2013•2014
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Hardware-implementatie en -evaluatie van een satellietpositieberekeningsmodule

Promotor :
dr. Nele MENTENS

Promotor :
dr. ir. WIM AERTS

Copromotor :
ing. RUBEN SMEETS

Jan Moors , Pieter-Jan Ceyssens

Proefschrift ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

KU LEUVEN | universiteit hasselt

KU LEUVEN | universiteit hasselt

2013•2014
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Hardware-implementatie en -evaluatie van een satellietpositieberekeningsmodule

Promotor :
dr. Nele MENTENS

Promotor :
dr. ir. WIM AERTS

Copromotor :
ing. RUBEN SMEETS

Jan Moors , Pieter-Jan Ceyssens

Proefschrift ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Voorwoord

De navigatiesystemen hebben sinds het begin der tijden een lange weg afgelegd om zo nauwkeurig mogelijk te zijn. De primitieve navigatie is immers al lang geleden ontdekt en het kompas en land- en zee kaarten waren de eerste methoden waarop de mensheid zich baseerde als het aankwam op navigeren. Gedurende de jaren zijn er echter veel veranderingen gebeurd en hebben er zich moderne technieken ontwikkeld. De meest recente is die van de satelliet-gebaseerde navigatie. Deze techniek is erg nauwkeurig en is publiek beschikbaar.

Maar het waren niet enkel de technieken die de keuze voor dit onderwerp hebben bekrachtigd. Er zijn nog veel andere factoren die ervoor gezorgd hebben dat het uiteindelijk deze keuze is geworden. Zoals bijvoorbeeld de lessen van dhr. Aerts, waar GPS kort ter sprake kwam. Hierdoor geraakten we gefascineerd door deze techniek en wouden we er graag meer van weten.

Allereerst zouden we graag onze thesisbegeleiders willen bedankt voor de kans die ze ons gaven. We waren niet de enige thesisstudenten die dit onderwerp graag wouden hebben en bij deze willen wij graag Wim en Nele bedanken voor hun vertrouwen. Daarnaast gaat ook een dankwoord uit naar Ruben, die ons steeds voorzien heeft van alles wat wij nodig hadden.

Nog enkele speciale personen zijn onze ouders. Zonder hun steun en toeverlaat waren wij nooit gekomen waar we nu staan. Zij hebben ons steeds alle kansen gegeven en er alles aan gedaan dat wij niets tekort kwamen. Ook een speciaal dankwoord gaat uit naar Elien, die mij doorheen de opleiding steeds gesteund heeft. Bedankt.

Pieter-Jan Ceyssens, Jan Moors, november 2013

Abstract - Dutch

Achtergrond *Het doel van deze thesis was het ontwerpen en implementeren van een meerkanaals GPS ontvanger op een FPGA. De vorige 3 jaar was er al succesvol gewerkt aan deze ontvanger. Zo werden de acquisitie, tracking en demodulatie modules al geïmplementeerd. Het zijn dus deze modules waarop deze thesis verder bouwt. Concreet wilt dit zeggen dat de navigatiedata uit het RAM-geheugen van de demodulatie-module gehaald wordt en dat men met deze data de gebruikerspositie bepaalt.*

Resultaten *In deze thesis wordt de navigatiedata eerst geparset, dit komt neer op het plaatsen van de juiste bits van het navigatiewoord in de juiste variabelen. Vervolgens worden deze variabelen omgezet naar een 64 bit double precision formaat.*

Na het parsen zijn dus alle parameters bekend en is het nog een kwestie van de juiste bewerkingen te doen. Het eerste wat gedaan wordt zijn de operatoren implementeren in hardware, vervolgens worden al deze operatoren juist verbonden en tenslotte worden ze aangestuurd door een FSM.

Een andere voorwaarde van deze module is dat deze geïmplementeerd wordt op een Virtex-4. Het spreekt dus voor zich dat er gekozen wordt om een hardware-implementatie te voorzien van deze module, dit doen we door gebruik te maken van VHDL en de Xilinx ISE tool. Aangezien de FPGA een beperkte werkoppervlakte heeft is het dus ook noodzakelijk dat de module binnen deze oppervlakte blijft.

Conclusie *Deze thesis bespreekt dus de implementatie van een volledig GPS systeem, waarbij de nadruk ligt op de calculatie-module.*

Abstract - English

Background *The goal of this thesis is to design and implement a multi-channel GPS receiver on an FPGA. The previous theses already accomplished the three primary components. They succesfully implemented the acquisition, tracking and demodulation units. This implies that the navigation data is retreived from the RAM-unit of the demodulation module and that the satelliteposition is calculated with this data.*

Results *The first thing that is done is the parsing of the navigation data, this means that the correct bits of the navigation word are placed in the proper variable. Subsequent is the conversion of these variables to a 64 bit double precision format. With this conversion, the accuracy of the receiver is very prompt.*

Next up is the calculation of a satelliteposition. Because all the parameters are known, we now need to solve the right equations, with the correct parameters. The first thing that is done is the implementation of the operators (in hardware), then these operators are connected and controlled with the main FSM.

Another requirement is that the implementation is done on a Virtex-4 FPGA. This means that the implementation has to be done in hardware, and this has been accomplished using VHDL and the Xilinx ISE tool. But the FPGA has limited resources in terms of digital logic, so we have to stay within the ristrictions of the area.

Conclusion *This thesis reviews the implementation of a GPS system, in which the focus lies on the calculation unit.*

Inhoudsopgave

Gebruikte afkortingen

GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GLONASS	GLObal NAvigation Satellite System
FPGA	Field Programmable Gate Array
RAM	Random Acces Memory
ASIC	Application Specific Integrated Circuit
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
MCS	Master Control Station
NAVSAT	Navy Navigation Satellite System
ESA	European Space Agency
CDMA	Code Division Multiple Access
FDMA	Frequency Division Multiple Access
P(Y)-code	Precision Code
C/A-code	Coarse/Acquisition Code
BPSK	Bi-Phase Shift Keying
PRN	Pseudo Random Noise
CDMA	Code Division Multiple Acces
TLM	Telemetry word
HOW	Handover word
TOW	Time Of Week
NCO	Numeric Controlled Oscillator

ROM	Read Only Memory
PLE	Prompt Late Early
FPU	Floating Point Unit
RINEX	Receiver Independent Exchange Format
ALU	Arithmetic Logic Unit
FSM	Finite State Machine
FF	Flip Flop
LUT	Look Up Table
DSP	Digital Signal Processing

Hoofdstuk 1

Inleiding

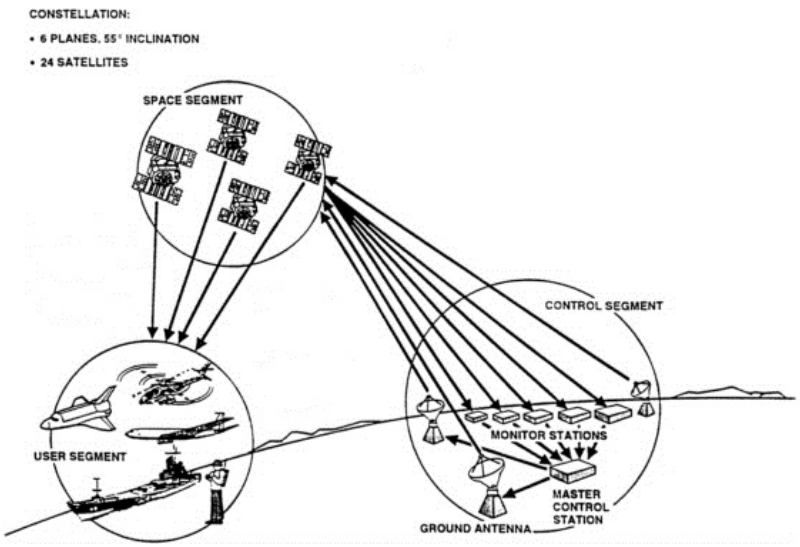
1.1 Situering

De sterrenwacht in Ukkel, Brussel, heeft verschillende onderzoeksdepartementen. Één daarvan is het departement Referentiesystemen en Geodynamica. Dit departement is dan ook nog eens verdeeld in verschillende afdelingen. Hetgene waar wij ons op focussen is Afdeling 1: Tijd, Aardrotatie en Ruimtegeodesie. Enerzijds integreert men in deze afdeling de referentiesystemen van heel België en anderzijds bestuderen zij ook de rotatie en dynamica van de aarde, met name de inwendige en uitwendige vervormingen. Een andere onderzoekstak is het geodetisch en geofysisch onderzoek van andere hemellichamen, o.a. Mercurius, Venus, Mars en de grote manen van ons zonnestelsel.

1.2 Global Navigation Satellite Systems

Een zeer belangrijke rol in dit onderzoek zijn de Global Navigation Satellite Systems (GNSS). Deze moeten ervoor zorgen dat de positiebepalingen steeds nauwkeurig zijn. Deze positiebepalingen steunen op tijdsmetingen. Om een zo nauwkeurig mogelijke plaatsbepaling te realiseren, moet men dus een exact tijds-referentiestelsel hebben en maken ze gebruik van GNSS systemen. Omwille van deze redenen worden er atoomklokken geïnstalleerd in deze GNSS systemen, zowel in het controle als het ruimtesegment, zoals weergegeven in figuur 1.1. Deze klokken worden zodanig geïmplementeerd dat als de satelliet een signaal uitzendt op een gekend tijdstip, het controlesegment dan perfect kan meten hoe lang het signaal erover gedaan heeft om van satelliet tot aarde te propageren.

De GNSS systemen kan men opdelen in verschillende categorieën. Zo wordt er een onderscheid gemaakt tussen het Amerikaanse Global Positioning System (GPS), het Russische GLObal NAvigation Satellite System (GLONASS), het Chinese Beidou en het Europese Galileo. Al deze systemen berusten op hetzelfde principe om aan plaatsbepaling te doen, namelijk het versturen van tijdsignalen via radiogolven.



Figuur 1.1: Gebruiker, ruimte en controle segment. Referentie: allaboutgps101.blogspot.be/

In deze radiogolven zit alle data die nodig is om aan positiebepaling te doen. Deze data wordt echter wel eerst gemoduleerd en gemixt alvorens verzending, zie figuur 3.1. Hetgeen dat al verwezenlijkt is door de voorgaande thesisstudenten zijn de acquisitie, tracking en de demodulatie van de GPS signalen op een Field Programmable Gate Array (FPGA). Voor meer informatie hierover verwijzen we door naar de voorgaande thesissen [4,5,6]. In deze voorgaande thesissen is er veel aandacht geschonken aan het schakelen tussen de verschillende GNSS systemen. Zo is er de mogelijkheid om met een minimale herconfiguratie te wisselen tussen deze systemen. De laatste toevoeging was het opslaan van de navigatiegegevens in een Random Acces Memory (RAM)-geheugen.

Dus kort samengevat; de FPGA spoort de satellieten op, traceert het signaal en slaat de navigatiedata op in het RAM-geheugen. Het doel van deze thesis is om deze data te analyseren en om de navigatiedata om te zetten in bruikbare parameters. Hiermee kan er met deze parameters aan positiebepaling gedaan worden.

1.3 Probleemstelling

Één van de absolute noodzakelijkheden is dat al deze bewerkingen gebeuren op een FPGA. De reden hiervoor is als volgt; de rekenkracht van de FPGA's is erg sterkt toegenomen in de laatste decennia, zodanig dat men alle bewerkingen kan uitvoeren die nodig zijn om een GNSS signaal te verwerven en deze te tracken. In tegenstelling tot vroeger, want toen was een GNSS ontvanger een Application Specific Integrated Circuit (ASIC) design.

Een voordeel om een FPGA boven een ASIC te kiezen is de kostprijs. Het is namelijk zo dat één enkele ASIC erg kostelijk is, rekening houdend met ontwikkelkosten, productiekosten, etc .... Bovendien is het ook zo dat als er een fout zit in een ASIC het hele ontwikkel- en productieproces moet worden overgedaan. Laat dit net de kracht zijn van een FPGA, want deze kan men *on the fly* herconfigureren. Er kan dus besloten worden dat een FPGA een betere keuze is geweest omwille van zijn flexibiliteit en kostprijs.

Bovendien is er al een bestaande ontvanger voortgebracht uit voorgaande thesissen en zij hebben daarbij gekozen om hun implementatie op een FPGA te doen. De reden hiervoor is dat de FPGA's erg flexibel zijn en aangezien binnen onderzoeken flexibiliteit altijd overwint van oplages hebben zij dus een FPGA gekozen.

Afgezien van al deze voordelen zijn er ook nadelen verbonden aan een FPGA design. Zo is naast rekenkracht ook de oppervlakte van de ingenomen logica van belang. De afweging *snelheid* versus *oppervlakte* is een erg belangrijke factor in het hedendaagse chip-design en is in deze thesis dus ook een belangrijk topic. De voorgaande thesisstudenten hebben er dan ook alles aan gedaan om zo klein mogelijk te werken en nog steeds aan de eisen van de sterrenwacht te voldoen. Maar al deze bestaande logica neemt natuurlijk al een deel in van de FPGA.

Zoals reeds vermeld is het onze taak om bruikbare parameters uit een dataset te halen en deze vervolgens te processen om uiteindelijk een positieberekening te kunnen doen. Om hiertoe te komen moeten er ook heel wat hardware-blokken voorzien worden, die uiteraard ook weer kostbare oppervlakte van de FPGA innemen. Voor deze thesis zal het dan ook weer een uitdaging worden deze oppervlakte tot een minimum te beperken.

1.4 Materialen en methoden

Doorheen de ontwikkeling van de positiebepaling op FPGA zal er gebruik worden gemaakt van verschillende materialen en methodes om tot het gewenste resultaat te komen. Hiervan zullen de belangrijkste hieronder kort worden toegelicht.

Het eerste belangrijk stuk hardware waar gebruik van gemaakt zal worden bij deze masterproef is een FPGA. Dit zal uiteindelijk de rekenkracht worden die, uitgaande van de aangekregen GPS signalen, de positie van de eindgebruiker zal bepalen. De FPGA die gebruikt zal worden is de Virtex 4 FPGA van Xilinx die ter beschikking wordt gesteld.

Om de FPGA te programmeren wordt er gebruik gemaakt van Very high speed integrated circuit Hardware Description Language (VHDL) waarmee in codevorm de hardware kan worden beschreven die later de GPS zal vormen. Om op een handige manier deze VHDL code te schrijven werd er gebruik gemaakt van de ISE Design Suite, welke gratis te downloaden is op de site van Xilinx. Buiten de VHDL code schrijven, is het met dit programma ook mogelijk testen uit te voeren om de correcte werking van de code te garanderen alvorens ze te gaan programmeren op de FPGA. Een andere handige functie van de ISE Design Suite die gebruikt werd is om na te gaan hoeveel resources het ontwerp zal gebruiken eens omgezet op de FPGA, om zo het ontwerp te optimaliseren naar een zo klein mogelijk resource gebruik.

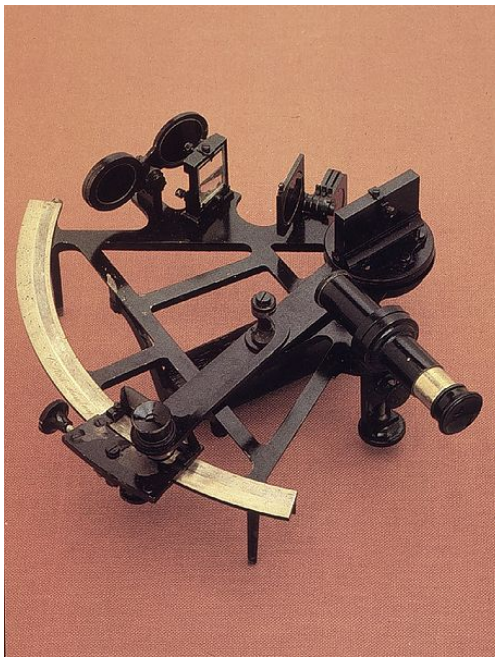
Tijdens de ontwikkeling is er ook veel gebruik gemaakt van Matlab. Dit vooral om het algoritme achter het berekenen van de satelliet- en userpositie uitvoerig te testen. In het ontwerp zijn er ook verschillende benaderingen gemaakt om de implementatie op FPGA mogelijk te maken. Het voordeel van Matlab is ook dat de berekeningen makkelijk grafisch weer te geven zijn, vergeleken kunnen worden met de werkelijke waarden en snel correcties kunnen worden doorgevoerd.

Hoofdstuk 2

Basisprincipes van de navigatie

Vooraleer de mensheid toegang had tot navigatie via satellieten, zijn er nog heel wat andere instrumenten vervaardigd. Reeds in 206 voor Christus was het eerste magnetische kompas al uitgevonden, in China. Uiteraard was deze bedoeld voor militaire toepassingen en zoals steeds worden nieuwe benodigdheden vaak in een militaire omgeving eerst uitgevonden, alvorens men overgaat naar civiele toepassingen. Een logische toevoeging aan het kompas was de landkaart, eens men deze twee tools had, kon men zich al vrij nauwkeurig navigeren. Het zou wel echter nog jaren duren vooraleer men deze kaarten erg nauwkeurig kon opstellen.

Een andere belangrijke navigatietool was de sextant. Dit instrument werd voornamelijk gebruikt om hoeken te meten tussen twee zichtbare objecten. Deze uitvinding dateert van 1730 en wordt in onderstaande figuur weergegeven.



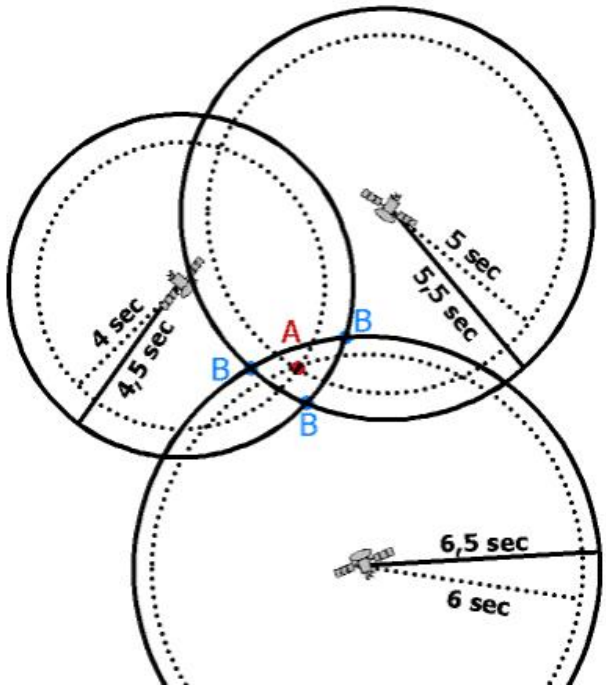
Figuur 2.1: Een voorbeeld van een sextant. Referentie: wikipedia.org

Met de opkomst van het digitale tijdperk werden veel van deze instrumenten vervangen door hun elektronische equivalent. Zo werd er in de scheepvaart veelvuldig gebruik gemaakt van radiocontact om zo de positie van de schepen te berekenen. In 1957 werd de eerste satelliet gelanceerd, de *Sputnik*, wat aanleiding gaf tot de *Space-race*. Dit zorgde ervoor dat alle grote wereldmachten zich meer toelegden op de ruimtevaart en op 20 juli 1969 landde de eerste mens op de maan. Al snel begonnen wetenschappers zich te interesseren in al deze nodes en waren de GNSS systemen geboren.

Zoals reeds vermeld in de situering, paragraaf 1.1, zijn er veel verschillende implementaties van de GNSS systemen. In deze paragraaf worden de vier verschillende systemen vergeleken en kort besproken. De eerste navigatiesatelliet werd ontwikkeld door het Amerikaanse leger, namelijk de Navy Navigation Satellite System (NAVSAT). Dit systeem maakte gebruike van het dopplereffect om posities te bepalen en had vijf satellieten in dienst. Omwille hiervan kon er slechts een paar uur per dag een positie berekend worden. Wat volgde was de ontwikkeling van verschillende GNSS systemen. In onderstaande subsecties wordt een korte inleiding gegeven over hoe GNSS systemen werken. Vervolgens worden kort verschillende grote soorten besproken maar voor meer info wordt doorverwezen naar andere bronnen [3].

2.1 Werking moderne GNSS systemen

Het principe van positiebepaling aan de hand van GNSS systemen berust volledig op de tijd dat het signaal, gestuurd door de satelliet, onderweg is naar de ontvanger van de eindgebruiker. De satelliet zal namelijk telkens zijn exacte tijdstip doorsturen. Wanneer deze boodschap de ontvanger bereikt kan deze het verschil in tijd berekenen tussen het verzenden en het ontvangen van het signaal. Hiermee kan vervolgens, wetende dat het signaal met de snelheid van het licht propageert, de afstand tot de desbetreffende satelliet worden bepaald. Wanneer men deze techniek toepast om de afstand tot meerdere satellieten te verkrijgen kan men de positie van de eindgebruiker bepalen. Hierbij wordt er gebruik

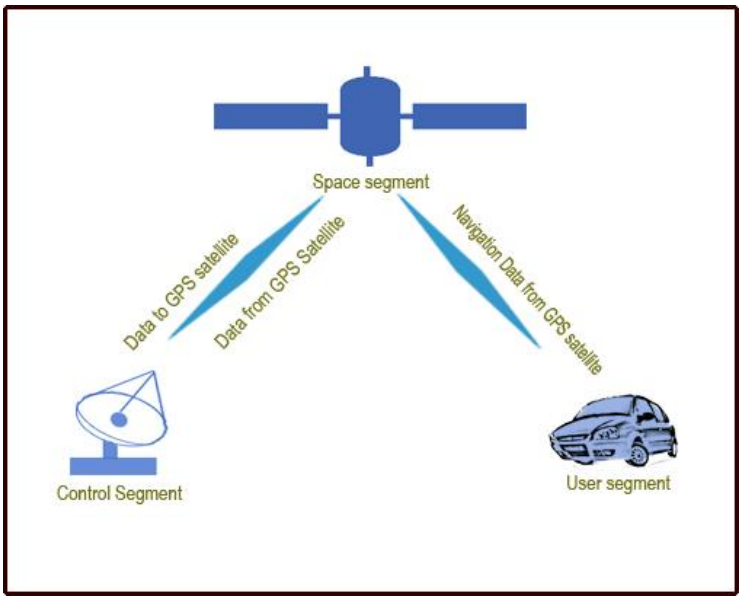


Figuur 2.2: concept van plaatsbepaling

gemaakt van cirkels met als straal telkens de afstand tot de desbetreffende satelliet. Om de positie van de gebruiker nu te bepalen moet men het snijpunt van de eerder vermelde cirkels berekenen zoals ook weergegeven in de bovenstaande figuur. Hierbij worden minstens 4 satellieten gebruikt om een zo accuraat mogelijke positie te bekomen. Op deze manier worden er vier bollen bekomen waarvan het snijpunt bepaald kan worden.

2.1.1 Structuur GNSS systemen

Om de structuur van de GNSS systemen verder uit te leggen kunnen we het systeem verder opdelen in drie grote segmenten. Deze bestaan uit het `gebruikerssegment`, het `controlesegment` en ten laatste het `ruimtesegment`. Elk van deze segmenten hebben een belangrijke rol om een precieze plaatsbepaling mogelijk te maken.



Figuur 2.3: Structuur van een GPS. Referentie: engineersgarage.com

Gebruikerssegment

Dit is het segment waar wij als eindgebruiker het beste mee vertrouwd zijn, ook omdat we hier bijna dagelijks mee in aanraking komen. Tot dit segment horen namelijk alle ontvangers die, voor civiel of militaire doeleinden, het mogelijk maken om aan plaatsbepaling te doen door middel van GNSS technologie. Bekende voorbeelden zijn de GPS ontvangers die we vinden in de auto of smartphone.

Controlesegment

Naast het gebruikerssegment hebben we ook het controlesegment. Dit segment zal de satellieten bijsturen zodat eventuele fouten op de uiteindelijke positiebepaling zo minimaal mogelijk blijven. Dit wordt gedaan in verscheidene monitorstations die op strategische plaatsen zijn gelegen, verspreid over de hele wereld zoals te zien op figuur 2.5.

Figuur 2.4: Voorbeeld van een ontvanger

Deze zullen erop toezien dat de klokken in de verschillende satellieten synchroon zijn en indien dit niet het geval is ze bijsturen. Ook zullen de monitorstations de banen van de satellieten bijhouden en vervolgens deze doorgegeven aan het Master Control Station (MCS), welke op zijn beurt deze informatie terug doorgeeft aan de satelliet.

Ruimtesegment

Het ruimtesegment (van de GPS) bestaat uit 24 satellieten die elk verdeeld zijn over 6 banen in de ruimte. Deze banen hebben een inclinatiehoek van 55° en snijden de evenaar om de 60°. De hoogte van de satellieten is zo gekozen dat elke satelliet 2 maal per siderische dag een volledig baan rond de aarde aflegt. Dit maakt dat elke satelliet een orbitstraal heeft van 26 561,8 km. De reden waarom de satellieten op deze manier verspreid zijn over de aarde is om ervoor te zorgen dat op elke moment en op elke locatie er steeds genoeg satellieten zijn om een nauwkeurige plaatsbepaling te verkrijgen.

Figuur 2.5: Voorbeeld van een monitorstation. Referentie: wikipedia.org

2.2 GPS

Dit systeem werd ontwikkeld in 1973 door het Amerikaanse *Department of Defense*, echter enkel beschikbaar voor militaire doeleinden. Dit veranderde in 1983 toen President Ronald Reagan besloot de GPS ook beschikbaar te maken voor civiele doeleindes nadat vlucht 007 van Korean Air Lines werd neergehaald omdat deze zich in verboden luchtruim van Rusland bevond. Er volgden 23 lanceringen waarna in 1994 alle satellieten, nodig voor een correcte werking van de GPS, operationeel waren. Een jaar later werd het systeem volledig operationeel.

Hoewel dat Ronald Reagan in 1983 had gesteld dat het GPS systeem ook voor civiele doeleinden kon gebruikt worden was er toch een verschil tussen het militaire en het civiele GPS signaal. Het signaal gebruikt voor civiele toepassing werd namelijk gedegradeerd door middel van Selective Availability (SA). Dit systeem zorgde voor een willekeurige fout die aan het signaal werd toegevoegd zodat de positiebepaling tot 100 m verschoven kon worden.

Later werd deze SA omzeild door op een gekende locatie de fout te bepalen van het GPS signaal en deze door te sturen naar de andere GPS ontvangers. Door het gebruik van deze techniek, gekend als Differential GPS (DGPS) konden de andere GPS ontvangers hun locatie aanpassen met de doorgestuurde fout, waardoor ook de civiele gebruikers een nauwkeurigere positiebepaling verkregen. Omdat het gebruik van SA overbodig werd door DGPS besloot President Bill Clinton dan ook dat de SA uitgeschakeld werd op 1 mei 2000.

Figuur 2.6: Voorbeeld van een satelliet. Referentie: wikipedia.org

2.3 Galileo

Galileo is een onderzoeksproject van de Europese Unie en de European Space Agency (ESA) met als doel een volledig autonoom GNSS systeem te maken, onafhankelijk van de andere systemen, en is genoemd naar de Italiaanse astronoom Galileo Galilei. Dit onderzoeksproject heeft als grote voordeel dat het erg nauwkeurig gaat zijn, alsook dat de Europese Unie dit navigatiesysteem kan blijven handhaven in tijden van oorlog.

Galileo is officieel gestart op 26 mei 2006 en zou tegen 2019 volledig operationeel moeten zijn. Om dit te realiseren krijgt het project de steun van veel verschillende overheden. Uiteindelijk zouden er 30 satellieten in gebruik moeten zijn en deze zouden in een baan rond de aarde zweven met een straal van 23,222 km.

In theorie zou Galileo dus erg nauwkeurig zijn en dit was tegen de zin van de Amerikanen want het zou dus mogelijk kunnen zijn dat deze techniek gebruikt werd om Amerikaanse doelen te treffen met raketten die geleid worden via Galileo.

Omdat Galileo deels ook in dezelfde frequentieband ligt als het GPS systeem is het voor de Amerikanen dus ook onmogelijk om (ten tijden van oorlog) de signalen van Galileo te blokkeren en het gerucht ging dan ook de ronde dat de Amerikanen de Galileo satellieten zouden neerhalen (wederom, ten tijden van oorlog). Om deze reden hebben de onderzoekers van het Galileo project ervoor gekozen om hun frequentieband anders te kiezen.

Galileo heeft drie verschillende segmenten namelijk:

- Space Segment; dit zijn de 30 satellieten, waarvan 27 operationeel en 3 zijn reserve.
- Ground Segment; er zijn twee *ground control centres* die gelegen zijn in Oberpfaffenhofen-Duitsland en Fucino-Italië.
- User Segment; er zijn verschillende soorten ontvangers, bijvoorbeeld de GARDA ontvanger.

2.4 GLONASS

2.4.1 Eerste generatie

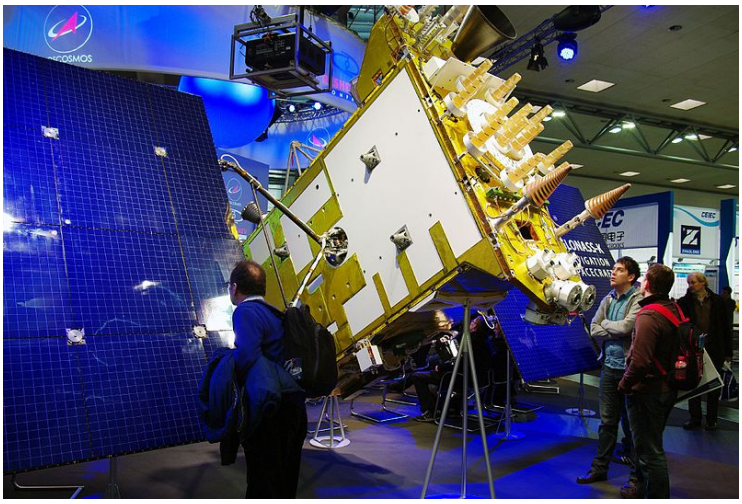
Net zoals bij Galileo, waarbij de Europese Unie niet meer afhankelijk wou zijn van het Amerikaans GPS is het GLONASS ook een volledig autonoom GNSS systeem. De ontwikkeling van dit systeem is begonnen in de jaren 60 en de eerste *Tsiklon* satellieten waren gelanceerd in 1967. Deze satellieten waren er om initiële testen mee uit te voeren. Deze satellieten (de eerste generatie) werden tussen 1985 en 1986 vervangen door *Block IIa* satellieten. Deze hadden verbeterde tijds- en frequentiestandaarden. De opvolger van deze *Block IIa* waren de *Block IIv* satellieten. Deze werden gebruikt van 1988 tot 2000. Deze reeks satellieten waren, volgens Russische bronnen, de beste satellieten van de eerste generatie.

2.4.2 Tweede generatie

De ontwikkeling van de tweede generatie satellieten, de *Glonass-M*, begon in 1990 en de eerste lancering was in 2003. Deze generatie had een langere levensduur, verbruikte minder vermogen dan hun voorganger en had cesium atoomklokken aan boord. Er werden in het totaal 14 van deze satellieten in een baan rond de aarde gebracht.

2.4.3 Derde generatie

De meest recente generatie zijn de *Glonass-K* satellieten. Deze generatie bood ook weer veel voordelen t.o.v. zijn voorganger, zo waren deze veel lichter (750 kg t.o.v. 1450 kg), hebben ze een levensduur van 10 jaar en maken ze gebruik van CDMA (Code Division Multiple Acces) in plaats van FDMA (Frequency Division Multiple Acces) signalen.

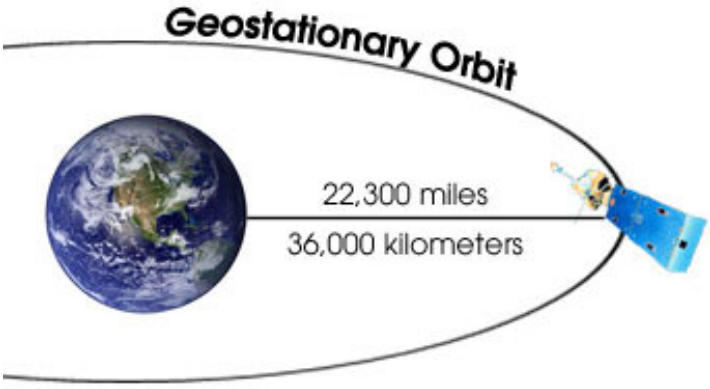


Figuur 2.7: Glonass-K satelliet. Referentie: wikipedia.org

2.5 BeiDou

Ook China heeft zijn eigen GNSS systeem ontwikkeld genaamd BeiDou Navigation Satellite System (BDS). Het systeem is momenteel nog in de ontwikkelingsfase maar het testsysteem is al operationeel.

Het eerste systeem wat ontwikkeld werd, was eerder een testsysteem en werd BeiDou-1 genoemd. Het systeem bestond uit drie satellieten die gebruikt werden voor plaatsbepaling in China zelf. Om ervoor te zorgen dat er met drie satellieten gewerkt kon worden werd er geopteerd voor satellieten die het aardoppervlakte volgen, de zogenaamde geostationaire satellieten. Dit zorgt ervoor dat de drie satellieten altijd te gebruiken zijn in China, en dat deze nooit achter de horizon verdwijnen. Hiervoor moeten deze satellieten op een hoogte van 36000 km boven het aardoppervlakte zweven. Later werd ook BeiDou-2 gecreëerd die



Figuur 2.8: Geostationaire satelliet

het mogelijk zal maken om niet alleen in China, maar overal aan plaatsbepaling te kunnen gaan doen. Dit systeem zal gaan werken met een reeks van 35 satellieten waarvan er 30 in een baan rond de aarde gaan bewegen en 5 ervan ook op geostationaire hoogte zullen hangen. In 2020 zou China BeiDou-2 volledig werkend willen hebben.

2.6 Overzicht

In onderstaande tabel wordt een volledig overzicht gegeven van de verschillende systemen.

Characteristics	GPS	GLONASS	Galileo	Compass
First launch	February, 1978	October, 1982	December, 2005	April, 2007
Full operational capability	February, 1995	January, 1996 - December, 2011	2012,2013	up to 2020
Funding	public	public	public & private	public
Nominal number of SV	24	24	27	27
Orbital planes	6	3	3	3
Orbit inclination	55°	64.8°	56°	55°
Semi-major axis	26,560 km	25,508 km	29,601 km	21,500 km
Orbit plane separation	60°	120°	120°	–
Revolution period	11h 57.96 min	11h 15.73 min	14h 4.75 min	12h 35 min
Geodetic reference system	WGS-84	PE-90	GTRF	CGS2000
Time system	GPS time, UTC (USNO)	GLONASS time, UTC(SU)	Galileo system time	Bei Dou System Time (BDT)
Signal separation	CDMA	FDMA	CDMA	CDMA
Number of frequencies	3-L1,L2,L5	one per two antipodal SV	3(4)-E1,E6,E5(E5a,E5b)	3-B1,B2,B3
Frequency [MHz]	L1: 1,575.420	G1: 1,602.000	E1: 1,575.420	B1: 1,575.420
	L2: 1,227.600	G2: 1,246.000	E6: 1,278.750	B2: 1,191.795
	L3: 1,176.450	G3: 1,204.704	E5: 1,191.795	B3: 1,268.520
Number of ranging codes	11	6	10	–

Figuur 2.9: Overzicht van de verschillende GNSS systemen. [3]

Hoofdstuk 3

Navigatieboodschap

In dit hoofdstuk wordt de opbouw van de verzonden/ontvangen navigatieboodschap besproken. Het is essentieel de boodschap op een juiste manier te interpreteren, aangezien de boodschap alle informatie bevat die nodig is om aan plaatsbepaling te doen. Merk op dat alle volgende implementaties gebaseerd zijn op het GPS systeem.

Eerst wordt er een algemene opbouw besproken van de boodschap, vervolgens worden de nodige parameters uit deze boodschap besproken (niet alle variabelen zijn noodzakelijk in deze thesis) en tenslotte wordt besproken hoe we met deze parameters aan positiebepaling kunnen doen.

3.1 Genereren van de navigatieboodschap

Een GPS-satelliet zendt op twee verschillende manieren data door. Enerzijds is er de Coarse/Acquisition (C/A) code en anderzijds de Precision (P(Y)) code. De P(Y) code is echter enkel beschikbaar voor militaire doeleinden en de C/A code voor commerciële. Dus wordt er in deze thesis enkel gewerkt met de C/A code, aangezien de P(Y) code gedecrypteerd moet worden. De C/A codes worden op een spreading code gezet met een bitrate van $1.023MHz$, en de P(Y) codes heeft een bitrate van $10.23MHz$.

Een GPS satelliet zendt zijn data in twee verschillende frequentiebanden uit, namelijk L1 en L2. L1 en L2 zijn respectievelijk de primaire en de secundaire L-banden. De centrale frequentie van L1 ligt op $1575.42MHz$ en die van L2 op $1227.6MHz$. Deze twee frequenties zijn coherent met een $10.23MHz$ klok, en zijn als volgt gerelateerd.

L1	$1575.42MHz$	$154 \times 10.23MHz$
L2	$1227.6MHz$	$120 \times 10.23MHz$

Het is van het grootste belang dat deze klokfrequentie van $10.23MHz$ erg nauwkeurig is. Dit wordt bereikt door gebruik te maken van een atoomklok. Het is zelfs zo dat bij de generatie van deze frequentie, de klok zo wordt ingesteld dat hij een iets lagere frequentie genereert. De klok genereert namelijk een frequentie die $4.4647 \times 10^{-10} Hz$ minder bedraagt. Dit wordt gedaan om het relativistische aspect in rekening te brengen. [1, pag 74]

Algemeen gezien kan men een signaal voorstellen door onderstaande vergelijking:

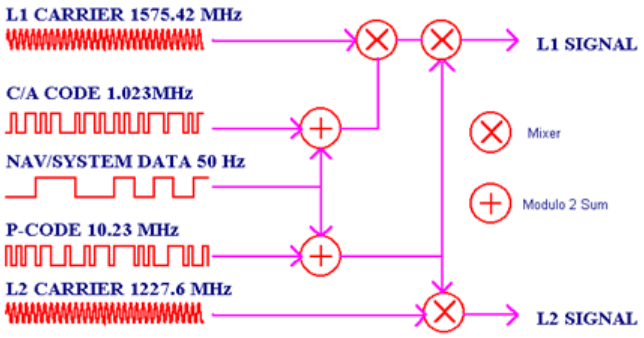
$$S = Asin(2\pi ft + \Phi)$$

Bij GPS systemen maakt men gebruik van fasemodulatie, concreet komt dit erop neer om Φ te gaan wisselen tussen 0 en π, naar gelang de databit. Dit noemt men Bi-Phase Shift Keying (BPSK)

Er kan een probleem ontstaan als alle satellieten op dezelfde frequentie hun data gaan uitzenden, dit wordt opgelost door het invoeren van Pseudo Random Noise (PRN) sequenties. Dit is een opeenvolging van bits die uniek zijn voor iedere satelliet. Op deze manier kunnen alle satellieten op dezelfde draaggolffrequentie zenden, en kan men ze toch nog van elkaar onderscheiden. Dit noemt men Code Division Multiple Acces (CDMA) signalen.

Voor iedere satelliet moet men dus een unieke PRN code kunnen ontcijferen. We weten ook dat iedere satelliet unieke navigatiedata uitzendt, dus wordt de PRN code gemengd met de navigatieboodschap. Figuur 3.1 geeft een samenvatting weer van hoe een navigatieboodschap tot stand komt.

Wat opvalt is dat de C/A-code enkel verzonden wordt in de L1 band, terwijl de P(Y)-code zowel in L1 als in L2 wordt verzonden. Dit heeft als voordeel dat als er een storing in de ene band voorkomt, deze niet per se in de andere band voorkomt. Helaas is dit enkel voor militaire toepassingen beschikbaar.

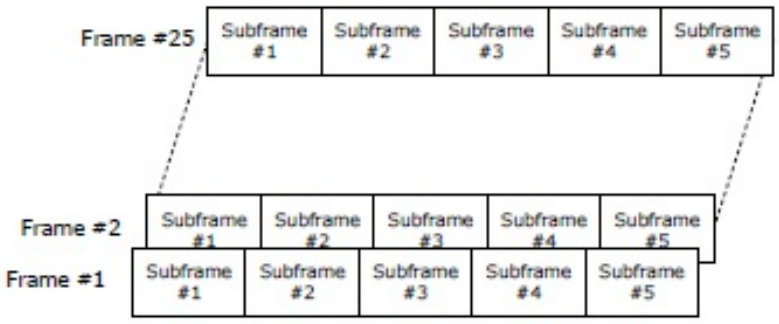


Figuur 3.1: Hoe navigatiedata tot stand komt. Referentie: colorado.edu

3.2 Opbouw van de navigatieboodschap

Een volledige dataset bevat 25 frames, één frame bevat 5 subframes, één subframe bevat 10 woorden en tenslotte is één woord opgebouwd uit 30 navigatiebits. De navigatieboodschap wordt overgezonden tegen een erg lage 50 bits/s. Dit wil dus zeggen dat iedere databit 20 ms lang is. Aangezien een C/A code een lengte heeft van 1 ms, wil dit dus ook zeggen dat er in 20 C/A codes in één navigatiebit zitten. Het duurt dus ongeveer 12.5 minuten vooraleer de hele dataset verzonden is.

Ieder subframe begint steeds met een Telemetry word (TLM) en een Handover word (HOW). Deze twee woorden geven aan dat er een nieuw subframe begint, en worden ook gebruikt om de ontvanger te synchroniseren. Ook zit in ieder subframe de Time Of Week (TOW). Deze geeft aan wanneer de satelliet zijn boodschap heeft uitgezonden.



Figuur 3.2: Opbouw van de navigatieboodschap. [3]

In het eerste subframe zit informatie over de toestand van de satelliet, de clock-correcties en de weeknummers. In subframe 2 en 3 bevinden zich de ephemeris data van de satelliet. Deze informatie is noodzakelijk om de positie van de satelliet te berekenen. Het zijn dus ook deze twee subframes die zo meteen meer in detail worden besproken. In subframes 4 en 5 bevinden zich de toestanden van de andere satellieten in de constellatie en zijn dus niet zo belangrijk binnen deze scope.

3.2.1 Subframes 1, 2 en 3

Zoals reeds vermeld bevindt de nuttige data zich in subframes 2 en 3 maar in deze thesis worden ook alle parameters van subframe 1 ingelezen. Dit is gedaan met het oog op de toekomst, het moet mogelijk zijn om via de boodschap de clock-correcties te gaan toepassen en bij deze is die mogelijkheid nu aanwezig. In onderstaande tabellen worden de verschillende parameters, samen met hun plaats in het subframe, weergegeven. Voor meer detail over de parameters wordt graag doorverwezen. [1]

Subframe 1

Parameter	Location	Number of Bits	Scale Factor (LSB)	Effective Range**	Units
WN: Week number	61-70	10	1		week
Satellite accuracy	73–76	4			
Satellite health	77–82	6	1		
IODC: Issue of data, clock	83–84 211–218	10			
T_{GD}: Satellite group delay differential	197–204	8*	2^{-31}		seconds
t_{oe}: Satellite clock correction	219–234	16	2^{4}	604,784	seconds
a_{f2}: Satellite clock correction	241–248	8*	2^{-55}		sec/sec^2
a_{f1}: Satellite clock correction	249–264	16*	2^{-43}		sec/sec
a_{f0}: Satellite clock correction	271–292	22*	2^{-31}		seconds

Figuur 3.3: Parameters van subframe 1. [1]

Subframe 2

Parameter	Location	Number of Bits	Scale Factor (LSB)	Effective Range	Units
IODE	61–68	8			(see text)
C_{rs}: Amplitude of the sine harmonic correction terms to the orbit radius	69–84	16	2^{-5}		meters
Δn: Mean motion difference from computed value	91–106	16*	2^{-43}		semicircles/ sec
M_0: Mean anomaly at reference time	107–114; 121–144	32*	2^{-31}		semicircle
C_{uc}: Amplitude of the cosine harmonic correction term to the argument of latitude of argument oflatitude	151–166	16*	2^{-29}		radians
e_s: Eccentricity	167–174; 181–204	32	2^{-33}	0.03	dimensionless
C_{us}: Amplitude of the sine harmonic correction term to the argument of latitude	211–226	16*	2^{-29}		radians
$\sqrt{a_s}$: Square root of the semimajor axis	227–234; 241–264	32	2^{-19}		meters$^{1/2}$
t_{oe}: Reference time ephemeris	277–286	16	2^4	604,784	seconds

Figuur 3.4: Ephemeris parameters van subframe 2. [1]

Subframe 3

Parameter	Location	Number of Bits	Scale Factor (LSB)	Effective Range	Units
C_{ic}: Amplitude of the cosine harmonic correction term to angle of inclination	61–76	16*	2^{-29}		radians
Ω_e: Longitude of ascending node of orbit plane at weekly epoch	77–84; 91–114	32*	2^{-31}		semicircles
C_{is}: Amplitude of the sine harmonic correction term to angle of inclination	121–136	16*	2^{-29}		radians
i_0: Inclination angle at reference time	137–144; 151–174	32*	2^{-31}		semicircles
C_{rc}: Amplitude of the sine harmonic correction term to the orbit radius	181–196	16*	2^{-5}		meters
ω: Argument of perigee	197–204; 211–234	32*	2^{31}		semicircles
$\dot{\Omega}$: Rate of right ascension	241–264	24*	2^{-43}		semicircles/sec
IDOE	271–278				(see text)
idot: Rate of inclination angle	279–292	14*	2^{-43}		semicircles/sec

Figuur 3.5: Ephemeris parameters van subframe 3. [1]

Hoofdstuk 4

Bepalen van de userpositie

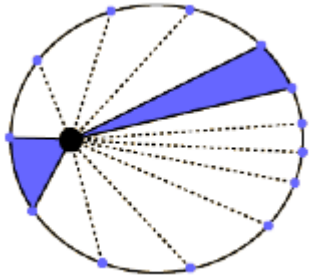
Aangezien bovenstaande paragraaf de parameters bespreekt die worden doorgezonden, is het nu de bedoeling om deze parameters juist te gaan interpreteren om uiteindelijk een positie te bepalen. In deze paragraaf volgen de basisprincipes die uitgevoerd worden om de (user)positie te bepalen.

4.1 Basisprincipes

De basis van satellietpositiebepaling berust op de drie wetten van Keppler [1], deze zijn als volgt:

- *Eerste wet:* De baan van iedere planeet is een ellips met de zon in zijn focus.
- *Tweede wet:* De voerstraal (de verbindingslijn van de zon met de planeet) doorloopt gelijke gebieden in gelijke tijdsintervallen. Figuur 4.1 maakt dit duidelijker. Zoals getoond is de afstand tussen opeenvolgende punten op de ellips variabel, toch zullen de blauwe oppervlakten even groot zijn, en dit binnen dezelfde tijd. Dit komt omdat de planeet sneller beweegt, naarmate ze korter bij de zon is. Mathematisch wordt deze als volgt voorgesteld.[1, p44]

$$\frac{t - t_p}{A_1} = \frac{T}{\pi a_s b_s} \tag{4.1}$$



Figuur 4.1: Tweede wet van Keppler.[6]

- *Derde wet:* Het kwadraat van de omlooptijd van een planeet is proportioneel met de derde macht van de gemiddelde afstand tot de zon. In wiskundige vorm kan deze wet als volgt worden geschreven.

$$\frac{T^2}{a_s^3} = \frac{4 \cdot \pi}{\mu} \tag{4.2}$$

In deze vergelijking is T de siderische periode van de satelliet, a_s de gemiddelde afstand tot de aarde. μ en π zijn constanten, respectievelijk de universele gravitatieconstante van de aarde en de verhouding van omtrek en diameter van een cirkel.

4.2 Bepalen van de satellietpositie

Aan de hand van deze wetten kan men de positie van een planeet uitdrukken in functie van de tijd. Voor een gedetailleerde uitwerking hiervan wordt verwezen naar [1, pagina 45 - 46]. De eerste vergelijking die wordt opgesteld is die van de *mean motion* n. Deze geeft de gemiddelde hoeksnelheid van de satelliet weer.

$$n = \sqrt{\frac{\mu}{a_s^3}} \tag{4.3}$$

In bovenstaande vergelijking stelt μ een constante voor, zijnde de gravitatieconstante en deze bedraagt $9386005 \times 10^8 m^3/s^2$.

De tweede vergelijking kan worden afgeleid uit de tweede wet van Keppler. Hieronder wordt enkel het resultaat weergegeven. Deze vergelijking wordt in vergelijking 4.6 verder uitgewerkt.

$$M \equiv (E - e_s \cdot \sin(E)) = n(t_c - t_p) \tag{4.4}$$

De GPS tijd die wordt doorgezonden door de satelliet heeft slechts een vooropgesteld aantal bits ter beschikking om zijn *time of transmission* mee te vullen. Om deze reden heeft men ervoor gekozen om de satelliet een tijd te laten doorsturen die aangeeft hoeveel seconden het geleden is sinds de overgang van zaterdagnacht op zondagnacht. Het is ook zo dat deze tijd iedere week opnieuw op 0 gezet wordt.

Deze transmissietijd (t_c) heeft dus op iedere maandagmorgen om 8u 00 min en 00 seconden de volgende waarde: 1 dag $\times$24 uur $\times$60 minuten $\times$60 seconden + 8 uur $\times$60 minuten $\times$60 seconden = 115200 seconden. Aangezien deze tijd iedere *week* gereset wordt kan deze dus maximaal 7 dagen $\times$24 uur $\times$60 minuten $\times$60 seconden = 604800 seconden bedragen.

Het is om deze reden dat er een correctie moet gebeuren op de transmissietijd van de gps. De correctie wordt als volgt gedaan, waarbij 302400 overeenkomt met een halve week.

$$als\, t_c - t_{oe} > 302400\, dan\, is\, t_c \Rightarrow t_c - 604800 \tag{4.5a}$$

$$als\, t_c - t_{oe} < -302400\, dan\, is\, t_c \Rightarrow t_c + 604800 \tag{4.5b}$$

Waarbij t_c de transmissietijd van de GPS is en t_{oe} de referentietijd is, die ook door de GPS wordt doorgestuurd.

M en E stellen respectievelijk de gemiddelde en de excentrische afwijking voor. Volgens tabel 3.3 zendt de gps satelliet enkel de *mean anomaly* 'M' door. Dit wil zeggen dat de excentrische afwijking als volgt kan worden berekend.

$$E = (M + e_s \cdot \sin(E)) \tag{4.6}$$

Op het eerste zicht lijkt deze vergelijking erg simpel op te lossen, maar deze is niet lineair. Deze vergelijking moet iteratief worden opgelost. Dit wordt als volgt gedaan.

$$E_{i+1} = (M + e_s \cdot \sin(E_i)) \tag{4.7}$$

waarbij E_{i+1} de huidige waarde is, en E_i de vorige. Bij de start van deze iteratie (dus bij E_0) is $E = M$. Het aantal iteraties is zelf te bepalen. Volgens [1] kan deze varieren van 5 tot 10. Ter referentie: In de voorgaande thesis heeft men deze 5 keer laten itereren alvorens men met E verder rekende.

Eenmaal E gekend is, kan men de *true anomaly* 'ν' berekenen. Deze wordt in detail besproken in [1, pagina 48 - 49]. Hieronder worden enkel de resultaten besproken. Men kan ν vinden door onderstaande vergelijkingen op te lossen.

$$\nu_1 = \arccos \frac{cos(E) - e_s}{1 - e_s \cdot cos(E)} \tag{4.8}$$

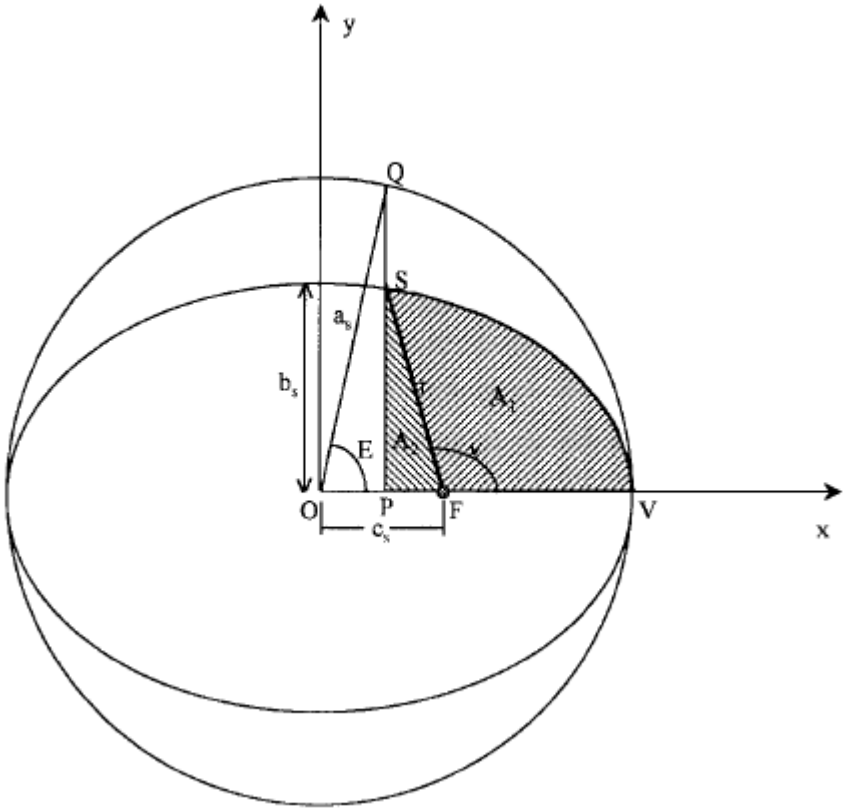
$$\nu_2 = \arcsin \frac{\sqrt{1 - e_s} \cdot sin(E)}{1 - e_s \cdot cos(E)} \tag{4.9}$$

Er valt op te merken dat ν_1 positief is voor alle waarden van E en dat ν_2 positief is voor $E = 0$ tot π en negatief voor $E = \pi$ tot $2 \cdot \pi$. De *true anomaly* kan dus als volgt gevonden worden.

$$\nu = \nu_1 \cdot sign(\nu_2) \tag{4.10}$$

Waarbij $sign(\nu_2)$ ofwel $+1$ ofwel -1 is. Merk op dat om ν te vinden, enkel M en e_s nodig zijn. Hoewel de helft van de hoofdas 'a_s' in de vergelijkingen voorkomt, staat deze niet in het eindresultaat.

In figuur 4.2 wordt het eindresultaat weergegeven. De excentrische afwijking E kan men vinden door eerst een lijn door de satelliet te tekenen die loodrecht staat op de hoofdas van de cirkel/ellips. Deze lijn zal zowel de cirkel als de ellips snijden (de respectievelijke lijnstukken zijn QP, voor de cirkel, en SP, voor de ellips). De hoek E wordt weergegeven door $Q\hat{O}P$. Deze hoek is noodzakelijk om de werkelijke afwijking, oftewel de *True anomaly* 'ν', te bereken. Op de figuur geeft ν de hoek aan tussen de satelliet en het brandpunt van de ellips, namelijk $S\hat{F}V$.



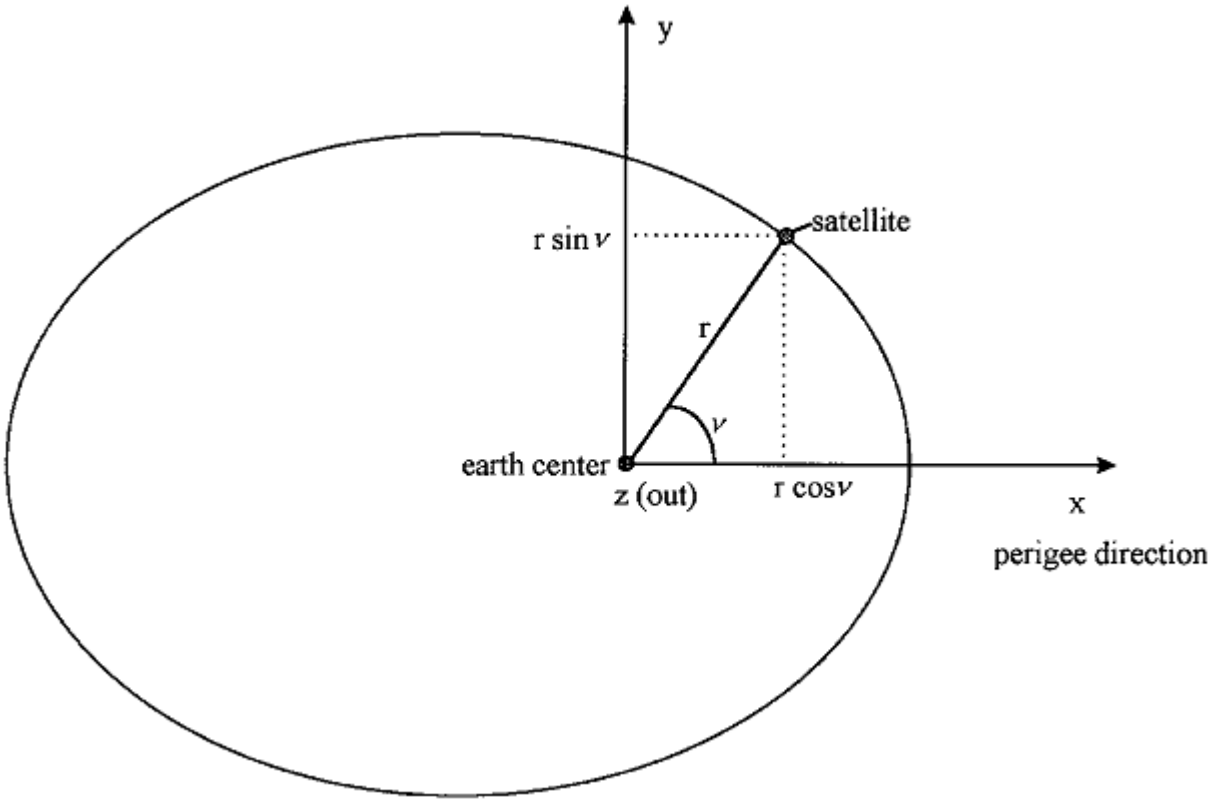
Figuur 4.2: Grafische voorstelling van de *true anomaly*. [1]

De afstand van de satelliet tot het centrum van de aarde wordt gegeven door onderstaande vergelijking en wordt weergegeven in figuur 4.3

$$r = \frac{a_s(1 - e_s^2)}{1 + e_s \cdot \cos(\nu)} \tag{4.11}$$

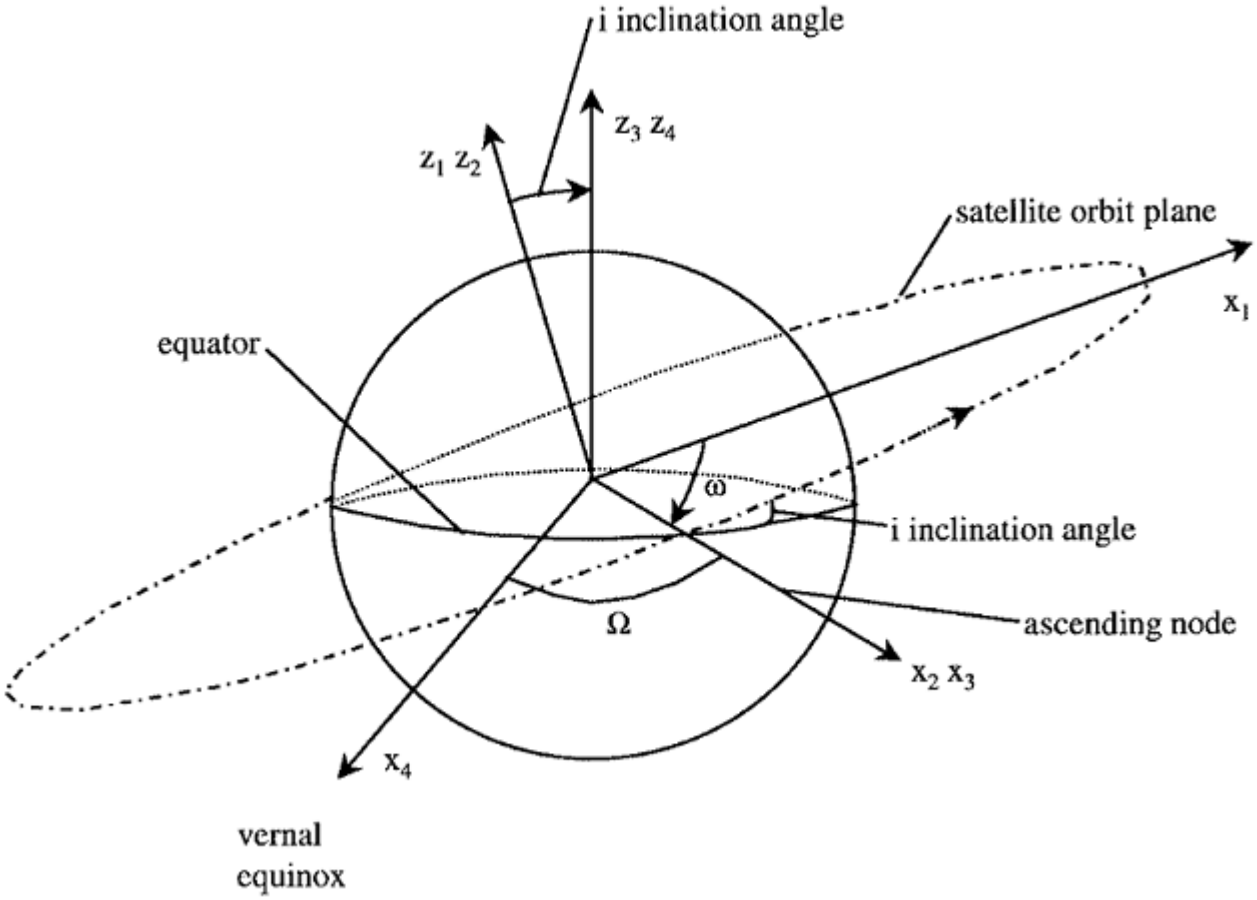
Men kan nu de positie van deze satelliet als volgt vinden:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos(\nu) \\ r\sin(\nu) \\ 0 \end{bmatrix} \tag{4.12}$$



Figuur 4.3: Satelliet in een orbit. [1]

Vergelijking 4.2 geeft de positie van de satelliet weer in een referentiestelsel ten opzichte van het centrum van de aarde. De gebruiker wilt zijn positie kennen in een meedraaiend assenstelsel, zodanig dat eenzelfde plaats dan ook steeds dezelfde coördinaten heeft. Figuur 4.4 maakt dit duidelijk.



Figuur 4.4: De evenaar en de baan van de satelliet. [1]

De matrix die gebruikt wordt om deze correctie in rekening te brengen wordt als volgt gegeven.

$$C_1^2 = \begin{bmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

In deze matrix is ω te vinden in de navigatieboodschap van desbetreffende satelliet. De volgende stap die moet gebeuren is de transformatie naar het evenaarsvlak. Bovenstaande vergelijking beschrijft enkel de baan van de satelliet, maar dus niet in functie van de hellingshoek i.

De hellingshoek is de hoek tussen de satellietbaan en de evenaar en deze wordt ook doorgestuurd door de satelliet. De transformatiematrix in functie van de hellingshoek wordt als volgt meegegeven.

$$C_2^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & -\sin(i) \\ 0 & \sin(i) & \cos(i) \end{bmatrix} \tag{4.14}$$

De hellingshoek i is ook hier in de negatieve richting. De laatste stap is het rekening houden met de aardrotatie. De aardrotatie is $\dot{\Omega_{ie}}$ en er wordt een tijd t_{er} gedefiniëerd zodanig dat als $t_{er} = 0$ de meridiaan van Greenwich zich op één lijn plaatst met de *vernal equinox*. De *vernal equinox* is een vlak waarin de baan van de aarde en de zon zich in bevinden met de zon als brandpunt. De transformatiematrix die de aardrotatie corrigeert wordt nu als volgt gebruikt:

$$C_3^4 = \begin{bmatrix} \cos(\Omega_{er}) & -\sin(\Omega_{er}) & 0 \\ \sin(\Omega_{er}) & \cos(\Omega_{er}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

In bovenstaande vergelijking is Ω_{er} de hoek gevormd door de satelliet en de Greenwich meridiaan.

Op dit moment zijn nu alle transformatie matrices gekend en moet men ze enkel nog vermenigvuldigen. Dit wordt door onderstaande vergelijking weergegeven.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = C_3^4 C_2^3 C_1^2 \begin{bmatrix} r\cos(\nu) \\ r\sin(\nu) \\ 0 \end{bmatrix} \tag{4.16}$$

Indien men C_3^4, C_2^3 en C_1^2 nu invult, dan bekomt men de positie van de satelliet, het eindresultaat wordt hieronder weergegeven.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r \cdot \cos(\Omega_{er}) \cdot \cos(\nu + \omega) - r \cdot \sin(\Omega_{er}) \cdot \cos(i) \cdot \sin(\nu + \omega) \\ r \cdot \sin(\Omega_{er}) \cdot \cos(\nu + \omega) + r \cdot \cos(\Omega_{er}) \cdot \cos(i) \cdot \sin(\nu + \omega) \\ r \cdot \sin(i) \cdot \sin(\nu + \omega) \end{bmatrix} \tag{4.17}$$

4.3 Relatieve looptijden

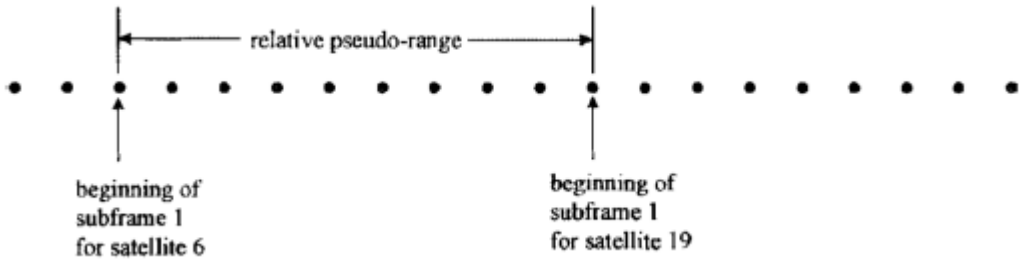
In deze paragraaf wordt er meer de nadruk gelegd op de looptijden van gps signalen, meer bepaald de relatieve looptijden. Dit is een belangrijk begrip binnen deze thesis aangezien er in het algoritme (sectie 6.3) veel gebruikt wordt gemaakt van verschillende soorten tijden.

Het eerste gegeven wat men moet weten is dat iedere GPS satelliet op hetzelfde ogenblik hetzelfde subframes uitzendt. De satellieten zijn voorzien van erg nauwkeurige atoomklokken, dus is het mogelijk om deze op *exact* hetzelfde moment te versturen. Desondanks dat alle satellieten op hetzelfde tijdstip de subframes doorsturen, zullen deze toch op verschillende tijden aankomen. Dit komt omdat de satellieten op verschillende afstanden van de ontvanger zijn, en dus hebben de elektromagnetische golven meer tijd nodig om een grotere afstand te overbruggen. Men spreekt over relatieve looptijden.

Het gemeten verschil tussen de onderlinge satellieten bedraagt maximaal 19 ms, dus indien men de relatieve looptijden gaat opmeten en de verschillen situeren zich rond de 19 ms, dan kan men er zeker van zijn dat de subframes op *dezelfde* tijd zijn verzonden.

Het opmeten van deze relatieve looptijden is niet zo moeilijk. Concreet komt dit neer op het bijhouden van het tijdstip wanneer subframe één van twee verschillende satellieten is binnengekomen, en vervolgens deze tijdstippen van elkaar aftrekken. Aan de ontvangstkant is dit mogelijk door gebruik te maken van de samplefrequentie van de gebruikte klok. Figuur 4.5 maakt dit duidelijk.

In deze figuur wordt de opeenvolging van databits voorgesteld. Het eerste wat men ontvangt is dus de meest *rechtse* databit. Vervolgens ontvangt men subframe 1 van satelliet 19. Op dit moment telt men hoe lang het duurt vooraleer men opnieuw het eerste subframe tegenkomt. Dit komt 9 bits later. Men kan nu deze 9 bits vermenigvuldigen met de tijd tussen 2 opeenvolgende bits en dan heeft men de relatieve pseudorange.



Figuur 4.5: Voorstelling van de relatieve pseudorange. 1

4.3.1 Berekenen van de pseudoranges

Zoals besproken in paragraaf 4.3 heeft iedere satelliet een unieke relatieve looptijd. De afstand kan gegeven worden door vergelijking 4.18.

$$\rho_{iT} = c \cdot (t_u - t_{si}) \tag{4.18}$$

In deze vergelijking is c de snelheid van het licht, t_u de *true time of reception*, t_{si} de *true time of transmission* en ρ_{iT} de *true value of pseudorange from user to satellite i*. Via deze vergelijking is het mogelijk om perfect de afstand te berekenen van user tot satelliet. Maar in praktijk is dit moeilijker, omdat zowel de user als satelliet klok onderhevig zijn aan klokerrors. Deze worden als volgt weergegeven.

$$t_{si,actual} = t_{si} + \Delta b_i$$
$$t_{u,actual} = t_u + b_{ut}$$

Waarbij $t_{si,actual}$ en $t_{u,actual}$ de actuele tijden zijn, die gecorrigeerd zijn door de satelliet clock error Δb_i en de user clock bias error b_{ut}.

Er zijn nog andere factoren die de pseudorange ρ_i beïnvloeden. Deze worden weergegeven door onderstaande vergelijking.

$$\rho_i = \rho_{iT} + \Delta D_i - c \cdot (\Delta b_i - b_{ut}) + c \cdot (\Delta T_i + \Delta I_i + \nu_i + \Delta\nu_i) \tag{4.20}$$

Waarbij ΔD_i de satelliet positie error op de afstand is, ΔT_i de *tropospheric delay error*, ΔI_i de *ionospheric delay error*, ν_i de *receiver measurement noise error* en $\Delta\nu_i$ de *relativistic time correction*. Enkele van deze errors kan men corrigeren. Zo kan men een model opstellen van de troposfeer en kan men de ionosfeer corrigeren a.d.h.v. een *two-frequency receiver*. Hoewel men deze correcties kan toevoegen, zal men nooit de klok van de user kunnen corrigeren, en zal deze dus steeds een onbekende blijven. Concreet wilt dit zeggen dat de pseudoranges als volgt moeten worden genoteerd:

$$\rho_1 = \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} + b_u, \tag{4.21a}$$
$$\rho_2 = \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} + b_u, \tag{4.21b}$$
$$\rho_3 = \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} + b_u, \tag{4.21c}$$
$$\rho_4 = \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} + b_u \tag{4.21d}$$

Het zijn deze vergelijkingen die opgelost moeten worden om de userpositie te vinden. x_i, y_i en z_i met $i = 1, 2, 3, 4$ zijn de satellietposities die gevonden zijn uit vergelijking 4.17. ρ_i en b_u zijn respectievelijk de pseudoranges tot de desbetreffende satelliet en de klok fout.

4.4 Berekenen van de userpositie

Uit vergelijkingen 4.21 worden de verschillende pseudoranges berekend van de verschillende satellieten. Deze vergelijkingen worden als volgt opgelost. Voor een gedetailleerde beschrijving van de oplossing van dit stelsel wordt er gerefereerd naar [1].

De eerste stap die ondernomen wordt, is het *differentiëren* van 4.21. Dit wordt gedaan omdat de vergelijkingen niet lineair zijn. Het resultaat wordt door vergelijking 4.22 weergegeven.

$$\delta\rho_i = \frac{(x_i - x_u)^2 \cdot \delta x_u + (y_i - y_u)^2 \cdot \delta y_u + (z_i - z_u)^2 \cdot \delta z_u}{\sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2}} + \delta b_u \tag{4.22a}$$

$$\delta\rho_i = \frac{(x_i - x_u)^2 \cdot \delta x_u + (y_i - y_u)^2 \cdot \delta y_u + (z_i - z_u)^2 \cdot \delta z_u}{\rho_i - b_u} \tag{4.22b}$$

Het oplossen van deze vergelijkingen wordt gedaan door ze eerst in een matrixvorm te schrijven. Deze wordt als volgt weergegeven.

$$\begin{bmatrix} \delta\rho_1 \\ \delta\rho_2 \\ \delta\rho_3 \\ \vdots \\ \delta\rho_n \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & 1 \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & 1 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \alpha_{n3} & 1 \end{bmatrix} \cdot \begin{bmatrix} \delta x_u \\ \delta y_u \\ \delta z_u \\ \delta b_u \end{bmatrix} \tag{4.23}$$

In vergelijking 4.23 zijn $\delta x_u, \delta y_u, \delta x_u$ en δb_u de enige onbekenden. De waarden van α_{nm} zijn wel gekend, deze zijn als volgt.

$$\begin{aligned} \alpha_{n1} &= \tfrac{x_i - x_u}{\rho_i - b_u} \\ \alpha_{n2} &= \tfrac{y_i - y_u}{\rho_i - b_u} \\ \alpha_{n3} &= \tfrac{z_i - z_u}{\rho_i - b_u} \end{aligned}$$

Gemakkelijkheidshalve wordt vergelijking 4.23 op een andere manier geschreven. Namelijk zoals weergegeven in vergelijking 4.25.

$$\delta\rho = \alpha \cdot \delta x \tag{4.25}$$

De waarden van de parameters zijn als volgt.

$$\delta\rho = \begin{bmatrix} \delta\rho_1 & \delta\rho_2 & \delta\rho_3 & \cdots & \delta\rho_n \end{bmatrix}^T \tag{4.26a}$$

$$\delta x = \begin{bmatrix} \delta x_u & \delta y_u & \delta z_u & \delta b_u \end{bmatrix}^T \tag{4.26b}$$

$$\alpha = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & 1 \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & 1 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \alpha_{n3} & 1 \end{bmatrix} \tag{4.26c}$$

De laatste omvorming is als volgt:

$$\delta x = \begin{bmatrix} \alpha^T \cdot \alpha \end{bmatrix}^{-1} \cdot \alpha^T \cdot \delta\rho \tag{4.27}$$

Het is deze vergelijking (4.27) die moet worden opgelost om de userpositie te vinden. Het uitwerken van deze vergelijking is echter een iteratief proces en neemt dus enige tijd in beslag.

Hieronder wordt het stappenplan van dit iteratief proces weergegeven.

1. Aanwijzing van x_{u0}, y_{u0}, z_{u0} en b_{u0}. De initiële waarden van deze parameters worden nu op 0 gezet. Grafisch gezien wil dit zeggen dat de userpositie zich in het midden van aarde bevindt en dat er geen klok bias is.

2. Berekening van de pseudoranges (ρ_i), door vergelijking 4.21. Deze verschillende pseudoranges zullen verschillen van de berekende waarden. Het verschil hiertussen wordt gegeven door $\delta\rho_i$.

3. De pseudoranges (ρ_i) worden vervolgens gebruikt om de α_{nm} parameters te bepalen.(4.25)

4. Bereken $\delta x_u, \delta y_u, \delta z_u$ en δb_u m.b.v. vergelijking 4.27

5. Hierna kan men een *foutvector*($\delta\nu$) opstellen. De waarde van deze *foutvector* geeft een indicatie hoe nauwkeurig de berekende userpositie is. Deze $\delta\nu$ gaat men dus vergelijken met een op voorhand bekende threshold. Indien de *foutvector* kleiner wordt dan deze threshold is de userpositie nauwkeurig genoeg. De waarde van deze foutvector wordt gegeven door vergelijking 4.28.

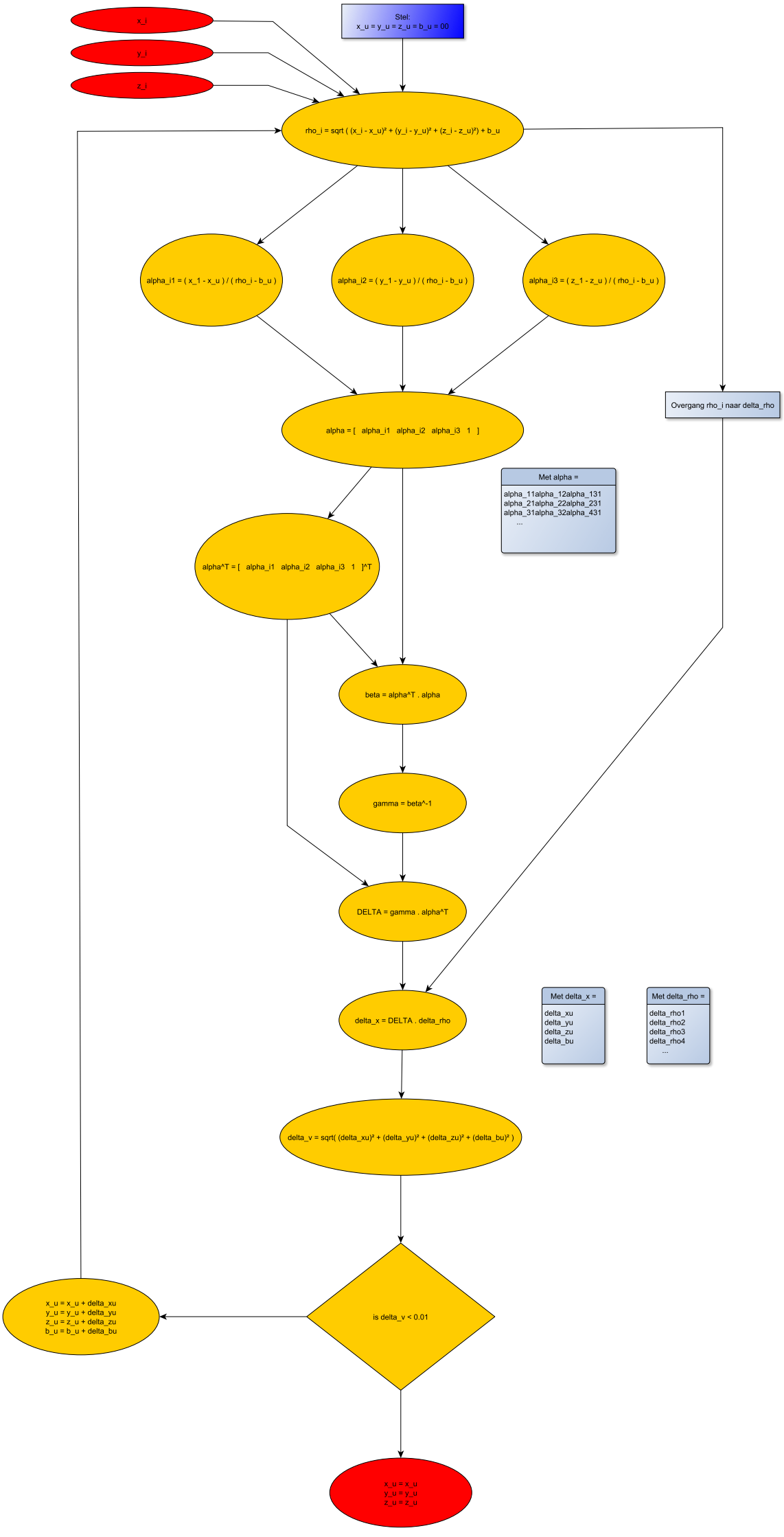
$$\delta\nu = \sqrt{\delta x_u^2 + \delta y_u^2 + \delta z_u^2 + \delta b_u^2} \tag{4.28}$$

6. Tel $\delta x_u, \delta y_u, \delta z_u$ en δb_u op bij de initieel gekozen posities (x_{u0}, y_{u0}, z_{u0} en b_{u0}). De uitkomst hiervan wordt nu weergegeven door x_{u1}, y_{u1}, z_{u1} en b_{u1}. Het zijn deze nieuwe waarden die nu opnieuw in de iteratie gebruikt gaan worden.

7. Herhaal de iteratie van 1 tot en met 6.

Het is de waarde van $\delta\nu$ die een richtlijn geeft wanneer men kan spreken van een correcte userpositie. Wanneer deze waarde kleiner is dan 0,01 heeft men de userpositie beet.[6]

De uitwerking van de userpositie wordt niet in deze thesis verder uitgewerkt. De focus ligt op het ontwerpen van de satellietberekeningsmodule en dus niet op het berekenen van de userpositie. Voor de duidelijkheid wordt hieronder een flowchart weergegeven om de userpositie te berekenen. We benadrukken dat deze bewerkingen NIET geïmplementeerd zijn in deze thesis.


x_i
y_i
z_i
Stel:
x_u = y_u = z_u = b_u = 00
rho_i = sqrt ((x_i - x_u)² + (y_i - y_u)² + (z_i - z_u)²) + b_u
alpha_i1 = (x_1 - x_u) / (rho_i - b_u)
alpha_i2 = (y_1 - y_u) / (rho_i - b_u)
alpha_i3 = (z_1 - z_u) / (rho_i - b_u)
Overgang rho_i naar delta_rho
alpha = [alpha_i1 alpha_i2 alpha_i3 1]
Met alpha =
alpha_11alpha_12alpha_131
alpha_21alpha_22alpha_231
alpha_31alpha_32alpha_431
...
alpha^T = [alpha_i1 alpha_i2 alpha_i3 1]^T
beta = alpha^T . alpha
gamma = beta^-1
DELTA = gamma . alpha^T
delta_x = DELTA . delta_rho
Met delta_x =
delta_xu
delta_yu
delta_zu
delta_bu
Met delta_rho =
delta_rho1
delta_rho2
delta_rho3
delta_rho4
...
delta_v = sqrt((delta_xu)² + (delta_yu)² + (delta_zu)² + (delta_bu)²)
is delta_v < 0.01
x_u = x_u + delta_xu
y_u = y_u + delta_yu
z_u = z_u + delta_zu
b_u = b_u + delta_bu
x_u = x_u
y_u = y_u
z_u = z_u

Hoofdstuk 5

Acquisitie, tracking en demodulatie

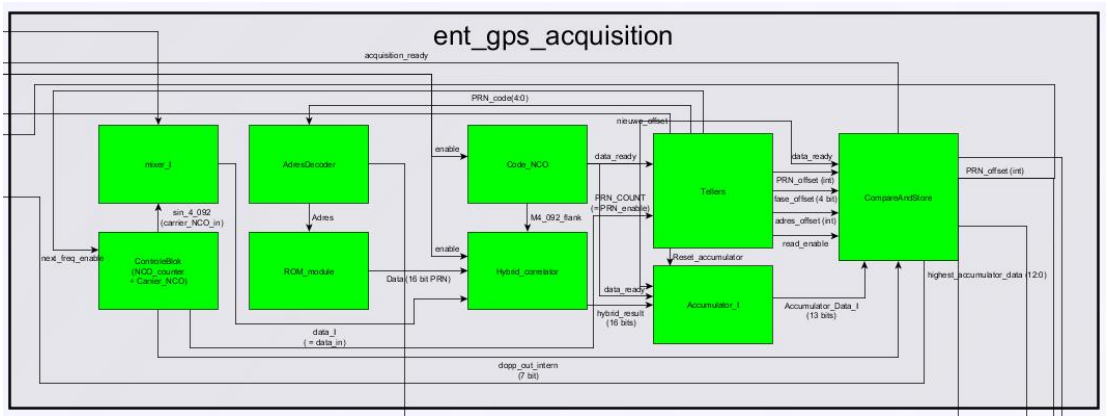
Het onderzoek is een project dat reeds vier jaar bezig is. Uiteraard heeft men in deze vier jaar al grote stappen gezet naar het eindresultaat. In dit hoofdstuk wordt kort de vooruitgang neergeschreven van de voorbije vier jaar, merk op dat onze bijdrage pas in de volgende hoofdstukken gedetailleerd wordt neergeschreven. Voor meer informatie wordt doorverwezen naar voorgaande thesissen.

5.1 Acquisitie

De eerste module die gemaakt werd, is de acquisitie module. Deze staat in voor het opsporen van een satelliet en moet er ook voor zorgen dat de juiste satelliet wordt opgespoord. Deze is tot stand gebracht door Raf Martino en Ben Willems. [6] Vervolgens zijn er enkele aanpassingen aan gemaakt, en is het eindresultaat vastgelegd door Jimmy Gysens. [4]

In onderstaande figuur wordt het blokschema weergegeven van de implementatie door Jimmy Gysens. [4] De gebruikte componenten zijn als volgt:

- o mixer
- o adresdecoder
- o code NCO
- o tellers
- o compare and store
- o controleblok
- o ROM-Module
- o hybride correlator
- o accumulator



Figuur 5.1: De gebruikte acquisitie module. [4]

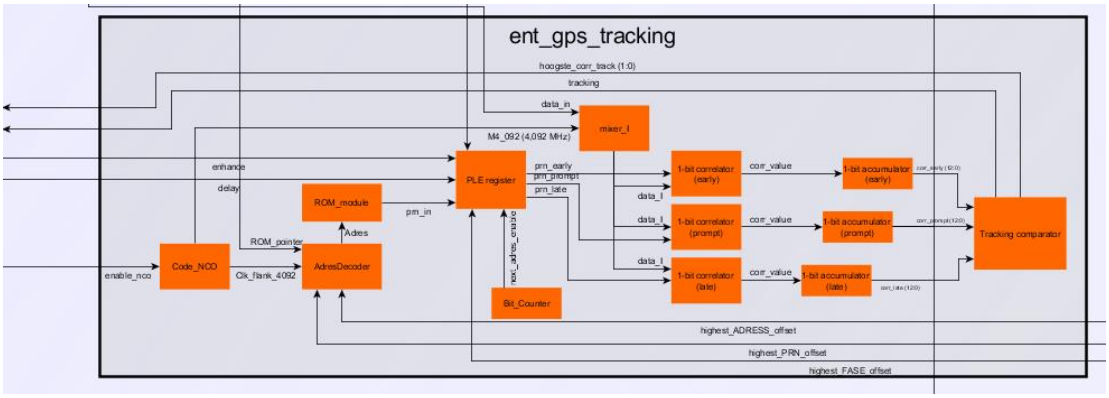
Deze module heeft als doel om na te gaan welke PRN-code overeenkomt met welke satelliet. Aangezien deze codes uniek zijn kan de acquisitie module dus slechts gebruikt worden om maar één satelliet tegelijk te zoeken. Voor een meer gedetailleerde uitleg wordt doorverwezen naar de thesis van Raf Martino en Ben Willems. [6]

5.2 Tracking

Na de acquisitie is het de bedoeling dat de satelliet gevolgd wordt. Daarom is de tracking module ontworpen. Deze module is initieel ontworpen door Raf Martino en Ben Willems [6] maar is doorheen de studie meermaals veranderd geweest. Deze veranderingen worden uitvoerig besproken in de thesissen van Ruben Smeets en Joris Volders [5] en Jimmy Gysens [4].

Het eindresultaat is vastgelegd door Jimmy Gysens [4] en de tracking module was voorzien van meerdere kanalen. Concreet wil dit zeggen dat er dus meerdere satellieten tegelijk kunnen worden gevolgd.

Wederom verwijzen we voor meer informatie door naar de desbetreffende thesissen.



Figuur 5.2: De gebruikte tracking module. [4]

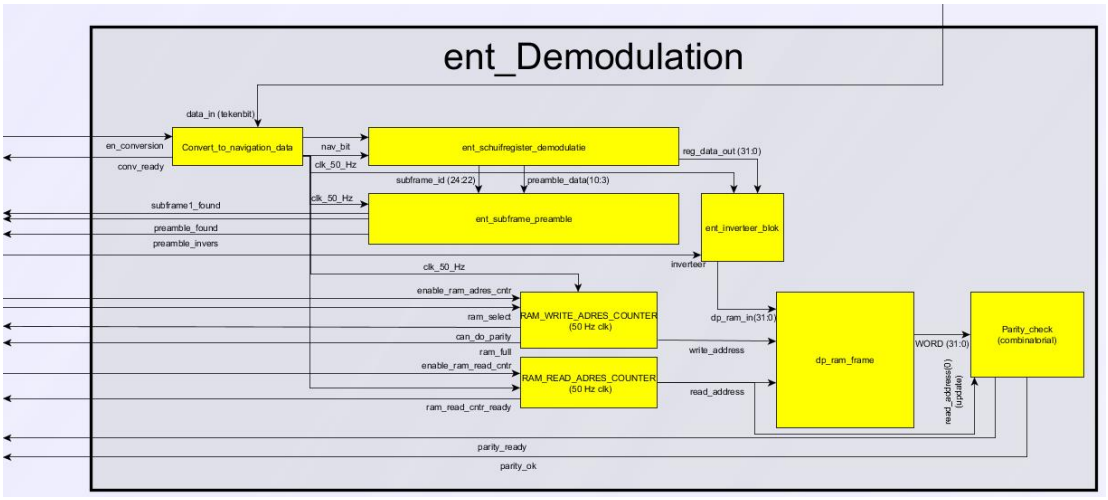
De gebruikte componenten zijn als volgt:

- o code NCO
- o adresdecoder
- o mixer
- o 3 x 1-bit correlatoren
- o tracking comperator
- o ROM-module
- o bit-teller
- o PLE-register
- o 3 x 1-bit accumulatoren

5.3 Demodulatie

De laatste stap die aan het eind van vorig academiejaar bereikt was, was de implementatie van de demodulatie module. Deze is volledig ontwikkeld door Jimmy Gysens. [4]

Deze module had als doel om het ontvangen signaal te demoduleren naar bruikbare data. Het is deze data die nodig is om uiteindelijk aan positiebepaling te doen. Onderstaande figuur is het hardwareschema van de module. Merk op dat aangezien er meerdere kanalen beschikbaar zijn, er dus voor ieder kanaal een demodulatie unit moet zijn.



Figuur 5.3: De gebruikte Demodulatie module. [4]

De gebruikte componenten zijn als volgt en voor meer informatie wordt doorverwezen naar de thesis van Jimmy Gysens. [4]

- o omzetten naar navigatiebits
- o subframe search
- o Ram adres counter
- o pariteitscontrole
- o 32-bit schuifregister
- o inverteerblok
- o Dual port ram

5.3.1 Aanpassingen

In de situatie zoals het nu is, wordt de navigatiedata opgeslagen in het RAM-geheugen. De eerste opdracht is dus om deze data hieruit te halen en vervolgens deze om te zetten naar een floating point notatie.

Dit is opgelost door een OR-poort toe te voegen voor de adres read counter, op deze manier kan men eenvoudig kiezen om gegevens uit het RAM geheugen te halen.

Hoofdstuk 6

Implementatie van de satellietpositieberekeningsmodule

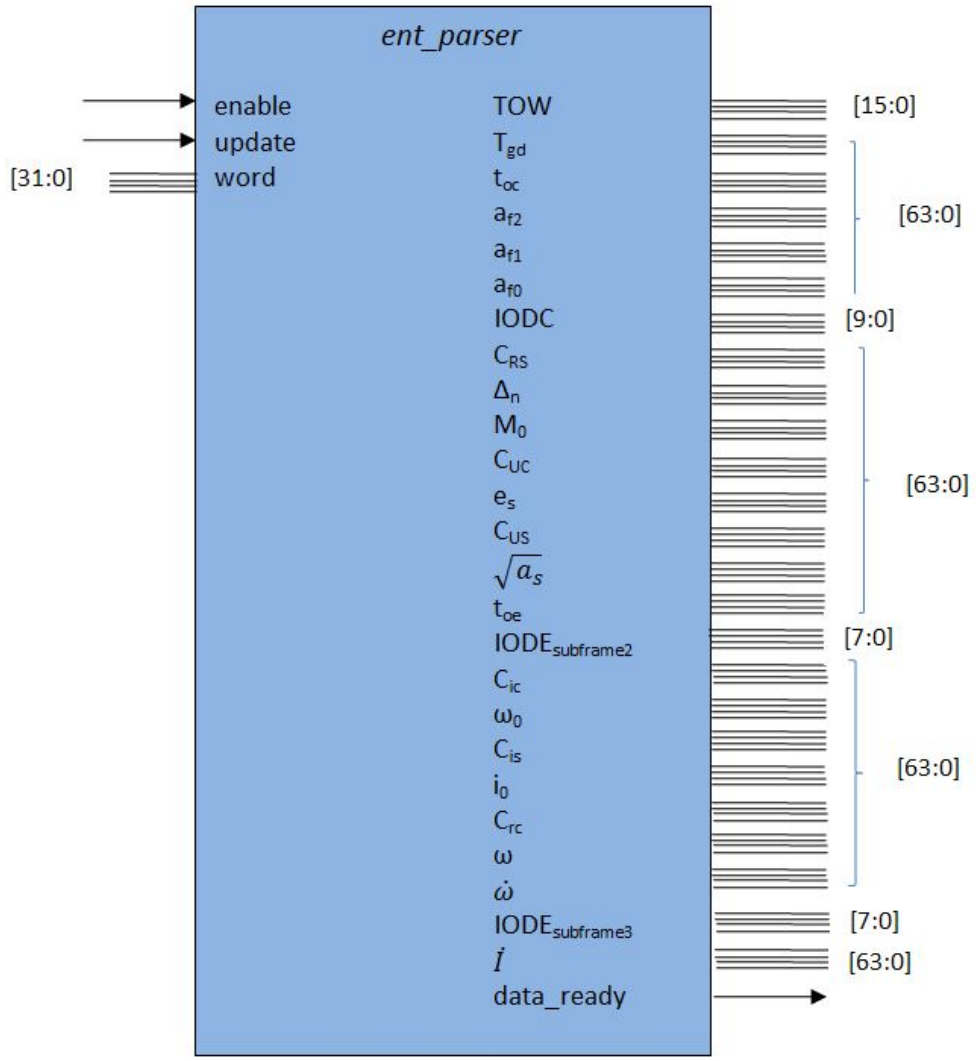
Uit de vorige hoofdstukken is gebleken dat er al erg veel werk verricht is om aan positiebepaling te doen, zo worden de satellieten al opgespoord, getraceerd en kan men de navigatiedata uitlezen. In dit hoofdstuk wordt meer aandacht besteed aan deze navigatiedata, meer bepaald wat deze data precies voorstelt, wat er met deze data precies gedaan wordt en ook een toelichting waarom bepaalde beslissingen zijn genomen.

Zoals besproken in hoofdstuk 3 is de dataset een aaneenschakeling van allemaal bits die op hun beurt verdeeld zijn in frames en subframes. Het eerste wat bereikt werd in deze thesis is de nodige parameters (zie figuur 3.3, 3.4 en 3.5) in variabelen steken.

6.1 Parser

Het onderverdelen van deze parameters in de juiste variabelen wordt gedaan in de parser entiteit. De FSM rond de parser stuurt de adres-teller van de demodulatie-unit aan en vervolgens ontvangt de parser het juiste woord (32 bits lang dus).

Initieel was de parser een aparte bouwblok in de demodulatie-unit, en dit volgens de gedachte dat deze parser de logische opvolging was na het verkrijgen van de navigatiedata. Maar dit bracht ook een nadeel met zich mee, zo moesten er grote aanpassingen gebeuren aan de code van Jimmy Gysens. Daarom is er uiteindelijk toch gekozen om een nieuwe module in te richten.



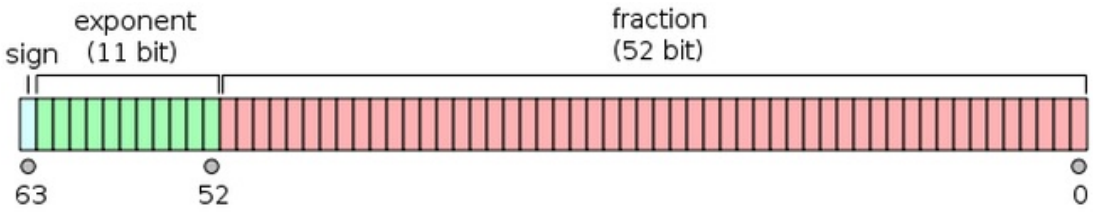
Figuur 6.1: De geïmplementeerde parser.

De eerste bewerking die moet worden uitgevoerd is het verwerken van de two's complement notatie. Dit wordt ook in de parser gedaan.

Daarnaast hebben alle variabelen ook vaste schaalfactoren die uniek zijn. Het is dus de bedoeling dat deze mee verwerkt worden in de desbetreffende parameter. Voor deze verwerking is de keuze gemaakt om alle parameters om te zetten naar de IEEE 754 double-precision binary floating-point format [10]. Dit formaat is noodzakelijk omdat nauwkeurigheid sleutel is bij GPS systemen. Voor deze nauwkeurigheid wordt er natuurlijk aan plaats ingeboet.

In de thesis van Martino en Willems [6] wordt de nauwkeurigheid van een 32 bit implementatie vergeleken met een 64 bit implementatie. Aan de hand van hun bevindingen ([6], pagina 97) kan men concluderen dat een 64 bit implementatie inderdaad nauwkeuriger is. Single precision is slechts nauwkeurig tot op 1,5m, waarbij double precision tot op $0,00005m$ nauwkeurig is, steeds gemeten t.o.v. de user positie.

Concreet wilt dit zeggen dat na deze omzetting alle parameters 64 bits lang zijn, waarbij de 1e bit een tekenbit is, de volgende 11 zijn de exponentbits en de laatste 52 de fractionbits. Een voorstelling wordt gegeven in figuur 6.2.



Figuur 6.2: Voorstelling van de IEEE 754 standaard. Referentie: wikipedia.org

De schalingsfactoren moeten ook in rekening gebracht worden om de juiste waarden van de parameters te hebben. In onderstaande code wordt weergegeven hoe het algoritme te werk gaat om de juiste schalingsfactor te integreren. Nota: in C_RS_na_word_signaal_controle zitten alle bits die uit het woord komen, reeds in one's complement notatie. In dit voorbeeld dus de bits van C_{RS}.

```
1  --C_RS is 16 bits lang ( dus van 14 naar 0  want de eerste bit is de tekenbit !)
2  -- -5 is schaalfactor!!
3  for i in 14 downto 0 loop
4          if (C_RS_na_word_signaal_controle(i) = '1') then
5                  C_RS( 63 downto  (52-i) ) <= C_RS_na_word_signaal_controle(15) &,
6                  (std_logic_vector(to_signed(1023 + (i-5), 11))) &,
7                  C_RS_na_word_signaal_controle((i-1) downto 0);
8                  C_RS ( 51-i downto 0 ) <= (others => '0');
9                  exit;
10         else
11         end if;
12 end loop;
```

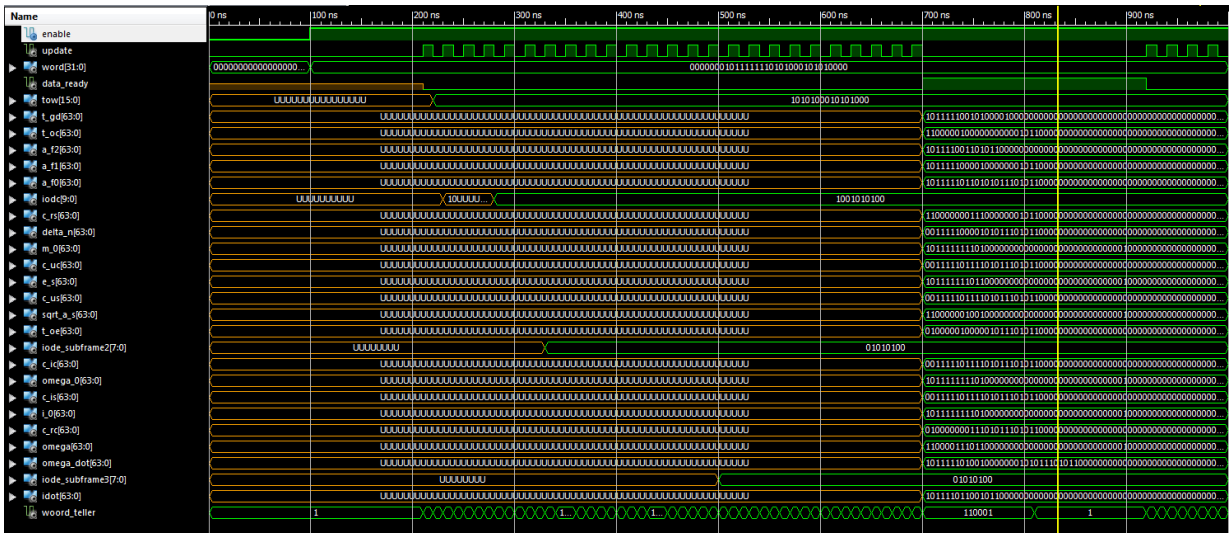
Het eerste wat gebeurt, is het toewijzen van de tekenbit en dit gebeurt in regel 5. De *for loop* zorgt ervoor dat alle bits van het woord worden overlopen. Aangezien in dit voorbeeld C_{RS} 16 bits lang is gaat de lus van 14 downto 0, omdat de eerste bit een tekenbit is.

Vervolgens worden de 11 exponentbits toegevoegd. Deze zijn afhankelijk van de schaalfactor (in dit voorbeeld is de schaalfactor gelijk aan -5) en ook afhankelijk van het formaat van de vector aangezien wij gekozen hebben om het formaat op 64 te zetten (aangezien double precision verkozen is). Er zijn 11 exponentbits, dit komt overeen met $2^{11} = 2048$ waarden. Getallen groter dan één hebben een exponent die groter is dan $1024_{10} = 10000000000_2$ en getallen die kleiner zijn dan één hebben een exponent die kleiner is dan $1023_{10} = 01111111111_2$. Door deze exponentbits toe te kennen, zorgt dit ervoor dat de IEEE-754 standaard wordt nagestreefd. Het is om deze reden dat men de schaalfactor hiervan moet aftrekken. Deze bewerking volgt in regel 6.

Daarna worden de resterende bits van het woord op de juiste plaats gezet, en dit wordt gedaan in regel 7. Ten slotte moet de lengte van de vector 64 bedragen, dus moeten er nog nullen aan toegevoegd worden, vandaar regel 8.

De werking van de parser wordt ook duidelijk aan de hand van de testbench (figuur 6.8). Ook is de parser voorzien van een enable en een update. De update moet getoggled worden om een nieuw woord in te lezen en tijdens deze toggle worden de juiste bits van het woord aan de juiste variabele toegewezen.

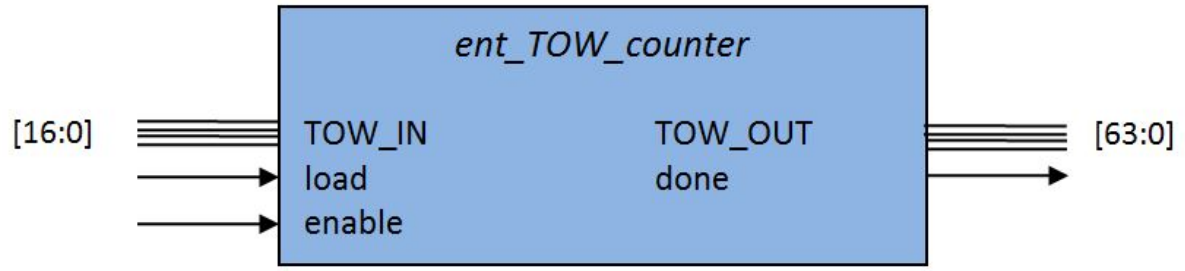
Op dit punt zijn dus alle woorden uitgelezen uit het RAM-geheugen, worden alle bits toegekend aan de juiste variabelen en worden deze in het IEEE 754 formaat omgezet. De volgende stap is het aanmaken van de operatormodules. Vooraleer dit gedaan kan worden, wordt er eerst grondig bekeken welke operatoren er nodig zijn en hoe deze efficiënt geïmplementeerd kunnen worden, aangezien er een afweging moet gebeuren tussen snelheid en oppervlakte.



Figuur 6.3: Testbench van de parser.

6.2 TOW - counter

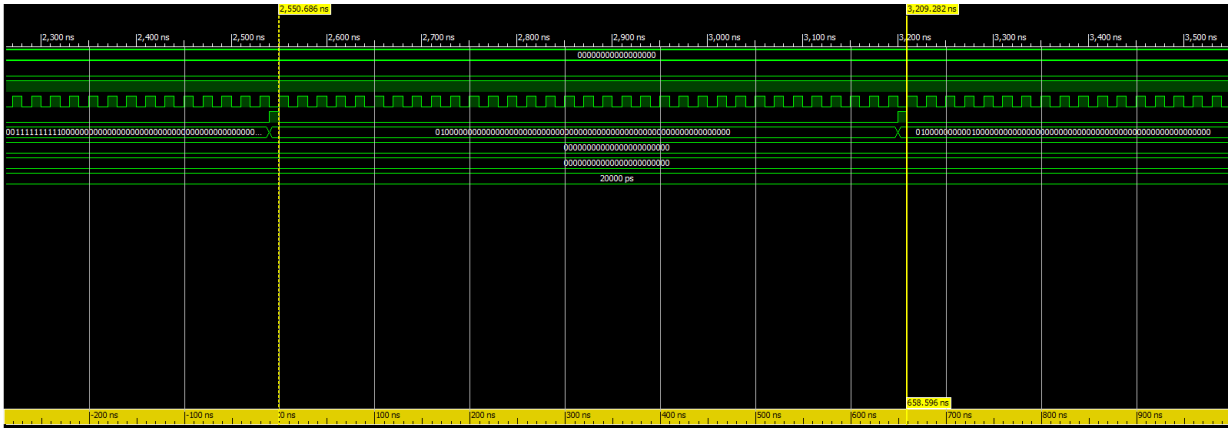
De *Time of week* is een parameter die aangeeft wanneer de satelliet precies zijn data heeft verzonden, zoals besproken in sectie 3.2 staat deze in het begin van ieder subframe. Om deze variabele up to date te houden kiezen we ervoor om een teller te maken die aan de hand van de user klok bijhoudt wat de echte *time of week* is. Op deze manier is het mogelijk om continu aanpassingen te gaan doen van de berekeningen. De entiteit die hiervoor zorgt is de TOW-counter, deze wordt weergegeven in figuur 6.4.



Figuur 6.4: Voorstelling van de TOW counter.

De TOW-counter leest een 17 bit waarde in die afkomstig is van de parser, de *truncated TOW*. Vervolgens wordt deze vermenigvuldigd met 6 aangezien de least significant bit 6 seconden voorstelt [1]. Daarna wordt deze waarde iedere seconde met één verhoogd en wordt ze omgezet in een 64 bit FPU formaat.

De gebruikte klokfrequentie bedraagt 50 MHz aangezien deze doorheen het hele ontwerp gebruikt wordt. Hierdoor is de TOW-counter ook voorzien van een prescaler, zodanig dat er een klok van 1 Hz kan gemaakt worden. Onderstaande testbench geeft de werking weer van deze teller. Een opmerking hierbij is dat de prescaler gezet is om na 32 klokpulsen al aan te geven dat hij *klaar* is, dit om simulatietijden te beperken.



Figuur 6.5: Testbench van de TOW counter.

6.3 Algoritme van de bewerkingen

Deze paragraaf beperkt zich tot het bepalen en implementeren van de operatoren die allemaal nodig zijn om aan satelliet-positiebepaling te doen, merk op dat hiermee enkel de posities van satelliet bedoeld worden, en dus niet van de gebruiker! De gebruikte bewerkingen steunen volledig op sectie 4.1 en worden hier enkel herhaald, dus niet in detail besproken. Onderstaande bewerkingen zijn dus als het ware het stappenplan dat gevolgd wordt om uiteindelijk tot een satellietpositie te komen.

De eerste bewerking die wordt uitgevoerd, is het berekenen van de *mean motion* 'n'. Deze vinden we als volgt (vergelijking 4.3).

$$n = \sqrt{\frac{\mu}{a_s^3}} + \Delta n \tag{6.1}$$

Zoals reeds vermeld, is $\mu = 9386005 \times 10^8 m^3/s^2$, vindt men a_s in subframe 2, bits 227 - 234 en 241 - 264. Δn bevindt zich in subframe 2, bits 91 - 106.

Zoals reeds besproken in vergelijking 4.5 moet er een correctie gebeuren op de GPS tijd op het moment van transmissie. Deze gebeurt als volgt.

$$\text{als } t_c - t_{oe} > 302400 \text{ dan is } t_c \Rightarrow t_c - 604800$$
$$\text{als } t_c - t_{oe} < -302400 \text{ dan is } t_c \Rightarrow t_c + 604800$$

In deze vergelijkingen kan men t_{oe} vinden in subframe 2, bits 271 - 286, en t_c is besproken in sectie 4.3.

Eens de GPS systeem tijd op het moment van transmissie 't_c' gekend is, kan de gemiddelde afwijking gevonden worden door deze vergelijking te gebruiken, die steunt op vergelijking 4.4.

$$M = M_0 + n \cdot (t_c - t_{oe}) \tag{6.3}$$

In deze vergelijking is M_0 de gemiddelde afwijking en kan men halen uit subframe 2, bits 107 - 114, 121 - 144.

Vervolgens kan men de excentrische afwijking bepalen. Dit wordt bereikt door vergelijking 6.4.

$$E = M + e_s \cdot \sin E \tag{6.4}$$

Waarbij e_s de excentriciteit is van de baan van desbetreffende satelliet en deze wordt geparsed uit subframe 2, bits 167 - 174. Merk op, uit vergelijking 4.7, dat de iteratie enkele keren moet gebeuren alvorens de volgende stap mag worden uitgevoerd. Het aantal keer dat de iteratie uitgevoerd wordt is vijf keer. Dit omdat in voorgaande thesissen ook dit getal gekozen was.

Vervolgens kan de relativistische correctieterm bepaald worden. Dit gebeurt door vergelijking 6.5.

$$\Delta t_r = F e_s \sqrt{a_s} \sin E \tag{6.5}$$

In vergelijking 6.5 is $F = -4.442807633 \times 10^{-10} sec/m^{1/2}$ en zijn de andere parameters reeds gekend. Vervolgens is het mogelijk om de algemene tijdscorrectie factor te berekenen. Deze wordt als volgt berekend.

$$\Delta t = a_{f0} + a_{f1}(t_c - t_{oc}) + a_{f2}(t_c - t_{oc}) + \Delta t_r - T_{GD} \tag{6.6}$$

In vergelijking 6.6 zijn a_{f0}, a_{f1}, a_{f2} en t_{oc} klok correctie parameters. T_{GD} is de verwachte groep delay, en alle parameters vindt men in subframe 2, respectievelijk in bits 271 - 292, 249 - 264, 241 - 248, 219 - 234, 197 - 204. Men kan nu de GPS tijd op het moment van de transmissie 't' (dus NIET de GPS systeem tijd 't_c') berekenen door middel van onderstaande vergelijking.

$$t = t_c - \Delta t \tag{6.7}$$

Het is nu de bedoeling om naast de tijd ook een positie te berekenen. Hiervoor doen we weer beroep op vergelijkingen 6.8, 6.9 en 6.10. Voor de duidelijkheid worden ze nog eens vermeld.

$$\nu_1 = \arccos \frac{cos(E) - e_s}{1 - e_s \cdot cos(E)} \tag{6.8}$$

$$\nu_2 = \arcsin \frac{\sqrt{1 - e_s} \cdot sin(E)}{1 - e_s \cdot cos(E)} \tag{6.9}$$

$$\nu = \nu_1 \cdot sign(\nu_2) \tag{6.10}$$

Indien we ν weten, kunnen we de hoek 'ϕ' bepalen. Deze wordt als volgt weergegeven.

$$\phi \equiv \nu + \omega \tag{6.11}$$

In vergelijking 6.11 is ω de perifocus. Deze bevindt zich in subframe 3, bits 197 - 204 en 211 - 234. Vervolgens moet men onderstaande correctietermen berekenen.

$$\begin{aligned} \delta\phi &= C_{us} \sin(2\phi) + C_{uc} \cos 2\phi \\ \delta r &= C_{rs} \sin(2\phi) + C_{uc} \cos 2\phi \\ \delta i &= C_{is} \sin(2\phi) + C_{ic} \cos 2\phi \end{aligned}$$

In bovenstaande vergelijking zijn $C_{us}, C_{uc}, C_{rs}, C_{rc}, C_{is}$ en C_{ic} te vinden in de navigatieboodschap. Respectievelijk op volgende plaatsen: subframe 2, bits 211 - 226, subframe 2, bits 151 - 166, subframe 2, bits 69 - 84, subframe 3, bits 181 - 196, subframe 3, bits 121 - 126, subframe 3, bits 61 - 76.

De correctietermen moeten als volgt worden toegepast.

$$\begin{aligned} \phi &= \phi + \delta\phi \\ r &= r + \delta r \\ i &= i + \delta + idot(t - t_{oe}) \end{aligned}$$

'idot' is de maat waarmee de hellingsgraad toeneemt en deze vindt men in subframe 3, bits 279 - 292. 't' vindt men via vergelijking 6.7. Vervolgens kunnen we de hoek berekenen tussen de node en de Greenwich meridiaan Ω_{er}. Deze correctietermen kan men vinden via vergelijking 4.15.

$$\Omega_{er} = \Omega_e + \dot{\Omega}(t - t_{oe}) - \dot{\Omega_{ie}}t \tag{6.14}$$

In deze vergelijking stelt $\dot{\Omega_{ie}}$ de hoekrotatie van de aarde voor, zijnde $7.2921151467 \times 10^{-5} rad/sec$. Ω_e en $\dot{\Omega}$ wordt gehaald uit de navigatieboodschap. Ω_e vindt men in subframe 3, bits 77 - 84 en 91 - 114 en $\dot{\Omega}$ vindt men ook in subframe 2, bits 241 - 264.

Op dit moment zijn alle variabelen gekend en kan men de x,y en z positie van satelliet 'i' bepalen.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\cos\Omega_{er}\cos\phi - r\sin\Omega_{er}\cos i\sin\phi \\ r\sin\Omega_{er}\cos\phi + r\cos\Omega_{er}\cos i\sin\phi \\ r\sin i\sin\phi \end{bmatrix}$$

Van deze bewerkingen is een uitgebreide flowchart opgesteld, dit wordt getoond in sectie 6.8.

6.4 Resources voor het algoritme

Na het overlopen van alle bewerkingen valt het op dat we deze kunnen reduceren tot 6 berekeningen namelijk $\pm, \times, /, \sqrt{}$, sin en arcsin. Voor de optelling en aftrekking hebben we ervoor gekozen om één blok te voorzien, voor de vermenigvuldiging ook één blok, voor de deling ook één blok en voor de vierkantswortel ook één blok. Ook is er gekozen om de sinus en de boogsinus te implementeren via een reeksontwikkeling, deze wordt in detail uitgelegd in paragraaf 6.5.

Tabel 6.1 geeft een overzicht weer van de gebruikte operatoren, samen met de ingenomen oppervlakte. Tijdens het ontwerpen van deze blokken is het belangrijk dat er een optimale verhouding is tussen oppervlakte en snelheid. In deze tabel wordt weergegeven hoeveel plaats iedere operator-blok inneemt op de FPGA.

Tabel 6.1: Resources voor de gebruikte operatoren.

	Mode	**# DSP SLICE**	**LUT-FF PAIR**	**LUTs**	**FF**
Multiply	medium usage	9	585	402	555
Add/Substract	Logic	0	1035	1035	1177
Divide	$C_{rate} = 18$	0	648	977	742
Sqrt	$C_{rate} = 54$	0	346	401	386
TOTAL		9	2884	2815	2860
MARGIN		47	10752	21504	21504
REMAINDER		38	7868	18689	18644

In tabel 6.2 wordt er weergegeven hoe lang het duurt om de bewerkingen uit te voeren. Kolom 1 geeft weer hoeveel klokcycli men nodig heeft om één bewerking uit te voeren, kolom 2 geeft weer hoeveel operaties er in het totaal moeten gebeuren van deze bewerking en in kolom 4 wordt de totale tijd nodig om alle bewerkingen uit te voeren weergegeven. Tabel 6.2 wordt berekend aan de hand van een systeemklok van $50MHz$.

Tabel 6.2: Benodigde tijd van de operatoren.

	# Cycli	**# Operaties**	**Totaal aantal klokcycli**	**Totale tijd (ns)**
add	15	14	210	4200
substract	25	17	425	8500
divide	57	5	285	5700
multiply	17	45	765	15300
root	57	2	114	2280
Sin*	557	17	9469	189380
Arcsin*	1320	1	1320	26400
			Som	251760 ns

(*) De sinus en boogsinus worden in besproken in 6.5.

6.5 Sinus en boogsinus

6.5.1 Implementatie van de sinus berekening

Om de sinus van een hoek uit te rekenen is er geen aparte blok geïmplementeerd omdat ervoor gekozen is om dit met een reeksontwikkeling uit te rekenen. Deze reeksontwikkeling bestaat uit vermenigvuldigingen, delingen en optellingen waardoor de bestaande bewerkingsblokken konden gebruikt worden. Ook bij de boogsinus werd er gekozen om met een reeksontwikkeling te werken. De reeksontwikkelingen zijn hieronder weergegeven.

$$\sin z = \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{(2n+1)!} \tag{6.15}$$

$$\arcsin z = \sum_{n=0}^{\infty} \frac{\binom{2n}{n} z^{2n+1}}{4^n(2n+1)} \tag{6.16}$$

Hierbij bepaalt het aantal termen in deze reeksontwikkeling de nauwkeurigheid van het resultaat. Aangezien nauwkeurigheid bij plaatsbepaling essentieël is, is het benodigd aantal termen uitvoerig onderzocht met Matlab. Hierbij werd aan de hand van een dataset uit een Receiver Independent Exchange Format (RINEX) file de positie van de gegeven satelliet bepaald [7]. Eerst werd dit gedaan met de ingebouwde sinus functie van matlab om een referentie te hebben. De afwijking op de sattelietcoördinaten (X, Y en Z) van deze berekening zijn in tabel 6.3 weergegeven.

Tabel 6.3: Sinus reeksontwikkeling vs. matlab implementatie

	x-coördinaat	**y-coördinaat**	**z-coördinaat**
afwijking, 7 termen (m)	−21138.98	95667.2	684.66
afwijking, 8 termen (m)	1522.00	−6942.28	−80.76
afwijking, 9 termen (m)	−77.44	375.61	11.81
afwijking, 10 termen (m)	1.96	−13.27	−1.55
afwijking, 11 termen (m)	0.12	0.00	0.17

Later werd dezelfde berekening herhaald, hierbij werd echter gebruik gemaakt van de sinus op basis van de reeksontwikkeling. Bij een reeksontwikkeling onder de zes termen bleek het resultaat niet significant te zijn.

Om nauwkeuriger te zijn, dient het aantal termen dus verhoogd te worden. Tabel 6.3 geeft de nauwkeurigheid van de reeksontwikkeling weer met meer dan zeven termen. Een reeksontwikkeling met zeven termen bleek niet voldoende te zijn aangezien de gevonden coördinaten tot 96000 meter afweken van de coördinaten gevonden met de Matlab sinus-functie. Aangezien dit een te grote afwijking is zijn er meer termen bijgevoegd om de nauwkeurigheid te verhogen. Bij 8 termen bleek dat de afwijking verkleind werd tot maximum 7000 meter. Pas wanneer er gebruik gemaakt werd van een reeksontwikkeling van 11 termen werd de afwijking kleiner dan 1 meter, met een maximale afwijking van 0,17 meter. Dit is een aanvaardbare afwijking waardoor er besloten werd om onderstaande reeksonwikkeling te gebruiken in de implementatie.

$$\sin z = \sum_{n=0}^{11} \frac{(-1)^n z^{2n+1}}{(2n+1)!} \tag{6.17}$$

In het algoritme moet er ook gebruik gemaakt worden van de *cosinus* operatie. Deze wordt ook aan de hand van de sinus reeksontwikkeling berekend en wel volgens onderstaande transformatie [9].

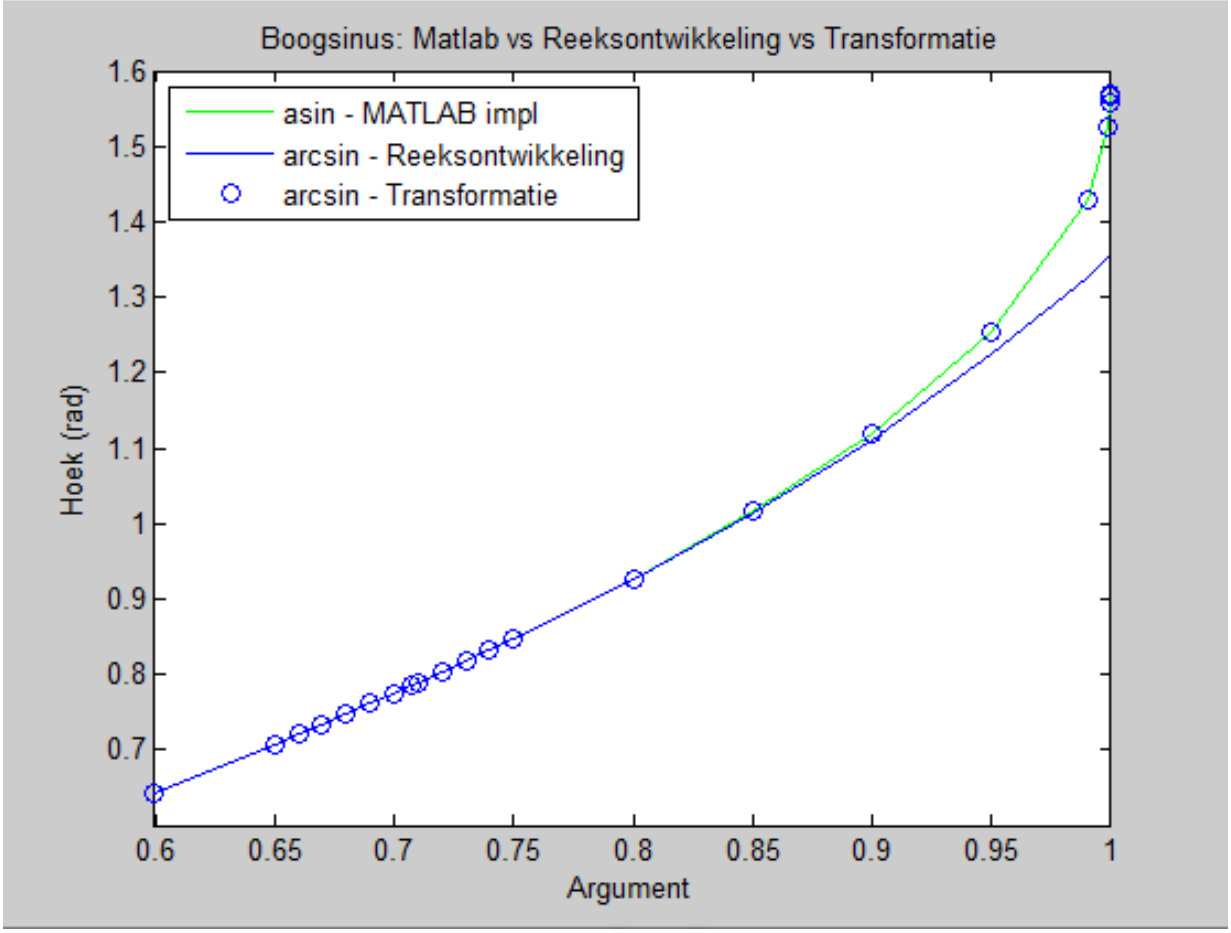
$$\cos z = \sin\left(\frac{\pi}{2} - z\right) \tag{6.18}$$

6.5.2 Implementatie van de boogsinus berekening

Om het aantal termen van de reeksontwikkeling van de boogsinus te bepalen werd dezelfde methode toegepast als voor de sinus reeksontwikkeling. Hierbij steeg de nauwkeurigheid ook naarmate er meer termen werden gebruikt maar was de reeksontwikkeling zeer onnauwkeurig voor waardes van hoeken boven $1/\sqrt{2} = 0,707\ldots\ rad$ zoals te zien in figuur 6.6. De gebruikte waarde, $1/\sqrt{2}$, die gebruikt wordt komt overeen met $7,071067811865475727373109293 69E-1\ rad$.

Dit wordt opgelost door hoeken die groter zijn dan $1/\sqrt{2}\ rad$ om te vormen met onderstaande code, om dan met de omgevormde hoek de boogsinus te bepalen.

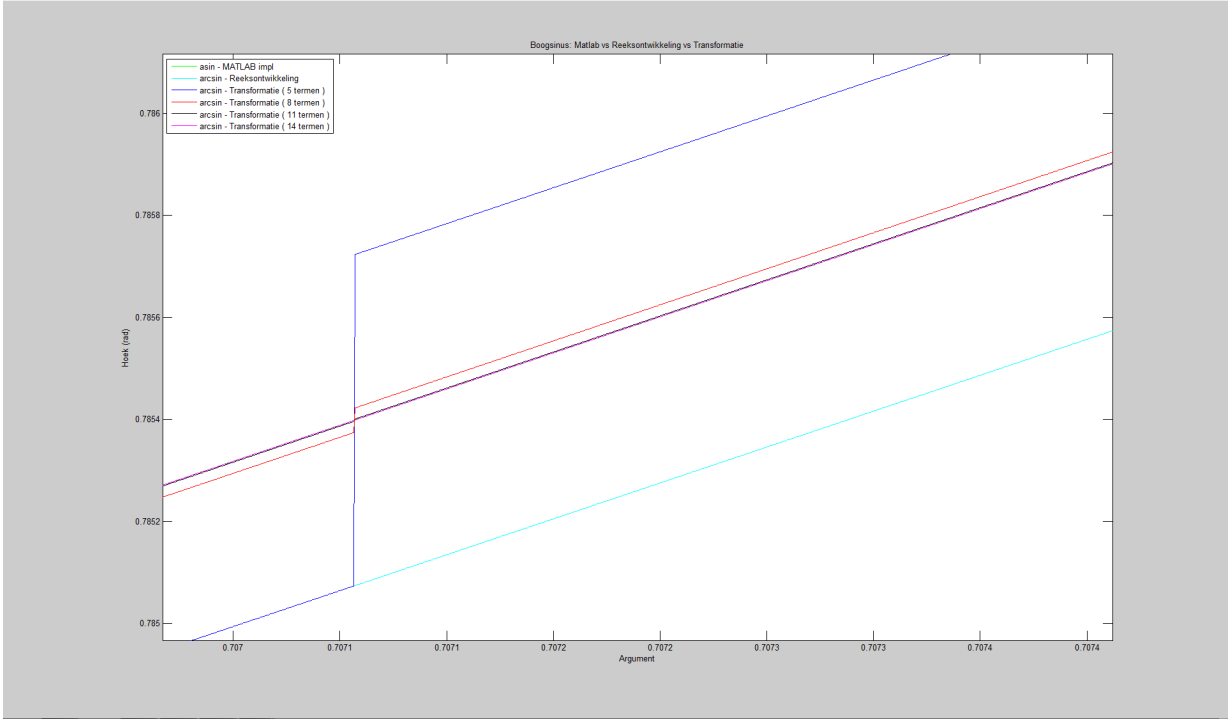
```
1 if(abs(x)  nb > 1/sqrt(2))
2     x_new = sqrt(1-x^2);
3     y_old = arcsin_forloop(x_new,i);
4     y = sign(x)*((pi/2)-y_old);
5 else
6     y = arcsin_forloop(x,i);
7 end
```



Figuur 6.6: Afwijking van gewone reeksontwikkeling boven $1/\sqrt{2}$

Nadat de boogsinus berekend is, wordt de uitkomst afgetrokken van $\pi/2$ en vervolgens vermenigvuldigd met het teken van de oorspronkelijke hoek. Dit zorgt ervoor dat de reeksontwikkeling ook voor hoeken groter dan $1/\sqrt{2}$ *rad* nauwkeurig is, in figuur 6.6 weergegeven door arcsin-transformatie.

Als het aantal termen dan wordt opgedreven, lijkt eerst dat de nauwkeurigheid al voldoende is bij ongeveer 7 termen. Wanneer er echter gekeken wordt naar het interval rond $1/\sqrt{2}$ *rad* blijken de resultaten hier niet aan de vooropgestelde nauwkeurigheid te voldoen zoals te zien in figuur 6.7.



Figuur 6.7: Afwijking op de boogsinus bij een verschillend aantal gebruikte termen

Om de nodige nauwkeurigheid ook rond $1/\sqrt{2}$ *rad* te behalen wordt er gekeken naar hoeveel termen er nodig waren om een nauwkeurig van 10^{-10} *rad* te behalen. Als dit niet gehaald kon worden met behulp van een realistische reeksontwikkeling zou er ook nog gekozen kunnen worden om een Look Up Table (LUT) te gebruiken, in de buurt van $1/\sqrt{2}$ *rad*, en reeksontwikkelingen buiten deze zone. Dit omdat de nauwkeurigheid van de reeksontwikkeling wel al voldoende is bij 7 termen.

Wanneer de reeksontwikkelingen getest worden in Matlab blijkt dat met 21 termen in de reeksontwikkeling de nauwkeurigheid van 10^{-10} behaald kan worden zodat er toch gekozen wordt om het volledig via een reeksontwikkeling te implementeren. De resultaten van deze tests zijn weergegeven in tabel 6.4. Vergelijking 6.19 geeft de gebruikte reeksontwikkeling weer die geïmplementeerd is.

$$\arcsin z = \sum_{n=0}^{21} \frac{\binom{2n}{n} z^{2n+1}}{4^n(2n+1)} \tag{6.19}$$

Tabel 6.4: Afwijking van reeksontwikkeling boogsinus tov. Matlab

Aantal termen	**Afwijking** (rad)
8 termen	$2.3786E-05$
11 termen	$2.0128E-06$
14 termen	$1.8483E-07$
17 termen	$1.7898E-08$
19 termen	$3.8561E-09$
20 termen	$1.7992E-09$
21 termen	$8.4205E-10$

6.5.3 Sinus controle blok

Bij de berekening van de sinus worden de hoeken, waarvan de sinus wordt berekend, steeds teruggebracht naar het interval $[-\pi, \pi]$. Hiervoor is er een aparte hardware bouwblok geïmplementeerd die zal nagaan als een bepaalde hoek groter is dan π of kleiner is dan $-\pi$. Dit zal gebeuren met behulp van onderstaande code.

```
1                if ( A(63) = '0' and B(63) = '0' )  then
2                        sin_controle_valid <= "10";-- - 2*pi doen!
3
4                elsif ( A(63) = '1' and B(63) = '1' ) then
5                        sin_controle_valid <= "01";-- + 2*pi doen!
6
7                else
8                        sin_controle_valid <= "00";--voer de normale taylor-reeks uit!
9                end if;
```

Vooraleer de sinus_controle_blok in werking treedt, zal de ALU FSM eerst de waarden van de te berekenen hoek + en - π doen. Deze twee waardes zullen aan de ingang van de sinus_controle_blok worden klaar gezet, waarna de blok wordt ge-enabled door de FSM. Hierna zal de sinus_controle_blok aan de hand van de tekenbits van de 2 inputwaardes bepalen als de hoek in het vooropgestelde interval ligt of niet.

De tekenbits worden verkregen door telkens de Most Significant Bits (MSB) van de twee ingangswaardes te nemen zijnde A(63) en B(63). Als de tekenbits beide '0' zijn wil dit zeggen dat de oorspronkelijke hoek groter was dan π. sin_controle_valid wordt dan op "10" gezet zodat de FSM weet dat er 2π van de hoek moet worden afgetrokken.

Wanneer beide tekenbits '1' zijn was de oorspronkelijke hoek kleiner dan $-\pi$ en wordt sin_controle_valid op "01" gezet.

De laatste mogelijkheid is dat de tekenbits van elkaar verschillen. Dit wil zeggen dat de hoek tussen -π en π ligt zodat de normale sinus reeksontwikkeling gebruikt kan gestart worden.

Met de sinus_controle_blok is het ook mogelijk om het argument van de boogsinus te controleren. Hierbij zal het argument niet worden terug gebracht naar het interval [$-\pi$,π], zoals bij de sinus, maar hier zal er worden gekeken als de absolute waarde van het argument groter is dan $1/\sqrt{2}$. Dit om de waardes die groter zijn dan $1/\sqrt{2}$ om te vormen, zoals vermeld in 6.5.2.

De methode om te controleren als het argument groter is dan $1/\sqrt{2}$ is gelijkaardig aan de manier die ook wordt gebruikt bij de sinus (om te controleren of de hoek binnen het bepaalde interval lag).

Eerst worden door de FSM 2 variabelen aangemaakt zijnde het te berekenen argument $+1/\sqrt{2}$ en het te berekenen argument -$1/\sqrt{2}$. Deze twee waarden worden naar de sinus_controle_blok gestuurd en daar wordt, aan de hand van de tekenbit van deze twee variabelen, bepaald of de absolute waarde van het argument al dan niet groter is dan $1/\sqrt{2}$.

Wanneer het argument groter is dan deze waarde zal het arcsin_controle_valid signaal "1" worden gemaakt, zodat de FSM weet dat deze het argument eerst moet transformeren volgens 6.5.2 alvorens de reeksontwikkeling van de boogsinus te beginnen.

Bij een arcsin_controle_valid signaal van "0" weet de FSM dat de normale boogsinus reeksontwikkeling onmiddellijk kan worden gestart.

6.6 ALU FSM

De ALU_FSM zal ervoor zorgen dat de juiste bewerkingen worden uitgevoerd op de ontvangen parameters om zo tot de satellietpositie te komen. Hiervoor worden telkens de juiste parameters aan de uitgang van de RAM blok klaargezet, en worden de juiste enable-signalen hoog gemaakt om de juiste bewerkingen uit te voeren. Alvorens de bewerkingen kunnen uitgevoerd worden, moeten de parameters eerst in het RAM-geheugen ingeladen worden. Dit gebeurt in het eerste deel van de FSM.

6.6.1 Parameters en variabelen inladen

Nadat de parser alle parameters uit de navigatieboodschap heeft omgezet naar 64bit floating point, zal er een `parser_done` signaal naar de *satellietpositiecalculatieFSM* gestuurd worden. Hierdoor weet deze FSM dat alle parameters klaar staan om berekeningen uit te voeren. Bijgevolg zal de *satellietpositiecalculatieFSM* het enable signaal van de ALU_FSM hoog maken. Vanaf het moment dat dit signaal hoog is begint de ALU_FSM met het wegschrijven van de parameters naar het RAM-geheugen.

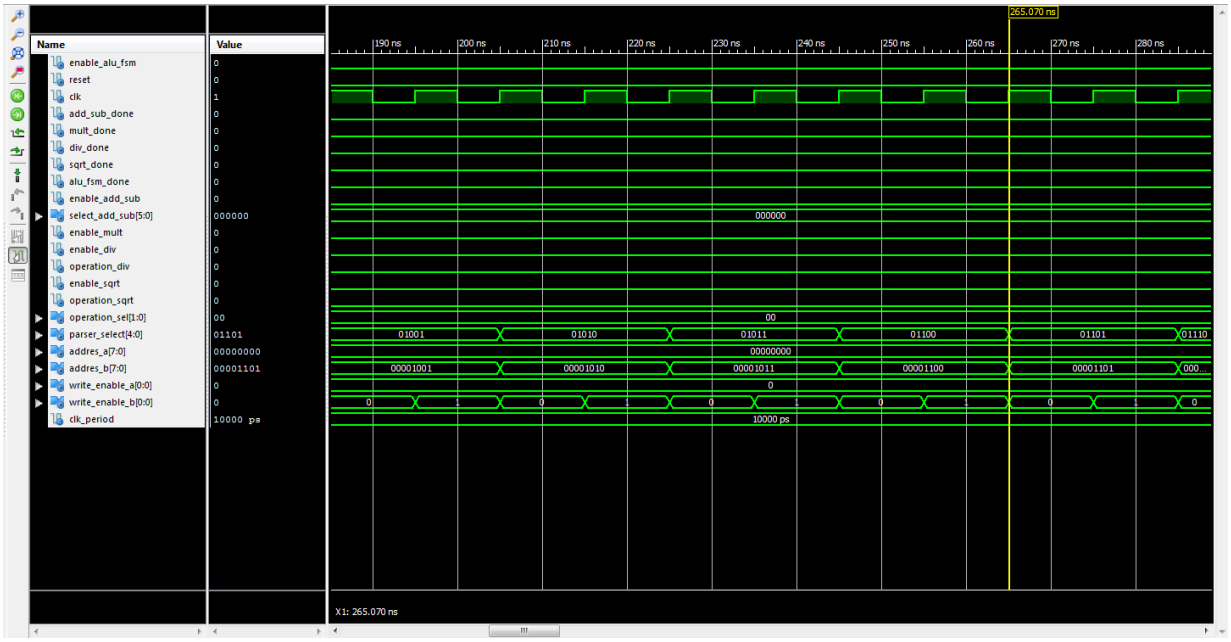
De parser, die de parameters moet gaan wegschrijven, heeft 25 uitgangen (één voor iedere variabele). Er kan slechts één uitgang van de parser verbonden worden met het RAM-geheugen. Hiervoor is er een *25 to 1 multiplexer* ontworpen die ervoor zorgt dat iedere variabele toch verbonden kan worden met het RAM-geheugen.

Het wegschrijven van de parameters gebeurt in 2 states. In de eerste state worden de selectielijn van de *25 to 1 multiplexer* en het adres van het RAM-geheugen aangestuurd. Met het adres van de *25 to 1 multiplexer* kan men de parameter die men wil wegschrijven selecteren en aan de uitgang van de *25 to 1 multiplexer* zetten. Deze uitgang is verbonden met de B-ingang van het RAM-geheugen. Het adres van het RAM-geheugen bepaalt waar de waarde naartoe wordt geschreven. Om dit moment staat alles klaar om de parameter definitief op te slaan in het RAM geheugen.

```
1                   when init_variabelen_01010 =>
2                           ADDRES_B <= "00001010";
3                           ADDRES_A <= "00000000";
4                           WRITE_ENABLE_B <= "0";
5                           parser_select <= "01010";
6
7                   when WE_B_01010 =>
8                           ADDRES_B <= "00001010";
9                           ADDRES_A <= "00000000";
10                          WRITE_ENABLE_B <= "1";
11                          parser_select <= "01010";
```

In de tweede state worden de signalen, die de selectie van de *25 to 1 multiplexer* en het adres van het RAM-geheugen bepalen, nog steeds aangestuurd met dezelfde waarden als uit de eerste state (`init_variabelen_01010`). Wat wel veranderd is, is dat het `WRITE_ENABLE_B` signaal van het RAM-geheugen hoog wordt gemaakt. Op deze manier wordt de parameter definitief opgeslagen in het RAM-geheugen.

Bij het wegschrijven van de volgende parameter wordt het `WRITE_ENABLE_B` signaal weer laag gemaakt en de adressenlijnen weer aangestuurd met de adressen van de volgende parameter. Belangrijk is om het `WRITE_ENABLE_B` signaal terug laag te maken nadat alle parameters zijn weggeschreven en de FSM begint met de eigenlijke berekeningen.



Figuur 6.8: Testbench waar de variabelen worden ingelezen.

Sommige variabelen, die op dit moment zijn ingeladen in het RAM geheugen, staan nog uitgedrukt in semicircles. Deze variabelen zijn Δn, M_0, Ω_0, i_0, Ω, $\dot{\Omega}$ en $idot$. Aangezien doorheen de berekeningen telkens gewerkt zal worden met radialen als eenheid voor hoeken zullen deze variabelen, alvorens de berekeningen te starten, omgezet worden naar radialen. Dit gebeurt door de waardes in semicircles te vermenigvuldigen met π. Hoe deze vermenigvuldiging concreet wordt uitgevoerd, wordt besproken in 6.6.2.

6.6.2 Berekeningen uitvoeren

Wanneer de parameters en de variabelen zijn ingelezen, kunnen de bewerkingen worden uitgevoerd. De bewerkingen zullen uitgevoerd worden in de volgorde die weergegeven is in bijgevoegde flowchart (zie 6.8) om zo aan het einde de positie van de satelliet te bekomen. Om de bewerkingen uit te voeren wordt er ook gebruik gemaakt van 2 states per bewerking. Onderstaand voorbeeld geeft de 2 states weer om een optelling uit te voeren.

```
1                when vb_optelling =>
2                        ADDRES_A <= "00000100";
3                        ADDRES_B <= "00010100";
4                        WRITE_ENABLE_A <= "0";
5                        WRITE_ENABLE_B <= "0";
6                        enable_add_sub <= '1';
7                        select_add_sub <= "000000";
8                        operation_sel <= "00";
9
10               when vb_optelling_save =>
11                       ADDRES_A <= "00110001";
12                       WRITE_ENABLE_A <= "1";
13                       enable_add_sub <= '0';
```

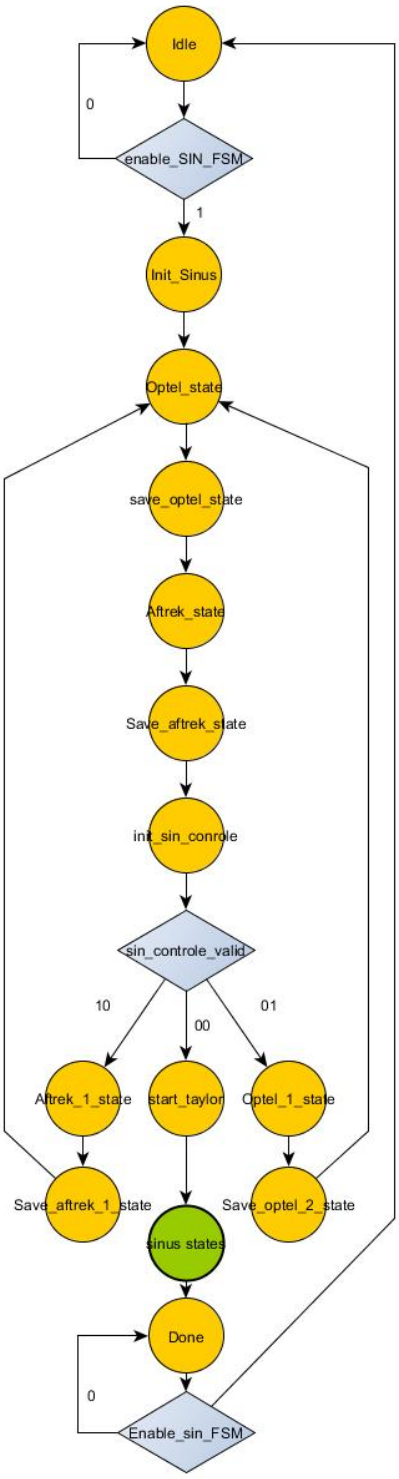
In de eerste state (`vb_optelling`) wordt het adres van de 2 gebruikte parameters in address_a en address_b gezet zodat de juiste parameters aan de uitgang van het RAM-geheugen staan. Bij de vierkantswortel zal alleen address_a worden aangestuurd omdat hierbij maar 1 parameter nodig is. Ook zal het juiste enable signaal hoog gemaakt worden, in dit geval is dit enable_add_sub. Speciaal bij deze blok is dat er nog bepaald moet worden of de blok moet optellen of aftrekken. Dit doen we met het select_add_sub signaal, waarbij er bij 000000 wordt opgeteld en bij 000001 de getallen van elkaar worden afgetrokken.

Het laatste signaal dat in deze state nog wordt aangestuurd is het operation_ sel signaal. Dit signaal controleert de *4 to 1 multiplexer* die voor de A ingang van het RAM-geheugen staat. Aan de vier ingangen van de *4 to 1 multiplexer* hangen telkens de uitgangssignalen van de 4 bewerkingsunits zijnde de add/sub operator, de vermenigvuldigingsoperator, de delingsoperator en de vierkantswortelоperator. Met het operation_sel signaal is het dus mogelijk om te kiezen welk resultaat er wordt opgeslagen.

In de 2de state (`vb_optelling_save`) wordt het resultaat weer opgeslagen in het gewenste adres. Deze state wordt uitgevoerd vanaf het moment dat het "add_sub_done" signaal hoog wordt. Om het resultaat op te slaan wordt address_A aangestuurd samen met het Write_enable_A signaal. Ook wordt het enable signaal van de gebruikte operator terug laag gemaakt.

Sinus bewerking

Aangezien de sinusbewerking redelijk vaak moet uitgevoerd worden, is ervoor gekozen om een FSM te implementeren die deze sinus uitrekent. De `Sin_FSM` zal aan de hand van 2 doorgestuurde adressen de bewerkingen kunnen uitvoeren. Deze twee adressen zijn afkomstig van de ALU_FSM en bevatten de volgende waarden. Op het eerste adres bevindt er zich de parameter waarvan de sinus berekend moet worden. Het tweede adres vertelt de SIN_FSM waar de uitkomst zal worden opgeslagen. Dit wordt gedaan zodanig dat de ALU_FSM na de berekeningen weet waar de uitkomst terug te vinden is. Ook zijn er 2 vaste adresruimtes voorzien in het RAM blok die uitsluitend dienen voor het opslaan van de tussenresultaten van de berekening. Deze zijn niet specifiek voor een bepaalde sinusbewerking en worden dus telkens overschreven als er een nieuwe sinus zal berekend worden.



Figuur 6.9: Flowchart van de sinus FSM.

In figuur 6.9 is de flowchart van de FSM weergegeven. Hier is te zien dat de FSM steeds in de idle state verblijft, tenzij het enable_SIN_FSM signaal hoog wordt. Wanneer dit gebeurt, beginnen de berekeningen van de sinus.

Eerst wordt er gekeken als de hoek in het interval $[-\pi, \pi]$ ligt. Hiervoor moeten, zoals vermeld in 6.5.3, het argument + en - π worden berekend. Dit wordt gedaan in de optel- en aftrek_state. Het resultaat van deze bewerkingen wordt telkens opgeslagen in de save_optel_state en de save_aftrek_state. Ook hier wordt er gebruik gemaakt van 2 gereserveerde adresruimtes die bij elke nieuwe sinus worden hergebruikt.

Wanneer dit gebeurd is kunnen deze waardes naar de sinus controle blok worden gestuurd. Dit gebeurt in de init_sin_controle state. Wanneer de sinus controle blok een "10" signaal terugstuurt naar de FSM, betekent dit dat de hoek groter is dan π. Als dit het geval is wordt er vervolgens 2π van de hoek afgetrokken.

Als de controle blok een "01" signaal terugstuurt wil dit zeggen dat de hoek kleiner is dan π waardoor er 2π bij de hoek zal worden opgeteld. Dit wordt herhaald tot de hoek ligt tussen $-\pi$ en π.

Wanneer de hoek in dit interval ligt zal de sinus controle blok een "00" signaal terugsturen waarna de eigenlijke sinus berekening kan gestart worden. Dit gebeurt via onderstaande reeksontwikkeling.

$$sin(x) = x - (x^3)/3! + (x^5)/5! - (x^7)/7! + (x^9)/9! - (x^{11})/11! + (x^{13})/13! - (x^{15})/15! + (x^{17})/17! - (x^{19})/19! + (x^{21})/21!$$

Hier is te zien dat er bij de uitwerking van de reeksontwikkeling verder gewerkt kan worden met voorgaande waardes, zodat deze niet telkens opnieuw moeten worden berekend. Zo is het mogelijk om de reeksontwikkeling zo efficiënt mogelijk en met minimale geheugenruimte uit te rekenen.

Om te beginnen wordt x in een geheugenplaats van het RAM blok opgeslagen. In een andere geheugenplaats (in dit geval Plaats 2) wordt de bewerking $x \cdot x$ opgeslagen.

Plaats 1	x
Plaats 2	x^2
Plaats 3	*leeg*
Plaats 4	*leeg*

Deze wordt daarna overschreven door zichzelf te vermenigvuldigen met x om zo x^3 te verkrijgen.

Plaats 1	x
Plaats 2	x^3
Plaats 3	*leeg*
Plaats 4	*leeg*

Als dit berekend is, is het mogelijk om de volledige tweede term van de reeksontwikkeling uit te rekenen. Dit omdat ervoor gekozen is om de faculteitstermen, die in de noemer zijn terug te vinden, op te slaan als constanten in het RAM-geheugen om zo een besparing te doen in rekentijd.

De deling duurt 57 klokpulsen om uit te voeren, zie tabel 6.2. Dus het duurt nu nog 57 klokpulsen vooraleer de deling is uitgevoerd. Later is dit nog versneld door niet de faculteit op te slaan, maar de volledige $\frac{1}{faculteitsterm}$. Aangezien er dan slechts 1 vermenigvuldiging moet worden uitgevoerd, en de vermenigvuldiging slechts 17 klokpulsen nodig heeft, maakt dit de berekening 40 klokpulsen sneller per term. In een volgende geheugenplaats komt dus $\frac{x^3}{3!}$.

Plaats 1	x
Plaats 2	x^3
Plaats 3	$(x^3)/3!$
Plaats 4	*leeg*

Merk op dat de term x^3 deze keer niet wordt overschreven zodat deze hierna nog kan gebruikt worden om x^5 uit te rekenen. De volgende stap die gemaakt wordt is de volledige uitkomst van de eerste twee termen van de reeksontwikkeling uit te rekenen en op te slaan op plaats 4. Hierna wordt de teller van de derde term uitgerekend door de 2de geheugenplaats 2 keer te vermenigvuldigen met de 1ste geheugenplaats om zo x^5 te bekomen.

Plaats 1	x
Plaats 2	x^5
Plaats 3	$(x^3)/3!$
Plaats 4	$x - (x^3)/3!$

Vervolgens kan de waarde op geheugenplaats 2 vermenigvuldigd worden met $\frac{1}{5!}$ waarna deze kan worden opgeslaan in geheugenplaats 3. Dit resultaat wordt daarna opgeteld bij $x - (x^3)/3!$. Dit wordt herhaald totdat alle termen van bovenstaande reeksontwikkeling zijn uitgerekend.

Al deze bewerkingen zijn in de flowchart (figuur 6.9) voorgesteld door de `sinus states` om de flowchart overzichtelijk te houden. Aan het einde van de berekening wordt het sinus_FSM_done signaal nog hoog gemaakt, in de done state, zodat de ALU_FSM weet dat de sinus berekend is en deze zijn bewerkingen weer kan verderzetten. Vervolgens gaat de Sin_FSM terug naar zijn idle state tot het volgende enable_SIN_FSM signaal weer hoog wordt.

Omdat er nu een FSM is bijgekomen (de SIN_FSM) is er dus een praktisch probleem omtrent de interfacing. Zo moeten de SIN_FSM en de ALU_FSM beiden toegang hebben tot alle operatoren. Om dit op te lossen zijn alle bestaande aanstuursignalen opgedeeld in twee aparte controlesignalen. Dit om het mogelijk te maken dat zowel de SIN_FSM als de ALU_FSM alle blokken kunnen aansturen.

De twee ontdubbelde controlesignalen worden dan gestuurd naar de ingangen van een *2 to 1 multiplexer*. De uitgang van deze *2 to 1 multiplexer* is dan op zijn beurt weer verbonden met de juiste ingang van de operator.

De *2 to 1 multiplexer* worden met een gemeenschappelijk `mux_switch` signaal aangestuurd. Dit signaal wordt gecontroleerd door de ALU_FSM, en zal standaard op 0 staan. In deze stand zullen alle componenten enkel naar de ALU_FSM luisteren. Wanneer er een sinus zal berekend moeten worden zal de ALU_FSM dit signaal hoog maken zodat de SIN_FSM alle componenten kan aansturen en op die manier de sinus berekenen. Wanneer de ALU_FSM een sinus_FSM_done signaal, zal deze het mux_switch signaal terug laag maken.

Om het overzicht te bewaren worden de verschillende *2 to 1 multiplexer* gegroepeerd weergegeven in het uiteindelijke blokschema. Deze noemt de *Multiplexer_Switch* en is weergegeven in het groen.

Boogsinus bewerking

De implementatie van de boogsinus is ook via een reeksontwikkeling, zoals reeds besproken 6.5.2. Aangezien er slechts één boogsinus in de berekeningen voorkomt is het dus overbodig om hiervoor een nieuwe FSM te gaan ontwerpen. Het is dus ook om deze reden dat de uitvoering van de boogsinus gedaan wordt door de ALU_FSM.

Transfer FSM

Gedurende het onderzoek werd er opgemerkt dat er "maar" 256 states in één FSM pasten. Helaas worden er ongeveer 550 states gebruikt. De oplossing hiervoor was het ontdubbelen van de ALU_FSM in drie delen. Indien de eerste ALU_FSM klaar is komt deze in de `transferFSM state` wat aangeeft dat de tweede ALU_FSM kan beginnen. Indien de tweede FSM klaar is wordt de derde FSM aangesproken.

Dit bracht uiteraard ook interfacing problemen met zich mee. Zo moeten de tweede en de derde FSM ook toegang hebben tot de operatoren. Dit is opgelost door gebruik te maken van een tweede MUX-switch, genaamd `FSM_Switch`. Deze laat de twee nieuwe ALU_FSMś toe tot de gebruikte modules. De eerste ALU_FSM is dus de hoofd FSM. Deze controleert de `FSM_Switch`.

6.7 Gebruikte resources en blokschema

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	4,107	21,504	19%
Number used as Flip Flops	4,042		
Number used as Latches	65		
Number of 4 input LUTs	4,963	21,504	23%
Number of occupied Slices	3,563	10,752	33%
Number of Slices containing only related logic	3,563	3,563	100%
Number of Slices containing unrelated logic	0	3,563	0%
Total Number of 4 input LUTs	5,128	21,504	23%
Number used as logic	4,379		
Number used as a route-thru	165		
Number used as Shift registers	584		
Number of bonded IOBs	69	448	15%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number of FIFO16/RAMB16s	2	72	2%
Number used as RAMB16s	2		
Number of DSP48s	12	48	25%
Average Fanout of Non-Clock Nets	3.23		

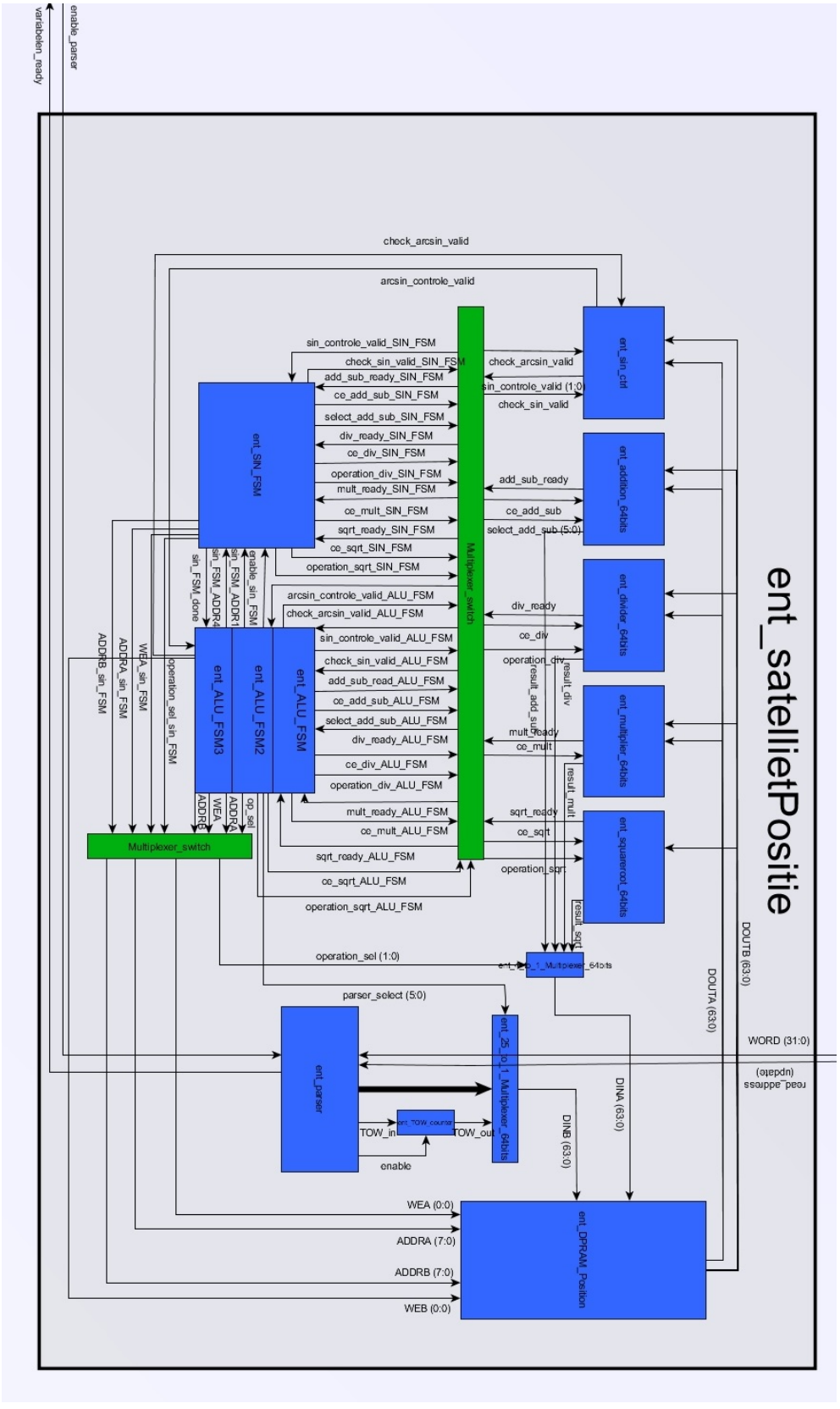
Figuur 6.10: De volledig ingenomen oppervlakte.

Tabel 6.5: implementatiegrootte bij 16 kanalen

	Volledige impl. (16 kanalen)	**Beschikbaar**	**Gebruikt**
Number of slices	7799	21504	36,27%
Number of 4 input LUTs	17665	21504	82,15%
Number of occupied slices	11065	10752	102,91%
Number of bonded IOBs	80	448	17,86%

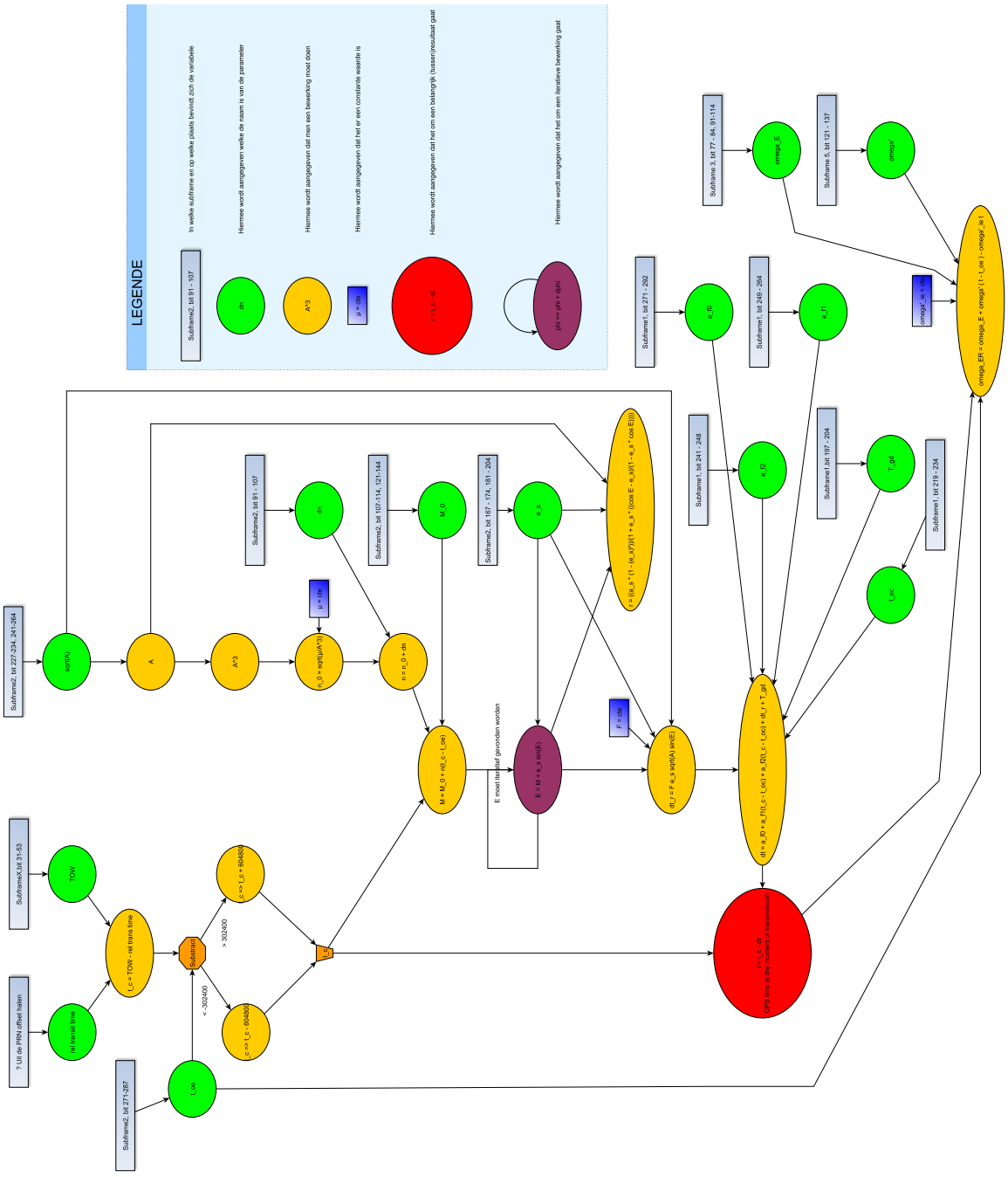
Tabel 6.6: implementatiegrootte bij 8 kanalen

	Volledige impl. (8 kanalen)	**Beschikbaar**	**Gebruikt**
Number of slices	6278	21504	29,19%
Number of 4 input LUTs	13314	21504	61,91%
Number of occupied slices	8382	10752	77,96%
Number of bonded IOBs	80	448	17,86%

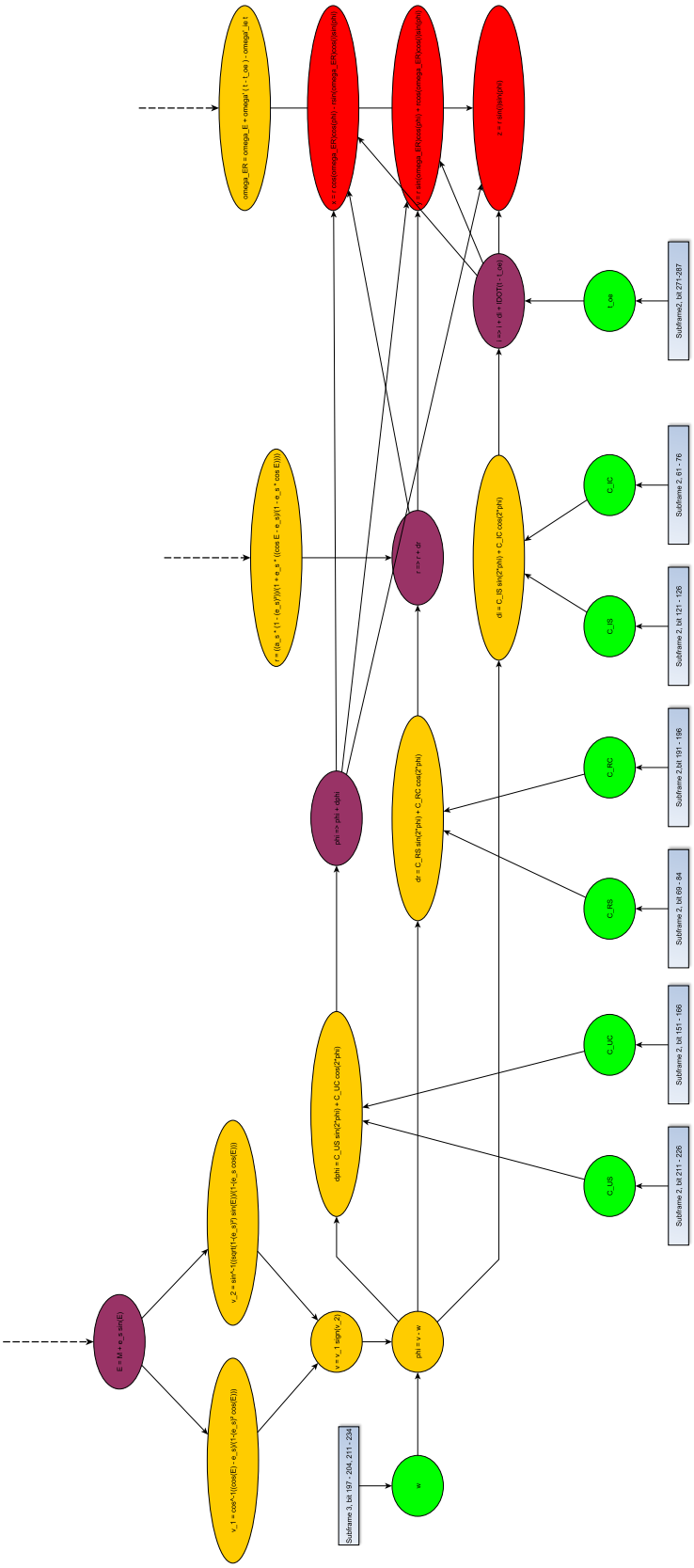


Figuur 6.11: Het volledige blokschema.

6.8 Flowchart om de satellietpositie te berekenen



Figuur 6.12: Flowchart pt1.



Figuur 6.13: Flowchart pt2.

Hoofdstuk 7

Resultaten, evaluatie en besluit

In dit hoofdstuk worden de resultaten, evaluatie en besluit van deze thesis besproken. Eerst wordt er een vergelijking gemaakt tussen de theoretische (matlab) en de praktische (Hardware implementatie) bekomen satellietposities. Vervolgens wordt de rekentijd van het algoritme besproken, gevolgd door een evaluatie van het onderzoek. Het hoofdstuk wordt afgesloten met een conclusie.

7.1 Resultaten

De resultaten zijn getest aan de hand van een RINEX bestand [7]. De parameters van dit bestand zijn ingelezen door de parser en worden als variabelen gebruikt. Ook worden deze variabelen in Matlab ingeladen en stap voor stap nagerekend. De resultaten bekomen door de hardware implementatie worden nu vergeleken met de resultaten van Matlab. Deze resultaten staan in tabel 7.1.

Zoals reeds vermeld is één van de variabelen t_c. Deze kan berekend worden via de *PRN-offset* en de *accumulator reset* (uit de aqcuisitie unit). Helaas is deze berekening niet geïmplementeerd aangezien in de laatste fase van dit onderzoek onze focus vooral uitging naar het correct berekenen van de eindcoördinaten. Wegens tijdgebrek is dit uiteindelijk niet gerealiseerd geraakt.

Tabel 7.1: Vergelijking van implementaties.

Variabele	**HW implementatie**	**Matlab**
E	9,1533489677519 ...	9,1533489675409
Δt_r (10^{-9})	-3,8474324029957 ...	-3,8474319197448
Δt (10^{-4})	-8,4437426475816 ...	-8,4437426475767
t (10^{5})	6,0480000084437 ...	6,0480000084437
$\sin(E)$ (10^{-1})	2,6810841964451 ...	0,2681083859691
Ω_{ER} (10^{1})	-4,3774729510505 ...	-4,37747195105049
$\cos(E)$ (10^{1})	-9,6338875511440 ...	-9,633887550584
ν_1	2,8944666885350 ...	2,8944666883061
ϕ	4,9640465658158 ...	4,9640465655870
i	1,1154765441543 ...	1,1154765447294
r (10^{7})	2,6720196404898 ...	2,6720196404890
$\delta\phi$ (10^{-6})	-7,3860692230231 ...	-7,3860691836379
δi (10^{-8})	4,9961361850870 ...	4,9961361487197
δr (10^{2})	-3,3114405252394 ...	-3,3114405200710
x (10^{6})	8,8560771810845 ...	8,8560761522451
y (10^{6})	-9,7645396902788 ...	-9,7645399076444
z (10^{7})	-2,3242043185364 ...	-2,3242043186724

Zoals weergegeven in tabel 7.2 blijkt de implementatie nauwkeurig te zijn tot op 1 meter. Dit wil zeggen dat de doelstelling om nauwkeurig aan positiebepaling te doen bereikt is.

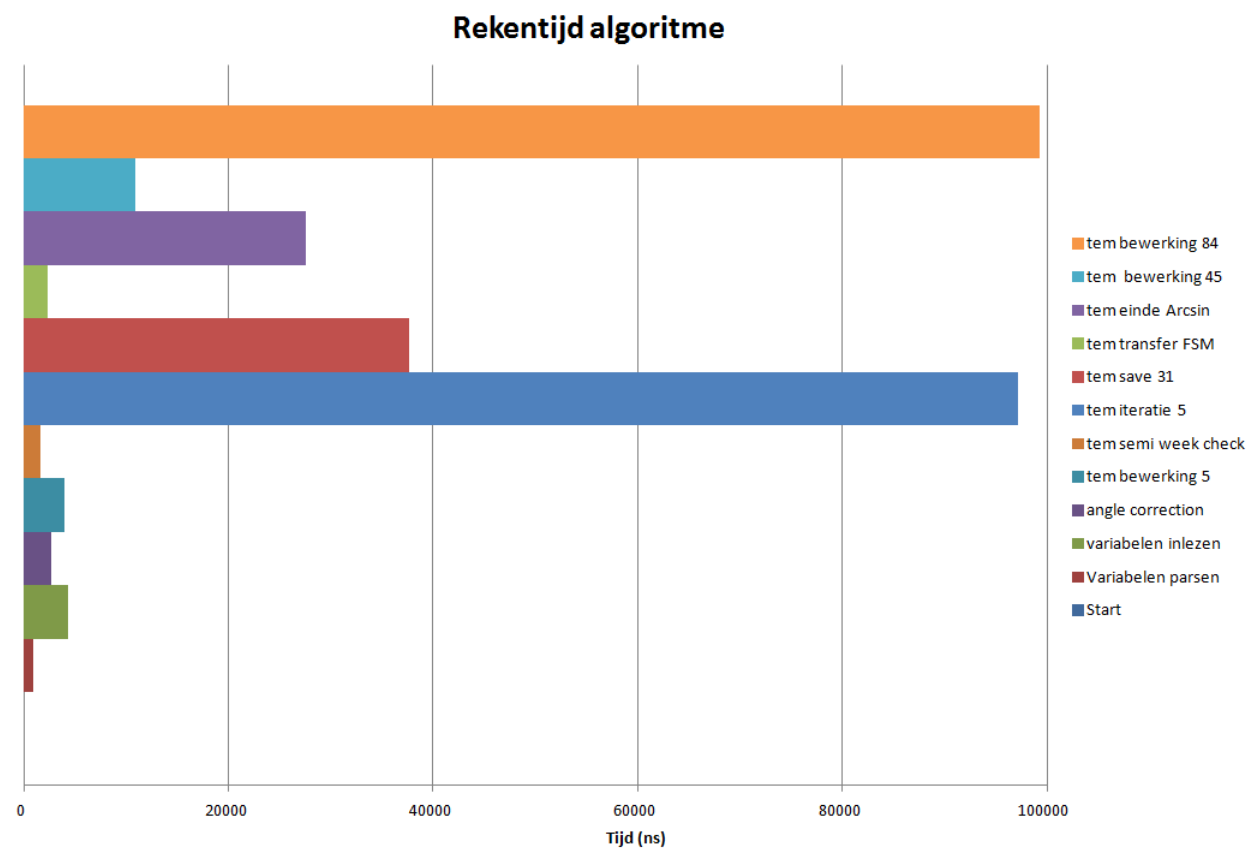
Tabel 7.2: Vergelijking van implementaties.

Variabele	**HW implementatie** $(10^6\ m)$	**Matlab** $(10^6\ m)$	**Afwijking** (m)
x	8,8560771810845 ...	8,8560761522451	1,0288
y	-9,7645396902788 ...	-9,7645399076444	0,2147
z	-23,242043185364 ...	-23,242043186724	0,0014

De bewerkingen van de ALU_FSM worden stelselmatig uitgevoerd. Figuur 7.1 geeft weer hoe lang het duurt vooraleer bepaalde bewerkingen zijn uitgevoerd.

Wat opvalt, is dat de grootste tijd ingenomen wordt door de laatste bewerkingssectie. In deze sectie wordt alles berekend vanaf ϕ, zoals getoond in figuur 6.13. Ook neemt de iteratiesectie veel tijd in beslag. Dit komt omdat er steeds een nieuwe sinus berekend moet worden, zie vergelijking 6.4. Tevens toont de grafiek dat de boogsinusbewerking relatief gezien veel tijd nodig heeft.

Vervolgens is het zo dat het inlezen en parsen van de gegevens erg weinig tijd in beslag neemt.



Figuur 7.1: De gebruikte tijd per sectie.

De waarden overeenkomstig met figuur 7.1 worden weergegeven in tabel 7.3.

Tabel 7.3: Berekingstijden implementatie

Secties	Tijd(ns)
Start	0
Variabelen parsen	920
Variabelen inlezen	4320
Angle correction	2750
tem bewerking 5	3970
tem semi week check	1680
tem iteratie 5	97120
tem save 31	37680
tem transfer FSM	2310
tem einde arcsin	27610
tem bewerking 45	10890
tem bewerking 84	99320
Totaal	288570

7.2 Evaluatie

Tijdens het afronden van het onderzoek zijn er ook topics aan bod gekomen die voor optimalisatie vatbaar zijn. Deze optimalisaties worden hier kort toegelicht voor toekomstig onderzoek aan deze satellietpositieberekeningsmodule.

7.2.1 ALU FSM optimalisatie

In het eindresultaat van de FSM's zitten ongeveer 550 states. Dit is erg veel en is ook erg omslachtig. Zo is het moeilijk om overzicht te bewaren over de uitgevoerde bewerkingen. Beter zou zijn indien de FSM's vervangen zouden worden door bijvoorbeeld een MicroBlaze. Deze softcore processor zou dan enkel de operatoren moeten aansturen. Deze aansturing gebeurt via c code. Dit heeft als voordeel dat er een overzicht wordt bewaard van de gebruikte bewerkingen.

Een andere soort aanpassing zou kunnen zijn dat de gehele satellietpositieberekening in een MicroBlaze gebeurt. Dit heeft als voordeel dat er geen aparte operatorentiteiten voorzien moeten worden. Dus is er ook een plaatsbesparing. Het nadeel zou wel zijn dat de snelheidswinst om het in hardware uit te rekenen teniet wordt gedaan.

7.2.2 Sinus FSM optimalisatie

Tabel 7.3 geeft weer dat de sinusbewerkingen de meeste tijd innemen. Er zijn optimalisaties mogelijk die deze tijd kunnen reduceren. Zo kan het algoritme om de sinus te berekenen worden aangepast. In dit onderzoek worden de coëfficiënten berekend door steeds de teller van de vorige waarde 2 keer $\times x$ te doen. Dit kan geoptimaliseerd worden door bijvoorbeeld x^2 in het RAM-geheugen op te slaan en dan $\times x^2$ uit te voeren.

Een andere manier om de rekentijd beter te besteden is als de sinus operatie gepipelined wordt. Dit kan bekomen door bij de SIN_FSM operatormodules te plaatsen, meer bepaald een `adder/subtractor` en een `multiplier`. Zo kan er een sinus berekend worden terwijl de ALU_FSM verder gaat met zijn routine.

7.3 Conclusie

Het doel van deze thesis was het ontwerpen, implementeren en evalueren van een satellietpositieberekeningsmodule. Eerst worden de gegevens uit het RAM-geheugen, van de demodulatiemodule, gehaald. Vervolgens worden deze geparset naar een 64 bit IEEE 754 standaard. Daarna worden deze variabelen in een nieuw RAM-geheugen geplaatst en ten slotte worden deze variabelen gebruikt om de positie van een satelliet te berekenen.

Het eerste deel van deze thesis beperkt zich tot het inlezen en parsen van de gegevens en deze stap is succesvol afgerond. Het tweede deel is het ontwerpen van de hardware van het bewerkingsalgoritme. Ook dit is succesvol afgehandeld in dit onderzoek.

Zoals de resultaten [7.1] aangeven, blijkt de hardware implementatie te werken, want de variabelen kunnen correct berekend worden. Zo worden alle operatoren foutloos gebruikt en de ALU - en SIN FSM's werken naar behoren. Ook geeft tabel 7.1 weer dat de vooropgestelde nauwkeurigheid bereikt wordt. De afwijking op de satellietpositie is ongeveer 1 meter. Ook is de hardware implementatie snel genoeg om meerdere kanalen te ondersteunen.

In sectie 7.2 is wel te lezen dat er zeker nog ruimte is voor optimalisatie van het geleverde werk. Dit zowel voor de SIN- als ALU-FSM.

Bibliografie

[1] James Bao-Yen Tsui, *Fundamentals of Global Positioning System Receivers: A software Approach*, John Wiley & Sons. Inc, 2000.

[2] US gov., *Global Positioning System Standard Positioning Service Signal Specification*, Department Of Defence, 2008.

[3] Safaa Dawoud, *GNSS principles and comparison*, Potsdam University,.

[4] Jimmy Gysens, *Plaatsbepalingseenheid op FPGA*, KHLim Hogeschool, 2013.

[5] Ruben Smeets, Joris Volders, *Het implementeren van een meerkanaals GNSS ontvanger met flexibele herconfiguratie op een FPGA*, KHLim Hogeschool, 2012.

[6] Raf Martino, Ben Willems, *Implementatie van hardwarebouwblokken voor een meerkanaals gps-ontvanger met positiebepaling op FPGA*, KHLim Hogeschool, 2011.

[7] Werner Gurtner,Lou Estey, *RINEX: The Receiver Independent Exchange Format*, University of Bern, UNAVCO, 2009.

[8] Thomas Herring, *Principles of the Global Positioning System*, MIT, 2012.

[9] Wikipedia, *http://en.wikipedia.org/wiki/Trigonometric_functions*, geraadpleegd op 3/06/2014.

[10] Hough, David, *Applications of the proposed IEEE 754 standard for floating-point arithetic.*, Computer 14.3 (1981): 70-74.

Lijst van figuren

Lijst van tabellen

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
H a r d w a r e - i m p l e m e n t a t i e e n - e v a l u a t i e v a n e e n satellietpositieberekeningsmodule

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Moors, Jan **Ceyssens, Pieter-Jan**

Datum: **6/06/2014**