de PCB zelf, met alle nodige componenten aanwezig. Hierbij dient rekening gehouden te worden met EMC, door de aanwezigheid van hoogfrequente signalen afkomstig van de transceiver en de PWM signalen voor de motoren.
Controlleren schematic	Tue 24/03/15	Mon 30/03/15	2 hrs	Nick	
Controlleren board	Tue 24/03/15	Mon 30/03/15	3 hrs	Nick	
Genereren Gerber files	Mon 30/03/15	Mon 30/03/15	1 hr	Nick	Het Gerber bestandsformaat is een universeel formaat dat gebruikt wordt bij de productie van PCB's. Om dus een correct geproduceerd bord te verkrijgen dienen deze bestanden te voldoen aan de eisen van de fabrikant.
Optical flow	Tue 31/03/15	Mon 4/05/15	60 hrs	Nick	
Image processing	Tue 31/03/15	Mon 20/04/15	40 hrs	Nick	Image processing is nodig om de optical flow te detecteren. Drift cancellation maakt namelijk gebruik van de data die verkregen wordt uit het correleren van het huidige beeld met het vorige beeld. Op die manier kan berekend worden hoeveel drift de quad...
Return home	Mon 20/04/15	Mon 4/05/15	20 hrs	Nick	De optical flow sensor kan ook ingeschakeld worden om te berekenen in welke mate de quadcopter beweegt. Hiervoor dient continu bijgehouden te worden in welke richting bewogen is in de x en y richting. Bewegingen in de z richting worden opgevangen door ...
Testing - Debugging en Optimalisatie	Tue 5/05/15	Mon 18/05/15	0 hrs	Gert-Jan & Nick	