Karel de Grote-Hogeschool
Katholieke Hogeschool Antwerpen
Departement Industriële Wetenschappen en Technologie

campus Hoboken

Uitlezing van een seriële link via een USB-poort

door Joris Van Goethem

Promotoren: ir. Wim Beaumont, UA-FYS

ing. Koen Lostrie, KdG–IWT

Proefschrift tot het
behalen van de graad van
Master in de industriële wetenschappen: elektronica-ICT
afstudeerrichting ICT
Hoboken, juni 2010

Voorwoord

Toen KdG-IWT (met name dr. ir. M. Temmerman) me vroeg om een master-eindwerk te willen uitvoeren met als titel 'Uitlezing van een seriële link via een USB-poort' kwamen er direct al een paar vragen opduiken. Zeker als het om een seriële link uitlezing gaat dat van een project uit het CERN kwam. CERN was toen nog een vaag begrip voor me en stond bij mij bekend om zijn deeltjesversneller en de uitvinding van het 'World Wide Web'. Na wat opzoekwerk bleek al duidelijk dat het CERN een internationale organisatie is die vooral onderzoekswerk doet naar elementaire deeltjes maar om dit waar te maken is er ook ICT en elektronica nodig. Het was dan ook meteen duidelijk dat er bij een organisatie zoals het CERN niet enkel fysici nodig zijn maar ook ICT-elektronica mensen die instaan voor de ontvangst en verwerking van data in het CERN.

De term 'USB' in de titel gaf ook voor mij de doorslag om aan dit eindwerk te beginnen. USB is in deze tijden een zeer populaire standaard die op bijna alle PC's wordt gebruikt. Het lijkt me dan ook uiterst interessant en nuttig om deze standaard verder uit te diepen en te praktiseren. Echter met een 'paar lijntjes' code heb je een USB-controller niet geconfigureerd bleek al gauw in mijn stageperiode. USB met name USBv2 is een vrij complexe standaard en vraagt enige tijd om te bestuderen. De combinatie tussen deze complexe USB-controller en de FPGA leek me dan ook een uitdaging die ik met enthousiasme aanging toen ik aan mijn stageperiode begon.

Vooraleer we starten met het onderzoek naar de USB-uitlezing zou ik graag nog enkele mensen willen bedanken. Eerst en vooral mijn promotor ir. W. Beaumont die altijd bereid was te helpen indien nodig en waarvan ik veel heb bijgeleerd op professioneel vlak. Ook mijn mentor ing. K Lostrie wil ik bedanken voor zijn deskundige begeleidingen en hulp bij problemen. Dr. ir. M. Temmerman die mij de mogelijkheid gaf om op dit interessante onderwerp te werken. Ook nog een woord van dank aan de rest van de onderzoeksgroep elementaire deeltjesfysica, KdG, UA en mijn medestudenten. Tot slot wil ik ook nog mijn ouders en vriendin bedanken om me altijd te steunen en te helpen met het verbeteren van de thesis op het niet-technologisch vlak.

Joris Van Goethem
Schilde, mei 2010

Opdrachtomschrijving

Primair zal de opdracht worden gericht op de software en firmware "aanpassingen" te ontwikkelen die nodig zijn om de COR (CASTOR optische receiver) via de USB uit te lezen. Hiervoor moet de aanwezige USB-controller chip worden geconfigureerd voor optimale communicatie tussen de buitenwereld en de uitleesmodule van de CASTOR (= QIE-kaarten). Verder moet er een eenvoudige GUI worden ontworpen zodat de gebruiker de data kan waarnemen en eventueel lokaal opslaan. Volgende zaken moeten worden behandeld:

1) Bestuderen van de componenten
   a) FPGA
   b) USB interface
2) gereed maken van de setup :
   a) ontbrekende componenten (XTAL, condensator)
   b) voedingen
   c) aansluiten
   d) signaal bronnen.
   e) Windows computer => dual boot Linux computer
3) Windows en /of Linux USB driver vinden
   a) installeren
   b) uitlezen interne register USB interface
4) Eenvoudige communicatie met de FPGA
   a) hello world (LED) , met signal tap
   b) hello world met simulator
   c) read / write via the USB

Wenselijk is het om een complete QIE readout setup te realiseren zonder de noodzaak de gehele VME interface structuur hier aanwezig te hebben, een 'lokale ontwikkel opstelling'.
Hiervoor moeten volgende zaken worden onderzocht:

1) bestuderen van de parallelle local bus implementatie, die wordt gebruikt voor configuratie binnen de FPGA
2) hoofdproject zo aanpassen dat de datastroom eenvoudig is om te leiden naar de USB.
3) Project repository gebruiken zodanig dat aanpassingen in het hoofdproject automatisch worden overgenomen in het USB project
4) oplossingen / strategieën definiëren zodat er geen buffer overflows worden veroorzaakt (throttle in het hoofd project). De data stroom is een push technologie. Er wordt aangenomen dat er altijd buffers vrij zijn in de ontvanger.

Samenvatting

In deze thesis wordt de wijze waarop een seriële link via een USB-poort kan worden uitgelezen verklaard. Het belangrijkste onderdeel voor deze uitlezing is de USB-controller of ook wel USB-interface genoemd. Deze zal moeten worden geconfigureerd zodat de FPGA data kan zenden via de USB-interface naar de gebruiker of PC. Voor het configureren van de USB-interface is gebruikt gemaakt van VHDL en de ontwikkelsoftware QUARTUS II. Er kan zowel data in als uit de USB-interface worden getransporteerd.

De data die wordt ontvangen door de gebruiker moet op een eenvoudige wijze worden waargenomen aan de hand van een gebruiksvriendelijke GUI. De GUI is ontwikkeld met Labview en toont grafisch een overzicht aan de gebruiker hoe hij data kan uitlezen en verzenden van en naar de FPGA.

Inhoudsopgave

Gebruikte afkortingen

CASTOR: Centauro And STrange Object Research
CERN: Conseil Européen pour la Recherche Nucléaire
CMS: Compact Muon Solenoid
COPTRX: CASTOR optic receiver
CPLD: Complex Programmable Logic Device
EOP: End of Packet
FIFO: First In First Out
FPGA: Field-programmable gate array
IP core: intellectual property core
LHC: Large Hadron Collider
NRZI: NonReturn to Zero Inverted
PeV: peta-elektronvolt
PID: Packet Identifier
SIE: Serial Interface Engine
SOF: Start of Frame
TeV: tera-elektronvolt
USB: Universal Serial Bus
VME bus: Versa Modular Eurocard bus

1 Inleiding

1.1 Voorstelling bedrijf: onderzoeksgroep elementaire deeltjes - UA fysica

De *Particle Physics* groep aan de Universiteit van Antwerpen doet experimenteel en fenomologisch[1] onderzoek naar botsingen van deeltjes uitgevoerd door de grootste deeltjesversnellers ter wereld. Hun onderzoek richt zich op de studie van quantumchromodynamics(QDC), het zoeken naar Higgs-boson en het zoeken naar extra dimensies van de ruimte.

De *Particle Physics* groep participeert in twee grote internationale experimentele samenwerkingsverbanden:

- Het CMS-experiment bij de Large Hadron Collider (LHC), de grootste en krachtigste deeltjesversneller ter wereld. Meer informatie kan gevonden worden in hoofdstuk 2: CMS Experiment @ UA.

- Het H1 experiment, één van de grote experimenten waarbij gegevens met botsende positron-proton bundels worden gedetecteerd in HERA. HERA of Hadron Electron Ring Accelerator was de eerste deeltjesversneller ter wereld en bevond zich in het Duitse nationale laboratorium voor fysica van elementaire deeltjes (DESY) in Hamburg, Duitsland. [17]

1.2 CERN

1.2.1 Wat is het CERN?

CERN is een Europese organisatie die fundamenteel onderzoek doet naar elementaire deeltjes. De organisatie is gehuisvest ten westen van Genève op de grens van Frankrijk en Zwitserland.

CERN was oorspronkelijk een afkorting voor *Conseil Européen pour la Recherche Nucléaire* (Europese Raad voor Kernonderzoek). Op 29 september 1954 werd de akte getekend die de raad ophief en verving door *l'Organisation européenne pour la Recherche nucléaire* (De Europese organisatie voor kernonderzoek). Niet veel later bleek dat ook deze naam de lading niet echt meer dekte (hoewel deze tot nog toe wel behouden is gebleven) en tegenwoordig wordt aan het CERN gerefereerd met: *Laboratoire européen pour la physique des particules* (Europees laboratorium voor deeltjesfysica). De naam CERN is al die jaren behouden gebleven als eigennaam voor het instituut en betekent tegenwoordig dus eigenlijk niets meer.

Bij de oprichting van CERN in 1954 waren er 12 landen lid waaronder België. Sindsdien zijn er nog enkele

Naast leden kent het CERN ook een aantal *waarnemers*. Deze staten mogen aanwezig zijn op de bestuursvergaderingen en ontvangen hiervoor de stukken, maar hebben geen stemrecht. Het betreft hier staten die geen lid kunnen of willen worden van CERN maar wel graag deel willen nemen. Naast leden en waarnemers nemen ook wetenschappers uit andere landen deel aan het onderzoeksprojecten van CERN.

[1] Fenomologie: leer van de verschijnselen

Bij CERN werken zo'n 2500 mensen voltijds. Ongeveer 8000 wetenschappers (van 580 universiteiten uit 85 landen) werken aan experimenten die bij CERN worden uitgevoerd.[3] Het wereldwijde web (www) ontstond bij CERN als een uitvinding van twee informatici, de Engelsman Tim Berners-Lee en de Belg Robert Cailliau.

Het doel van die experimenten in de deeltjesversnellers bij CERN is inzicht te krijgen in hoe de materie is opgebouwd: uit welke deeltjes bestaat de materie en wat zijn de krachten die de deeltjes bij elkaar houden. De deeltjesversneller laat deeltjes met grote snelheid (bijna de lichtsnelheid) op elkaar botsen. Tijdens de botsing vallen de deeltjes uit elkaar in elementaire deeltjes zoals quarks. Op deze manier wordt er getracht om theorieën te vinden die de vier fundamentele krachten (elektromagnetische kracht, zwaartekracht, zwakke kernkracht en de sterke kernkracht) kunnen verklaren vanuit één elementaire kracht. [19]

1.2.2 Large Hadron Collider (LHC)

Figuur 1: overzicht LHC

De Large Hadron Collider, 'grote hadronen-botser' (afgekort tot LHC), is het grootste door mensen gemaakte apparaat ter wereld en wordt gebruikt om natuurkundig onderzoek aan elementaire deeltjes te doen. De LHC ligt in een tunnel 175 meter onder de grond en heeft een omtrek van 27 kilometer. De deeltjesversneller is gebouwd door CERN en werd op 10 september 2008 voor het eerst in gebruik genomen. [21]

De LHC is voorlopig de krachtigste versneller, maar er staan nog zwaardere en krachtiger machines op de tekentafel, zoals de ILC (International Linear Collider), die ergens tussen 2020 en 2030 in gebruik moet komen.

1.2.2.1 Gebruik

Met de LHC worden protonen versneld tot 99,9999964% van de lichtsnelheid, waarna een botsing volgt. Een proton doet er ongeveer 90 microseconden[2] over om heel de hoofdring van de LHC af te leggen, dit geeft een snelheid van 11,111 kHz. De protonen worden samengebundeld in 2808[3]

[2] $\frac{\text{Ringafstand}}{\text{lichtsnelheid} \cdot 99,9999964\%} = \frac{27\,000\text{m}}{299\,792\,458\frac{\text{m}}{\text{s}} \cdot 99,9999964\%} = 90,06\ \mu s$

[3] 90 µs/25 ns = 3600. Van dit totaal aan protonwolken wordt er nog een deel weggelaten voor synchronisatie wat een totaal van 2808 protonwolken geeft.

gescheiden protonwolken per 90 microseconden. De tijdsspanne tussen 2 intervallen is 25 nanoseconden. Er zijn twee pijpen waarin de deeltjes rondgaan, de pijpen staan in tegenovergestelde richting van elkaar. Om een botsing te creëren zal de ene pijp licht afbuigen zodat beide pijpen samen lopen voor een afstand van 30 meter. Hierdoor komen beide deeltjesbundels samen en botsen ze op elkaar.

Uit die botsing proberen wetenschappers allerlei informatie te halen door middel van verschillende soorten detectors die om de buis aangebracht zijn. Aanvankelijk zullen protonen worden versneld en op elkaar gebotst worden vanuit beide richtingen met een energie van 7 TeV, samen dus 14 TeV. Ook zal er getracht worden om zwaardere deeltjes zoals loodkernen tegen elkaar te laten botsen, met energieën van meer dan 2 PeV. Het belangrijkste deeltje dat wordt gezocht is het Higgs-deeltje. Dit boson moet de verklaring geven voor de massa van deeltjes en het fundament van het standaardmodel van de deeltjesfysica vormen. Het standaardmodel voorspelt dat er bij de energieën die de LHC kan bereiken een mechanisme moet bestaan dat ervoor zorgt dat de energie van bepaalde deeltjes niet naar oneindig zal gaan. Het standaardmodel zelf heeft hiervoor het Higgs-mechanisme. Met LHC moet het duidelijk worden of dit daadwerkelijk het theoretisch standaardmechanisme is of dat er naar andere theorieën moet worden gezocht.

Wat verdere gevolgen zijn van de LHC, kan moeilijk gezegd worden. Er wordt gezocht naar bevestigingen van andere theorieën ,met name de werking van de zwaartekracht en het bestaan van mogelijke onzichtbare dimensies. Misschien kunnen er zelfs minuscule zwarte gaten mee gemaakt worden, die volgens de theorie van Stephen Hawking[4] onmiddellijk weer zouden verdampen. Dan zouden verschillende andere kosmologische theorieën getest kunnen worden. [21] [10]

1.2.2.2 Defect

Door een fout in een tweetal lassen van de elektrische verbindingen ontstond er een lek in de met vloeibaar helium gekoelde magneten eind september 2008. Hierdoor moesten 53 magneten worden vervangen. De start van de LHC liep hierdoor een vertraging op van ongeveer een jaar.

Op 20 november 2009 werd de LHC opnieuw opgestart en werden de energieniveaus van de protonwolken (deeltjes) geleidelijk aan verhoogd tot een energiewaarde van 3.5 TeV.

Door ontwerpfouten zullen de verbindingen tussen de supergeleidende magneten hoogstens het energieniveau van 3.5 TeV aankunnen. De LHC zal dus op 'halve kracht' draaien in 2010 en 2011. Eind 2011 zal de LHC voor minstens een jaar worden stilgelegd om de noodzakelijke aanpassingen aan de verbindingen aan te brengen. Pas rond 2013 of 2014 kunnen ze dan verder gaan en de oorspronkelijk bedoelde 7 TeV per bundel bereiken. [21]

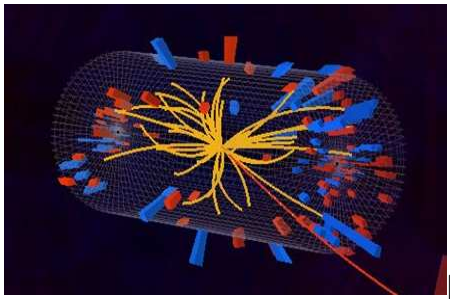
1.2.2.3 Succes

<u>30 november 2009</u>
In de nacht van zondag 29 op maandag 30 november slaagden de wetenschappers erin de deeltjes eerst met 1,05TeV in de LHC te injecteren, waarna de energie kon worden opgevoerd tot 1,18TeV. Daarmee is de vorige recordhouder, de deeltjesversneller in het Amerikaanse Fermi-lab die 0,98TeV haalde, voorbijgestreefd. [21]

[4] Stephen Hawking is een Britse natuurkundige, kosmoloog en vooraanstaand theoretisch natuurkundige

19 maart 2010

Er is een nieuwe recordenergie bereikt. De twee bundels met protonen hadden ieder een energie van 3,5 TeV. De deeltjes circuleerden in twee tegengestelde richtingen, maar er zijn nog geen deeltjesbbotsingen geweest. [21]



[5]

Figuur 2: protonen botsing in het CMS-experiment met een totale energie van 7 TeV

30 maart 2010

Op 30 maart 2010 is het voor het eerst gelukt om de twee stralen protonen van telkens 3.5 Tev op elkaar te laten botsen met een totale energie van 7 TeV (7 biljoen elektronvolt). Dit is een nooit geziene gebeurtenis en kan dus worden omschreven als een wereldrecord. Het plan is om de LHC tot eind 2011 continu botsingen te laten produceren, eind 2010 kort onderbroken door technische werkzaamheden. [15]

2 CMS Experiment @ UA

De elementaire Fysica onderzoeksgroep van Universiteit Antwerpen is nauw betrokken bij de bouw van de CMS tracker en de CASTOR calorimeter.

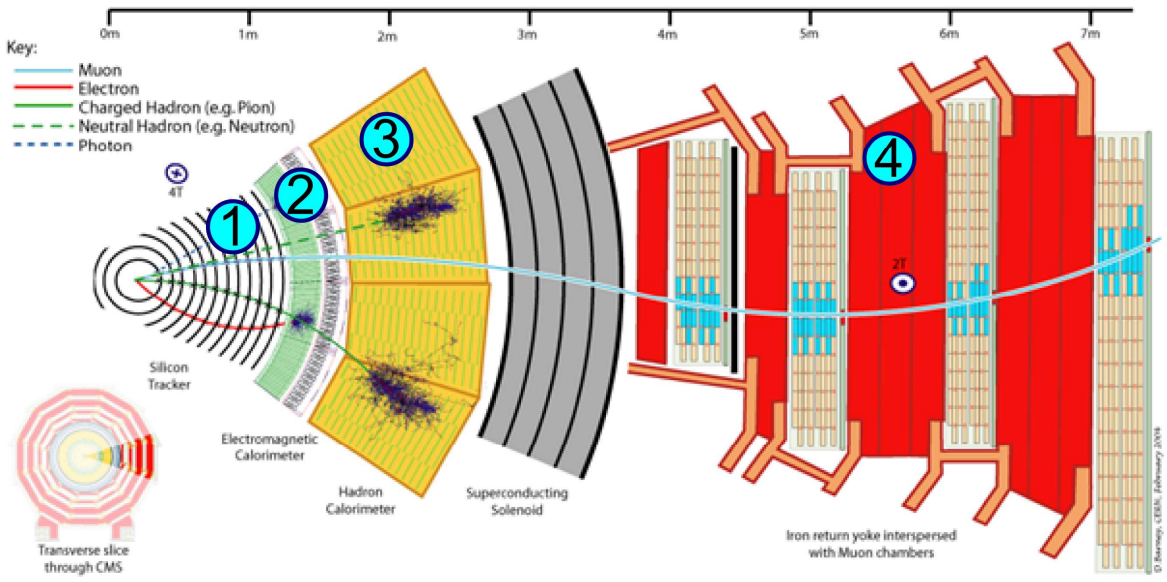
2.1 CMS

De Compact Muon Solenoid (CMS) is een van de vier grote hoofdexperimenten opgesteld rond de LHC-ring. Het is één van de twee grote algemene detectoren die rond een LHC interactiepunt zal staan om de botsingen waar te nemen. Een deeltjesdetector zoals CMS is opgebouwd in een cilindervorm en bestaat uit verschillende subdetectoren die concentrisch rond het interactiepunt zijn opgebouwd.

Een magneetveld zorgt voor de afbuiging van geladen deeltjes waaruit hun impuls afgeleid kan worden. De CMS-detector bestaat uit een supergeleidende solenoïde (= spoel) die een magneetveld van 3,8 T opwekt, met daarin subdetectoren.

De subdetectoren bestaan uit volgende delen (van binnen naar buiten) en worden weergegeven in figuur 3.
Volgende genummerde delen worden hierin weergegeven:

- Sporenkamer (=silicon tracker , 1)
- Elektromaganetische calorimeter (2)
- Hadronische calorimeter (3)
- Muonkamers (4)



[4]

figuur 3: Dwarsdoorsnede van de CMS-detector

De eerste soort detectoren die de deeltjes op hun weg tegenkomen zijn de sporenkamers. Het is de bedoeling om met deze detectoren de trajecten te reconstrueren die de deeltjes volgen. Het materiaal waaruit de sporenkamers zijn opgebouwd moet heel licht en ijl zijn, want de deeltjes mogen niet botsen en van richting veranderen, wat het opmeten van hun traject onmogelijk zou maken. De sporen worden gedetecteerd doordat de elektrische deeltjes het materiaal dat ze

doorkruisen ioniseren[5] of exciteren. Deze ionisaties en excitaties worden omgezet in een elektrisch signaal, dat door computers wordt uitgelezen en "vertaald" naar een spoor. Door het sterke magnetisch veld zullen de elektrisch geladen deeltjes in de sporenkamer afgebogen worden. Uit de kromtestraal van die gebogen baan wordt de impuls of snelheid achterhaald.

Na de sporenkamers komen de deeltjes terecht in calorimeters. Deze bestaan uit zeer dicht materiaal (bv. wolfraam of uranium, ...), zodat de deeltjes haast onmiddellijk met atomen en moleculen uit het materiaal botsen. Daaruit ontstaan nieuwe deeltjes die op hun beurt met het materiaal interageren, zodat uiteindelijk een lawine van elementaire deeltjes ontstaat. In speciale detectielagen worden de ionisaties of excitaties gemeten die de geladen deeltjes in de lawine veroorzaken. Het aantal deeltjes in de lawine staat in verband met de totale energie van het oorspronkelijke deeltje en dus kan er met dergelijke calorimeters de energie van die oorspronkelijke deeltjes gemeten worden.

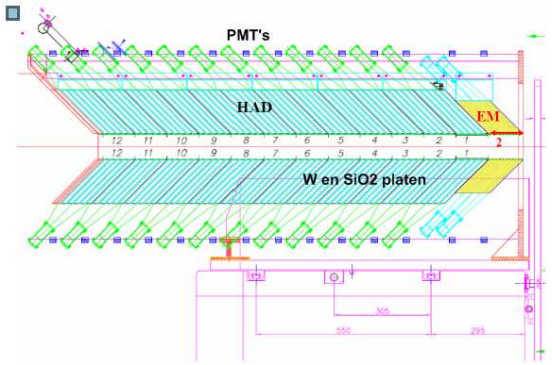
De meeste elementaire deeltjes worden volledig geabsorbeerd in de calorimeters. Er zijn nog wel deeltjes die dwars door de calorimeter gaan (muonen en neutrino's). Hierom worden er buiten de calorimeters opnieuw sporenkamers geplaatst om deze deeltjes nog te detecteren. Dit type detectoren noemen worden muonkamers genoemd.[11]

2.2 CASTOR

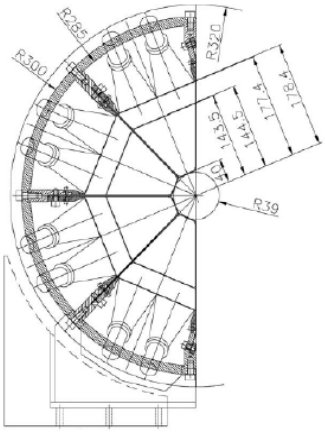
CASTOR staat voor CAlorimeter for STrange Object Research en is een calorimeter die in de lengte van de protonenbundel is geplaatst op 14.38 meter van het interactiepunt. De CASTOR is mede ontwikkeld en gebouwd door UA en zal de energie van de deeltjes meten (calorimeter). De werking is dezelfde als een gewone calorimeter. Absorptieplaten worden afgewisseld met detectieplaten. Hier bestaat het absorberend materiaal uit wolfraam en de detectie gebeurt via kwartsplaten. Een inkomend deeltje zal met de (Haevermaet, 2009) botsen en veroorzaakt zo een lawine aan elementaire deeltjes. Wanneer elektrisch geladen deeltjes doorheen het kwarts vliegen met een snelheid die groter is dan de lichtsnelheid, worden er fotonen uitgezonden, het zogenaamde Cherenkov-effect.. Dit geproduceerde licht wordt vervolgens opgevangen door licht-geleiders en naar photomultipliers gebracht. Dat zijn lichtgevoelige detectoren die een lichtsignaal omzetten in een elektrisch signaal.[9]

CASTOR bestaat uit 14 longitudinale modules die zijn opgedeeld in 16 sectoren. Iedere sector bevat op zich een uitleeskanaal, namelijk de photomultiplier tubes (PMT) wat het totaal aantal PMT's op (14 x 16 =) 224 brengt.

[5] **Ionisatie** is het proces waarbij een atoom of molecuul uit ongeladen toestand een elektron kwijt raakt of er bij krijgt; en als gevolg daarvan verandert in een ion



Figuur 4: Longitudinale doorsnede van de Castordetector

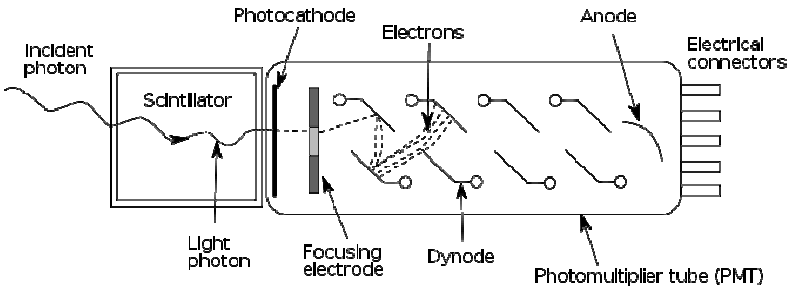


Figuur 5: Vooraanzicht van de CASTOR detector

2.2.1 PMT

Een fotomultiplicator bestaat ruwweg uit een buis met een kathode[6] (fotokathode), een anode[7] en tussenliggende dynodes. Een dynode is een van de tussenliggende elektrodes die op een spanning wordt gehouden die één stap hoger is dan de kathode en die per dynode stapsgewijs oploopt tot één stap lager dan de spanning van de anode. Met een fotomultiplicator of PMT kunnen zeer zwakke lichtsignalen of zelfs individuele fotonen worden gemeten.

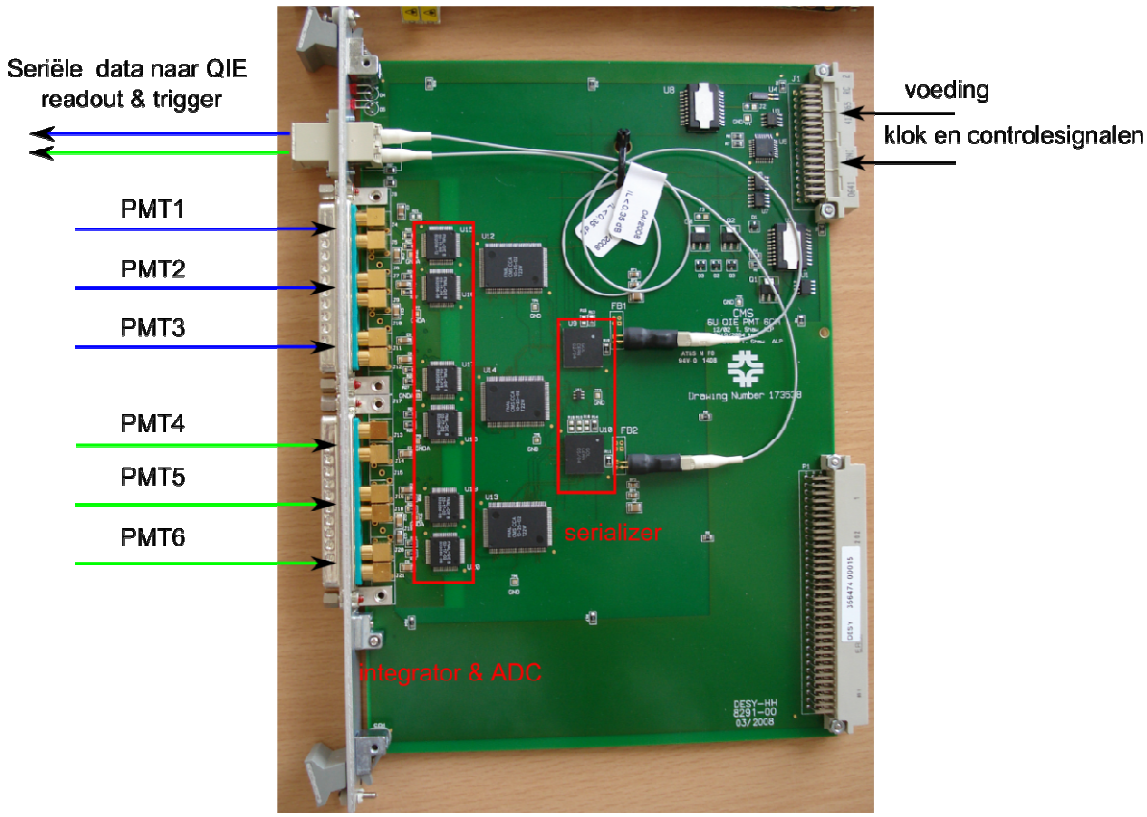
Een invallend foton maakt door het foto-elektrisch effect een elektron vrij in de fotokathode. Elke dynode heeft een hogere spanning dan de kathode, waardoor de vrijgemaakte elektronen versneld worden in de richting van de dynode. Wanneer de vrijgekomen elektronen met hoge snelheid botsen met de dynode, komen er meer elektronen vrij. Dit verschijnsel wordt secundaire emissie genoemd. De elektronen worden dan naar de volgende dynode versneld, waar het proces zich herhaalt en nog meer elektronen worden vrijgemaakt. Een fotomultiplicator bevat een aantal van deze "trappen", en bij iedere trap wordt het aantal elektronen vergroot. Met acht tot tien dynodes komen er uiteindelijk voldoende elektronen vrij om op de anode een meetbaar signaal op te leveren. De gebruikte PMT heeft 13 dynodes. [20]



Figuur 6: werking PMT

[6] Kathode is de pool waaruit de elektronen komen
[7] De anode (elektrode) of tegenpool is de pool waar de elektronen in gaan

2.3 QIE of data-aquisitiekaarten

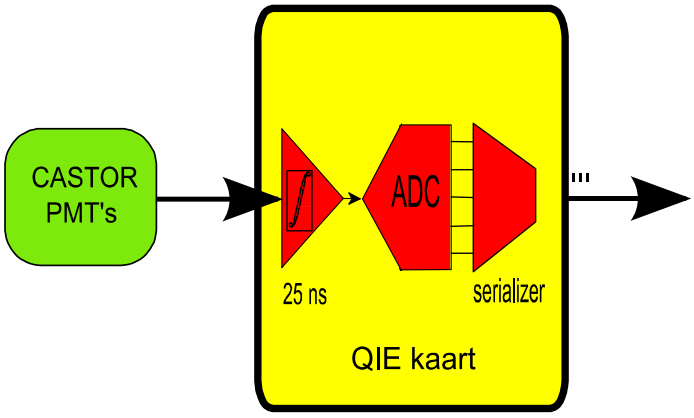


Figuur 7: afbbeelding QIE-kaart

Na de detectie d.m.v. PMT's zal de data worden verstuurd naar de QIE-kaarten. QIE staat voor Q (lading), Integration, Encoding. Iedere 25 ns wordt het stroomsignaal van de PMT's geïntegreerd, zodat alle informatie van een botsing wordt omgezet in een spanning. Vervolgens wordt deze spanning omgezet naar digitale signalen met behulp van een niet lineaire ADC. Het parallelle signaal wordt serieel gemaakt door een serializer. Deze drie stappen (omzetting, ADC, serializer) zijn de voornaamste taken van de QIE kaart. Een QIE-kaart heeft 2 fiberuitgangen en per fiber zijn er 3 kanaalingangen afkomstig van de PMT's in CASTOR. De omzetting van de 3 kanaalingangen naar de fiberuitgangen wordt uitgevoerd door een serializer die al de data serieel doorstuurt. Er worden in totaal 6 QIE-kaarten gebruikt, wat het totaal aantal uitgaande fibers op 12 brengt. De snelheid van de QIE-data wordt berekend in hoofdstuk 6.3: USB overdrachtsnelheid.
Tabel 1 toont de QIE pakketten die er worden verstuurd per fiber. Per fiber worden er data van drie PMT's verzonden (mant sect0 …).

fiber 1																												
	PMT1										PMT2										PMT3								
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
dv	ER	CapId		Exp		mant sect0 EM0					ER	CapId		Exp		mant sect0 EM1					ER	CapId		Exp		mant sect0 H0					1
fiber 2																															
	PMT4										PMT5										PMT6								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
dv	ER	CapId		Exp		mant sect0 H1					ER	CapId		Exp		mant sect0 H2					ER	CapId		Exp		mant sect0 H3					1

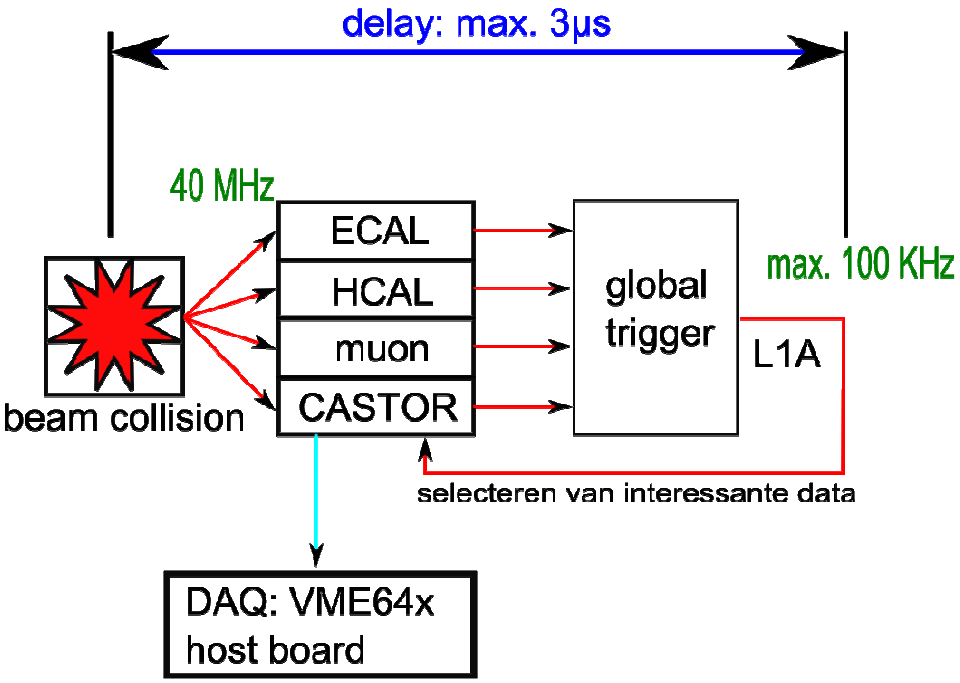
Tabel 1: QIE-data per fiber



Figuur 8: QIE-kaart met gekoppelde aan PMT's in CASTORs

2.4 QIE readout & trigger architecture

Een tweede deel bestaat uit de QIE readout en trigger[8] architectuur. Trigger logica bepaalt hier welke data interessant is om te worden doorgestuurd voor verdere verwerking.



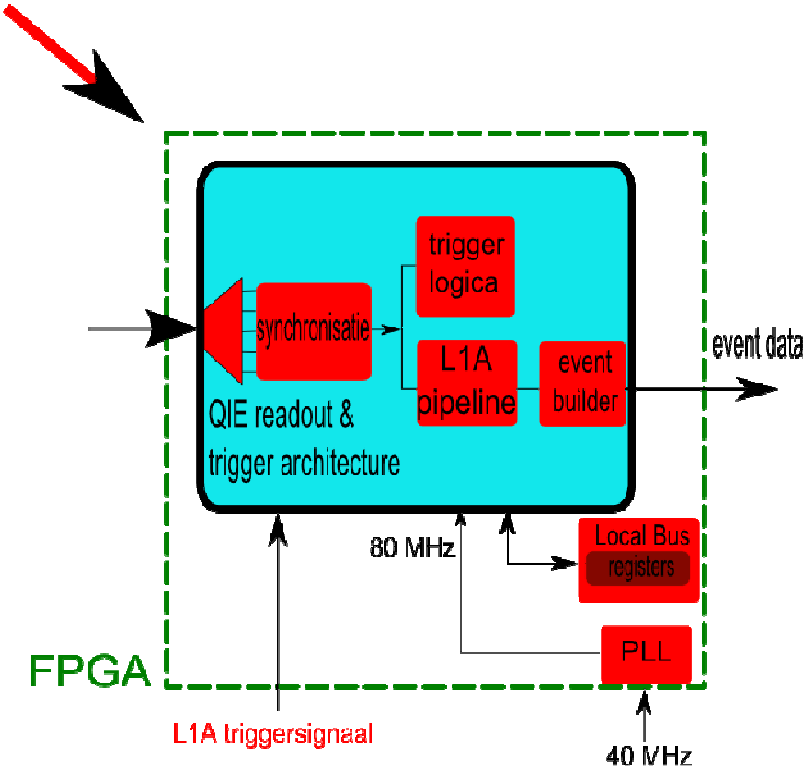
Figuur 9: selectie interessante data door selectiesignaal L1A

Iedere 25 ns (40 MHz) wordt er data afkomstig van de detectoren uitgelezen. Deze data wordt per uitleesmodule (CASTOR, Hydronische calorimeter, …) opgeslagen in een buffer (=Level1 pipeline). Tevens wordt elke 25 ns de data bewerkt en wanneer het resultaat boven een drempelwaarden uitkomt wordt er een signaal (L1) uitgestuurd naar het global trigger. Het global triggerblok bepaalt aan de hand van verschillende L1 signalen afkomstig van de detectoren welke signalen er moeten

[8] Het woord trigger is een technologisch begrip vanuit elektronica en kan het best worden omschreven als een grenswaarde (ingesteld door de gebruiker) waaraan voldaan moet worden. De trigger wordt actief bij het overschrijden van deze grenswaarde.

worden geselecteerd als interessant data. De global trigger blok geeft 'interessante' data aan met het uiteindelijke selectiesignaal L1A (Level 1 accept signal) waardoor de detectiemode (in dit geval CASTOR) weet dat het de data moet selecteren. De maximale frequentie van L1A bedraagt 100kHz. De vertraging tussen het ontvangen van de data en het uitsturen van L1A mag maximaal 3 microseconden bedragen. Momenteel wordt er vooral getriggerd op deeltjes die mogelijk door een Higgs deeltje worden geproduceerd.

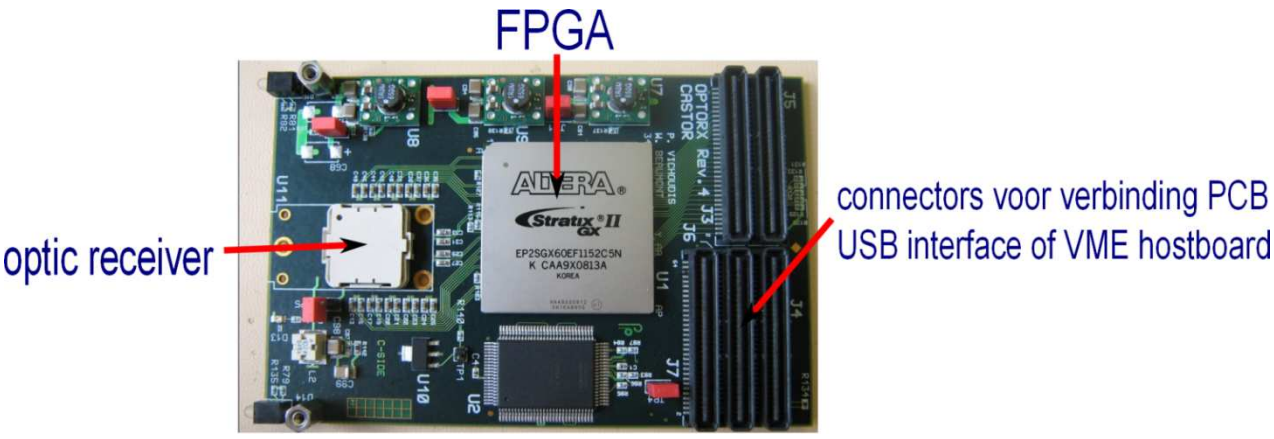
De data wordt verwerkt en getriggerd d.m.v. hardware, ontworpen met VHDL-code (trigger logica blok) om te voldoen aan de 3 µs verwerkingstijd. Het selecteren van interessante data door de QIE readout & triggerarchitectuur wordt een event genoemd en de data die door deze trigger logica wordt geselecteerd, is event-data. Event-data zal in de event builder worden gebundeld in pakketten bestaande uit header, payload (= event-data) en een trailer.



Figuur 10: QIE readout & trigger architectuur

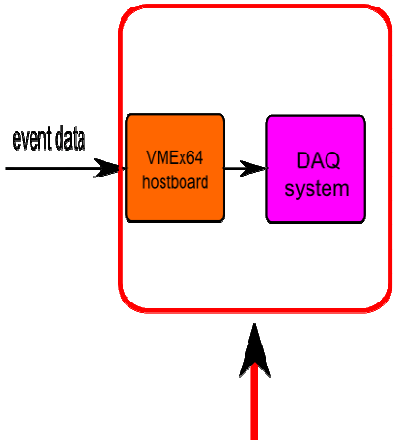
FPGA PCB: CASTOR OPTRX

Het QIE readout & trigger architectuur blok bevindt zich in de FPGA die op de CASTOR OPTRX PCB is geplaatst. Op deze PCB bevindt zich ook een optische ontvanger die de signalen van de QIE-kaarten uitleest. Verder staan er op deze PCB ook nog vijf connectiesleuven die verbinding maken met het VME Host Board (zie 2.5 VME64x Host Board voor verdere bewerking van data. De connectiesleuven worden in dit project gebruikt om verbinding te maken met de PCB waarop de USB controller staat (zie 3.1 Hardware: de PCB met USB controller: USB PCB).



Figuur 11: CASTOR OPTRX PCB of FPGA PCB

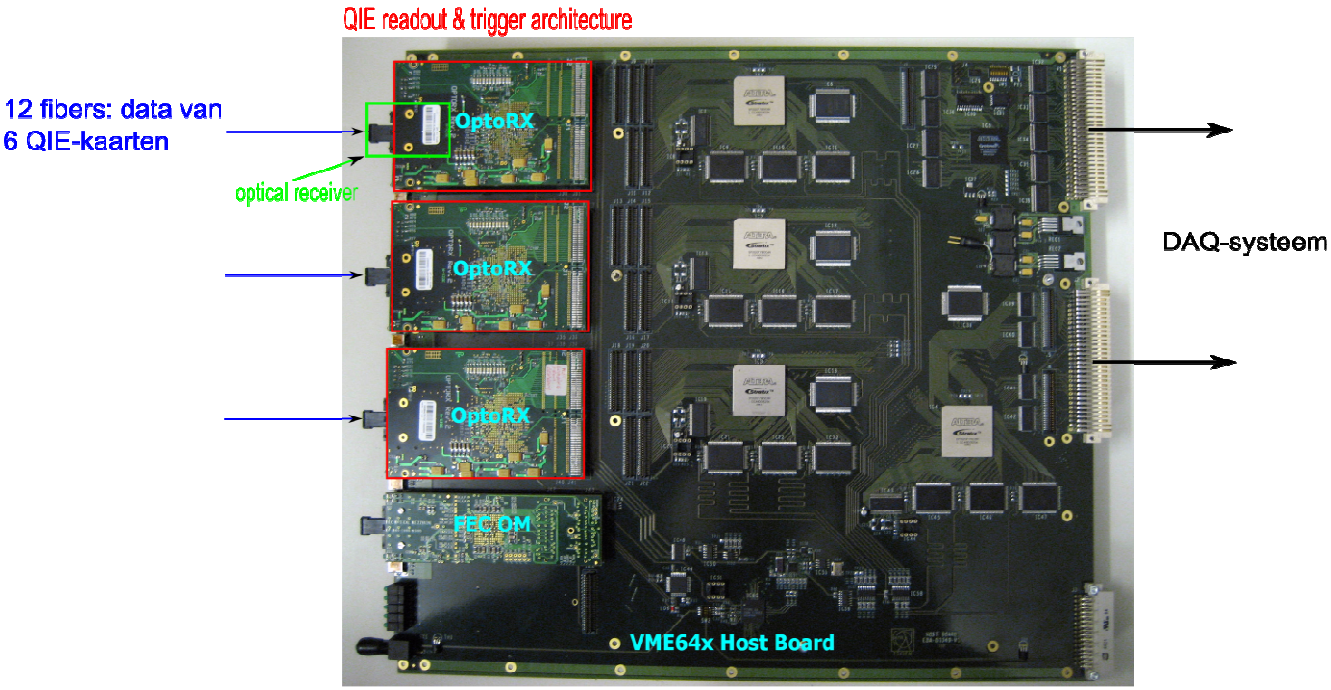
2.5 VME64x Host Board



Figuur 12: VMEx64 host board en DAQ

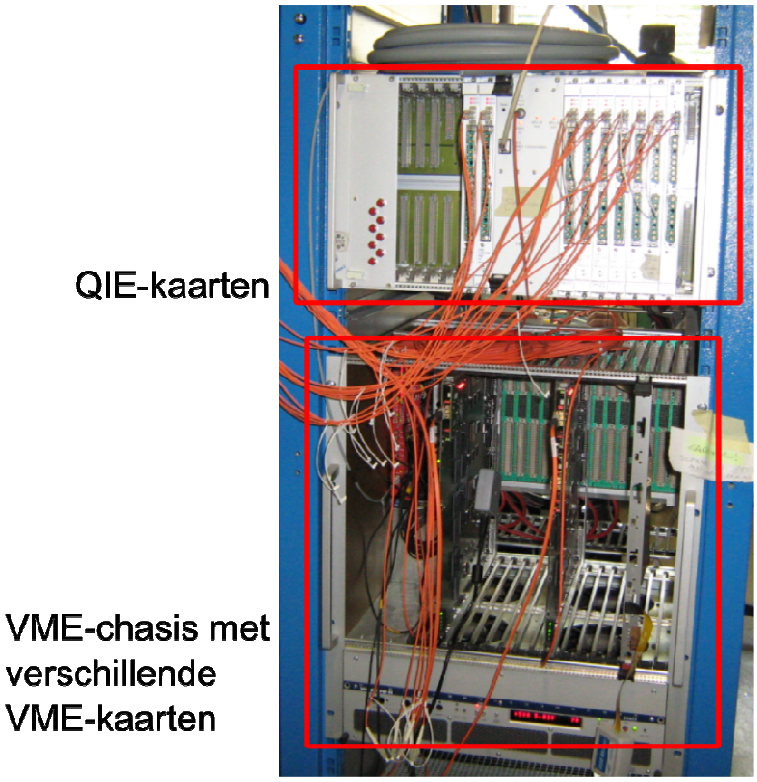
VME is een open busstandaard en wordt gebruikt voor vele toepassingen in de elektronische wereld. VME staat voor Versa Module Eurocard. De originele standaard heeft een busbreedte van 16 bit. De huidige VME64, die hier gebruikt wordt heeft een breedte van 16 bit. De bussnelheid heeft een typische waarde van 40 MB/s.[14]

Het VME64x bord is speciaal ontworpen voor uitlezing van de PMT,s. Het bord bevat vijf connectiesleuven waarop de OPTRX PCB's kunnen worden gemonteerd. De data wordt op het VME host board verder verwerkt en doorgestuurd naar het data acquisitiecentrum.



Figuur 13: VME64x Host Board

Verschillende van deze VME host boarden worden in een chasis-systeem geplaatst. Dit chasis is duur en neemt ruimte in. Om deze dure investering te vermeiden is een uitleesmechanisme ontwikkelen dat eenvoudig, voldoende snel is en weinig ruimte in beslag neemt, gebaseerd op een USB uitlezing. Nu kan lokaal het QIE readout & trigger architectuur mechanisme getest worden dat zich bevindt op de CASTOR OPTRX PCB. Dit wordt toegelicht in het volgende hoofdstuk 8 Implementatie van QIE-test.



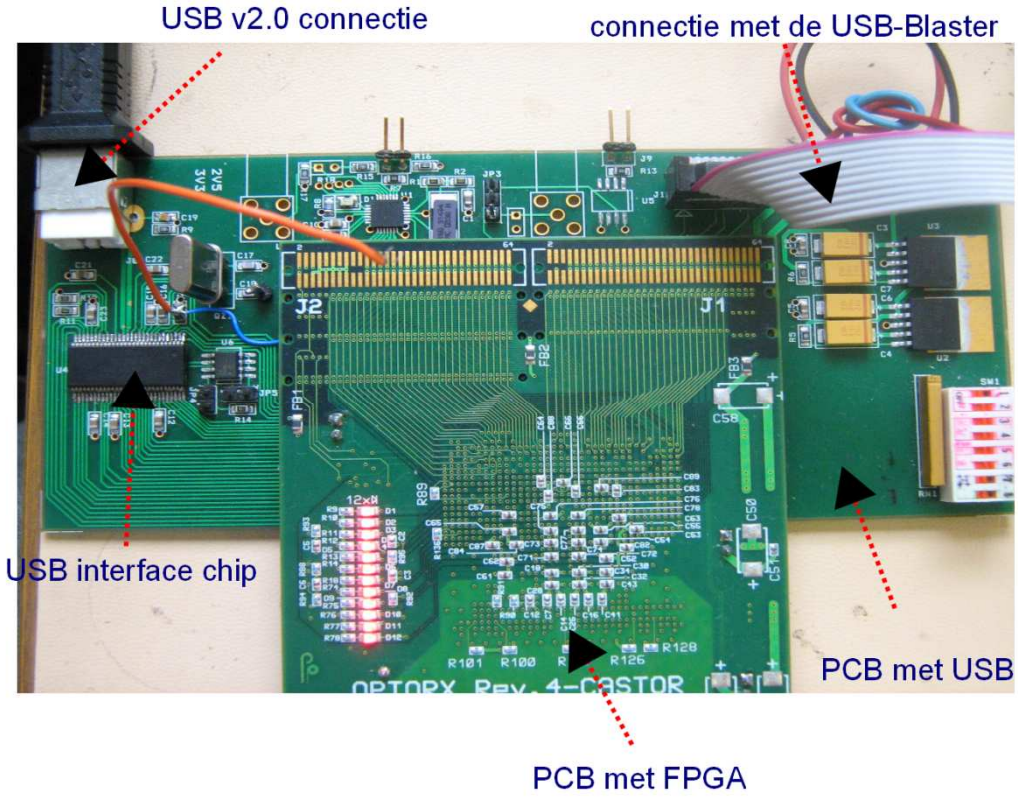
Figuur 14: : rack met QIE boven -en een VME chasis (onder) met verschillende VME-kaarten

3 USB uitlezing

3.1 Hardware: de PCB met USB controller: USB PCB

De USB PCB kan verbonden worden door middel van vijf connectiesleuven[9] met de FPGA PCB (CASTOR OPTRX PCB). Op de USB PCB staan de nodige componenten die communicatie tussen de FPGA en de USB verzorgen:

- een USB-interface chip die essentieel is voor USB communicatie, meer info kan de lezer terugvinden in hoofdstuk 5 USB-interface: cypress cy7c68001
- een USB v2.0 connectie waarop een USB v2.0 kabel kan worden aangesloten die de fysieke verbinding is met een extern toestel (in dit project: een PC met als besturingssysteem Windows XP SP3)
- een connectie met een USB-Blaster voor het programmeren van de FPGA alsook voor het uitlezen van de signalen van de FPGA, meer info in paragraaf 6.1.1.1 Signaltap II Logic Analyzer.

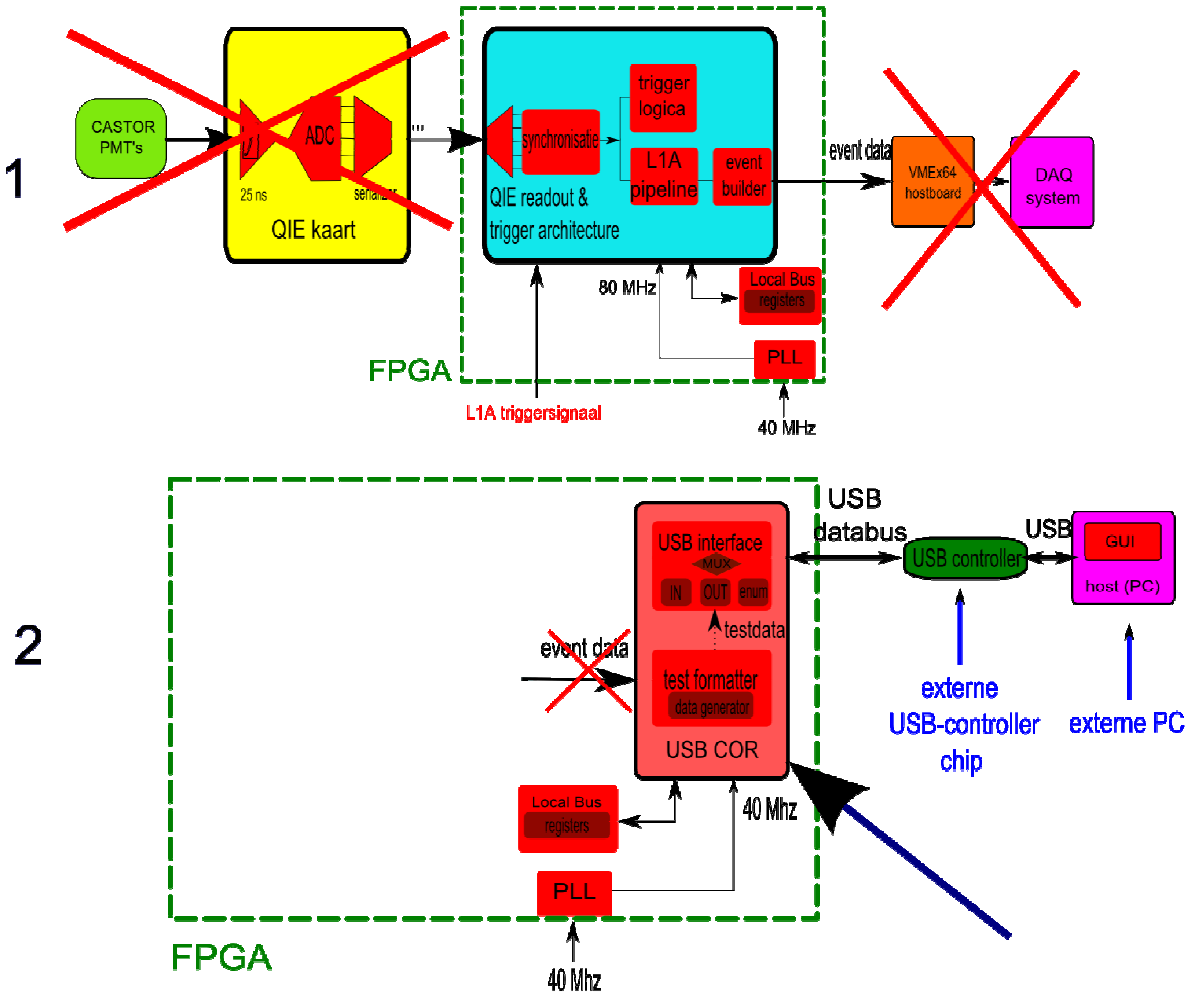


Figuur 15: USBCOR testopstelling

Deze opstelling zal de data verzenden naar een externe gebruiker via USB. Hiervoor werd de USBCOR firmware ontwikkeld, dit is een testblok waarin testdata (gegenereerd door een teller) via de USB wordt verzonden. Zo kan de volledige USB uitleesmodule worden ontwikkeld zonder hiervoor reële eventdata te moeten gebruiken van de QIE-kaarten.

[9] Connectiesleuven of 'sockets'

3.2 Firmware : het USB COR project



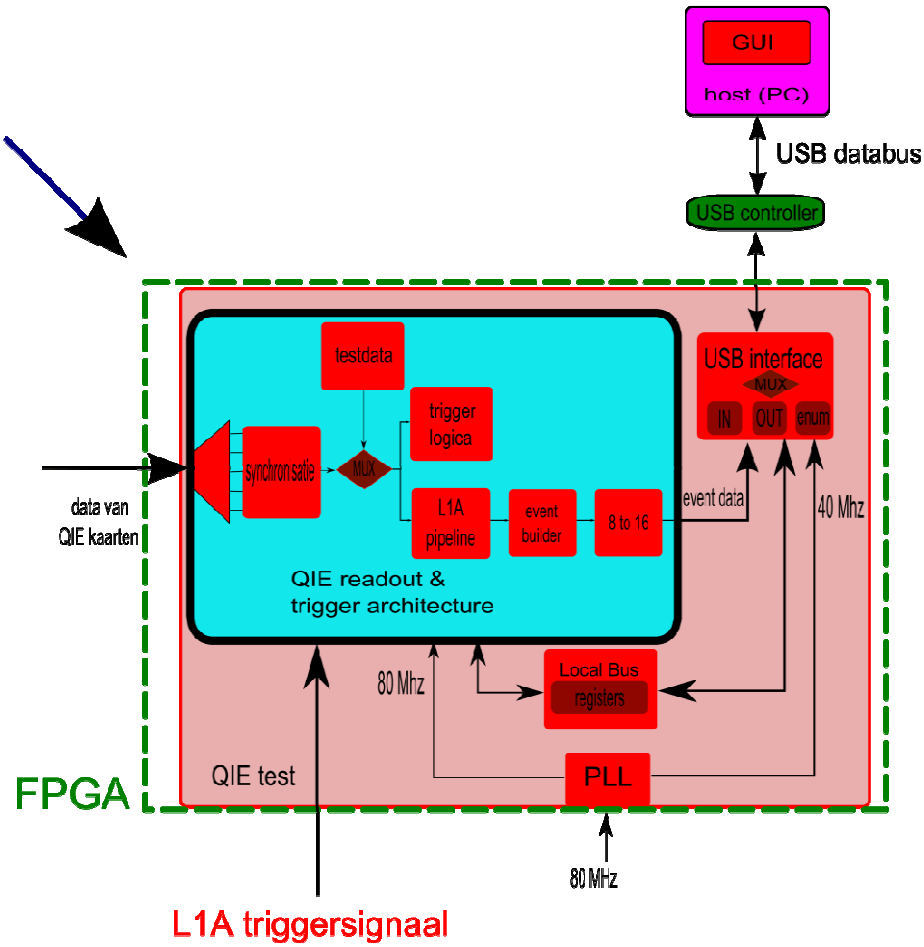
Figuur 16: 1) normale uitlezing waarbij PMT, QIE, VME en DAQ wegvallen geeft 2) USBCOR data uitlezing zonder QIE testdata te ontvangen

Bij deze opstelling wordt er enkel testdata doorgestuurd via de USB. Het doel van deze entiteit is om de instellingen die nodig zijn voor communicatie tussen USB-controller en FPGA te configureren. Er zal dus geen event data van de eventbuilder worden verzonden. Hiervoor wordt de testdata van de testformatter in het USBCOR project gebruikt. De code wordt geschreven in de FPGA. Verder wordt er nog gebruik gemaakt van een externe USB interface chip(zie ook hoofdstuk 5) en een GUI. In hoofdstuk 6: Implementatie van het USBCOR project in VHDL-code wordt het USBCOR project verder toegelicht.

3.3 Uitlezing van de USB: Grafische interface of GUI

Tot slot wordt de data ontvangen op een externe PC en opgeslagen met behulp van een overzichtelijke GUI gemaakt in Labview, zie Hoofdstuk 7 Implementatie van de GUI in Labview.

3.4 QIE-test software



Figuur 17: QIE test

Deze opstelling omvat het QIE readout & trigger architectuurblok en het USB-interface blok. Bij de opstelling van USBCOR wordt er direct data door het UBS-interface blok gestuurd. Nu zal er testdata worden gegenereerd in het QIE readout en trigger blok. Er is ook mogelijkheid om reële data te verwerken afkomstig van de QIE-kaarten. Met een multiplexer kan er gekozen worden tussen testdata of reële data. De verzonden data moet nu worden getriggerd door het L1A selectiesignaal en wordt vervolgens verpakt in de event builder of ook wel formatter genoemd. De data wordt met de USB-interface verzonden via de USB naar een GUI. Diepgaandere uitleg kan er terug gevonden worden bij hoofdstuk 8 Implementatie van QIE-test.

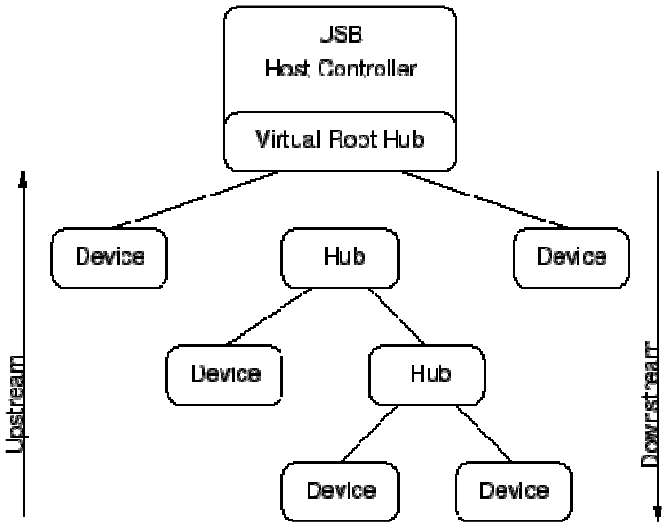
4 USB –Algemeen

USB is ontwikkeld om een groot aantal randapparaten (low & medium speed) te bevestigen aan een PC. Met een maximumoverdracht van 12 Mbits/sec voor USB v1.0 is het sneller dan de meeste seriële en parallelle poorten. Meer informatie kan gevonden worden in . In dit hoofdstuk worden enkele termen en eigenschappen van USB toegelicht die gebruikt zijn in het project.

Bijzondere aandacht werd besteed aan behoeften voor audio en video apparaten, die aan een grote opmars bezig waren bij het gewone publiek. Technische problemen zoals busafsluitingen en het toekennen van device identificatie wordt behandeld door de hardware en software architectuur zodat deze configuratie geen zorg wordt voor de gebruiker.

4.1 USB Toplogie

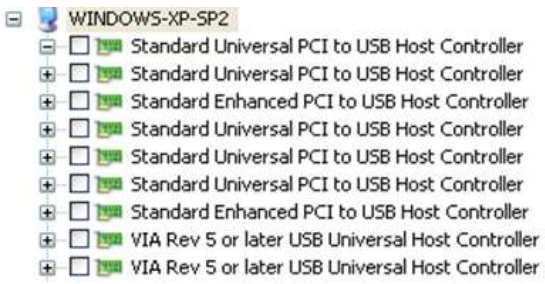
USB maakt gebruik van een meer-lagen stertopologie die lijkt op een boomstructuur. De plaats waar de bus zich in 2 of meerdere takken opsplitst wordt een hub genoemd. Een computer kan over 1 of meerdere host controllers beschikken en iedere 'host controller' bevat een USB hub, een root hub genoemd. Een root hub is een interne hub die direct is doorverbonden met de USB-controller en voorziet enkel toegangspunten, USB poorten genoemd om USB toestellen of externe hubs aan te koppelen.



Figuur 18: USB topologie

Een Host controller interface (HCI) is een interface die het toelaat dat de host controller hardware kan communiceren met het besturingssysteem van de host computer. Er zijn 3 types van HCI's:

- OHCI (Open Host Controller Interface): USB 1.0 en USB 1.1, meer hardwarematig dan UHCI.
- UHCI (Universal Host Controller Interface): USB 1.0 en USB 1.1, meer softwarematig.
- EHCI (Enhanced Host Controller Interface): USB 2.0, high, full en low speed
- xHCI (Extended Host Controller Interface): USB 3.0, super speed



Figuur 19: PC met verschillende Host Controller Interfaces

Met USB versie 1.0 en 1.1, waren er twee HCI specificaties, OHCI specificatie ontwikkeld door Compaq, Microsoft en National Instruments en de UHCI specificatie ontwikkeld door Intel. Beide HCI standaarden voorzien dezelfde vereisten en werken met alle USB-toestellen maar ondersteunen geen high-speed overdrachten. OHCI belast vooral de hardware en maakt gebruik van eenvoudige software. De UHCI zal echter gebruik maken van een eenvoudige en goedkope hardware implementatie, maar dit vereist dan wel een complexe software en dus ook meer CPU verwerking. Met de introductie van USBv2 was er een nieuwe Enhanced Host Controller Interface specificatie ontwikkeld door Intel, Compaq, NEC, Lucent en Microsoft. Deze enkele EHCI specificatie elimineert de problemen die optraden tussen de twee voorgaande concurrerende specificaties. EHCI wordt gebruikt door de USB v2.0 toestellen en is de enige HCI die high-speed overdrachten ondersteunt.

Ieder USB-toestel bestaat uit een businterface en 1 of meerdere functies. De businterface is standaard voor alle USB toestellen. Het toestel kan één of meerdere functies bevatten die softwarematig worden uitgevoerd door één of meerdere endpoints in het toestel zelf. Een toestel kan ook meerdere functies uitvoeren door een ingebouwde 'hardware' hub.
Een voorbeeld van een multifunctioneel apparaat met meerdere endpoints is een printer die kan printen, scannen en kopiëren.
Een voorbeeld van een toestel met een ingebouwde hub is een multifunctioneel toetsenbord met ingebouwde hub voor toetsenbord, numeriek toetsenbord, trackerball en externe apparatuur.

Hubs hebben poorten of bevestigingspunten wat toelaat om andere USB toestellen aan te sluiten. Iedere kabel start bij de hub en eindigt bij een ander toestel. Iedere connector is afgesloten zodat kabelafscherming automatisch wordt uitgevoerd. De gebruiker hoeft zich over deze kabelafscherming dus geen zorgen te maken. [13]

4.2 USB - Communicatie

De 'client' communiceert direct met zijn device[10]. Ieder device heeft een uniek adres dat wordt toegekend door de systeemsoftware om conflicten te vermijden.

De communicatie tussen devices en de client-software wordt opgezet door het gebruik van 'pipes'. Iedere 'pipe' is een communicatiekanaal tussen de software op de host en het 'endpoint' op het device. Ieder 'endpoint' op zich representeert dan weer een specifiek doel van het device zoals ontvangen van commando's, zenden van data. Een full-speed device heeft maximaal 16 endpoints, low-speed devices hebben maximaal 3 endpoints.

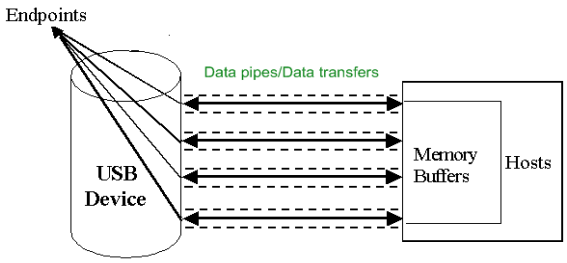
[10] Device = toestel. Meestal wordt de term 'device' gebruikt, soms de term 'toestel'.

Alle USB devices gebruiken ‘ENDPOINT 0’ bij het opstarten. Dit endpoint is het eindpunt van de standaard pipe. Nadat de aansluiting van een toestel (device) is waargenomen (plug and play), zal de USB-software ENDPOINT 0 gebruiken om het device te initialiseren, algemene configuratie in te stellen (i.e. niet device-specifiek) en informatie verkrijgen van de andere endpoints ondersteund door het device. Endpoints zijn gekarakteriseerd door hun endpoint-nummer (bepaald bij het design), bandbreedte van de bus, toegangsfrequentie, timing en het gedrag, eisen bij foutafhandeling.

USB heeft 4 transfertypes:
- control transfer: gebruikt bij command of status operations
- interrupt transfer: geïnitialiseerd door het device zelf, dat een actie verwacht van de HOST
- isochonous transfer: tijdkritische data transfers (bvb. Video en spraak).
- bulk transfer: gebruikt alle beschikbare bandbreedte, maar is niet tijdkritisch

Er kunnen verschillende pakketten verstuurd worden over alle endpoints (endpoint 0 niet): token, data, handshake en SOF pakketten. Token pakketten bepalen welk transfertype er moet worden gevolgd. Datapaketten bevatten de data of payload, handshake-pakketten worden gebruikt voor bevestiging van data (=ACK) en het rapporteren van fouten (errors). SOF of ‘start of frame’ pakketten geven de start van een nieuw frame aan. (zie bijlage F.d USB 2.0 Pakketten). [6]

Er bestaan 2 soorten pipes:
- message pipe: hierdoor worden control transfers gestuurd. De data die wordt doorgestuurd is bedoeld voor de USB systeem software.
- stream pipe: interrupt, isochroon en bulk transfers. Transport tussen client-software en het device.



Figuur 20: USB Endpoints

4.3 USB v2 – high speed

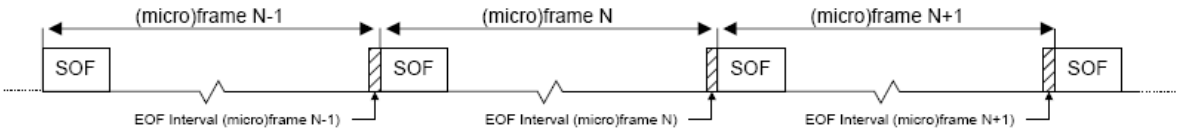
Omdat de USB-interface bij het project (cypress-cy7c68001) gebruikt maakt van USB2.0 – high speed volgt hieronder een kleine toelichting over USB 2.0.

USB 1.1 heeft een maximum snelheid van 12 Mbits/sec of 1.5MB/sec. Door de toename van flash-kaarten, grote opslagruimte HD, snellere randapparatuur werd duidelijk dat de maximale overdracht snelheid (=1.5 MB/sec) bij USB v1.1 als snel bereikt werd en dus werd deze beperkte snelheid een bottleneck. Daarmee kwam de ontwikkeling van USB 2.0 met een overdrachtsnelheid van 480 Mbits/sec of 60 MB/sec. Er moet rekening gehouden worden met het feit dat 60 MB/sec een theoretische waarde is en dat er in praktijk maximaal 40 MB/sec kan worden gehaald (Esquenet, 2006)[8] doordat er een marge ontstaat tussen het verzenden van de 'start of frame' (SOF)[11]'s (=125µs). Een pakket zal pas gezonden worden als bij een nieuwe SOF, dus om de 125 µs . Wanneer meerdere apparaten op een USB 2.0 poort aangesloten worden d.m.v. hubs zullen deze de beschikbare bandbreedte moeten delen wat ook leidt tot snelheidsvermindering. Tot slot worden alle pakketten via USB verzonden naar een software applicatie op de PC. Dit zorgt nog eens voor een vertraging door het toekennen van opslagruimte, verwerkingstijd van het pakket door de softwareapplicatie en hangt af van de CPU rekenkracht. [6]

Meer algemeenheden over USB v2.0 kunnen gevonden worden in bijlage F: USB v2 – high speed.

4.3.1 Microframes

Bij full-speed gebruikt worden er frames gebruikt, bij high-speed microframes. Deze tekst handelt over microframes. Microframes worden aangemaakt door de USB-controller van de gebruiker, waarvan het begin wordt gekenmerkt door de SOF (Start of Frame). Het interval tussen 2 SOF's is zoals al eerder vermeld 125 µs (1 ms bij frames, full-speed), dit zijn dus 8000 microframes/sec. Na het zenden van de SOF door de host USB-controller (gebruiker) kan er transport van data plaatsvinden voor een periode van 125 µs. Ieder microframe wordt gekenmerkt door een uniek ID-nummer (min. 14 bits). Wanneer er een nieuw microframe zal worden gestart zal dit ID-nummer met 1 vermeerderen. [6]



4.3.2 USB 2.0 Pakketten

Ieder pakket wordt gekenmerkt door verschillende velden. Elk veld typeert een bepaalde functie. In F.d (USB 2.0 Pakketten) kan de beschrijving van belangrijkste velden (Sync, PID, ADDR, ENDP, CRC en EOP) van het USB-pakket worden teruggevonden.

[11] SOF = Start of Frame

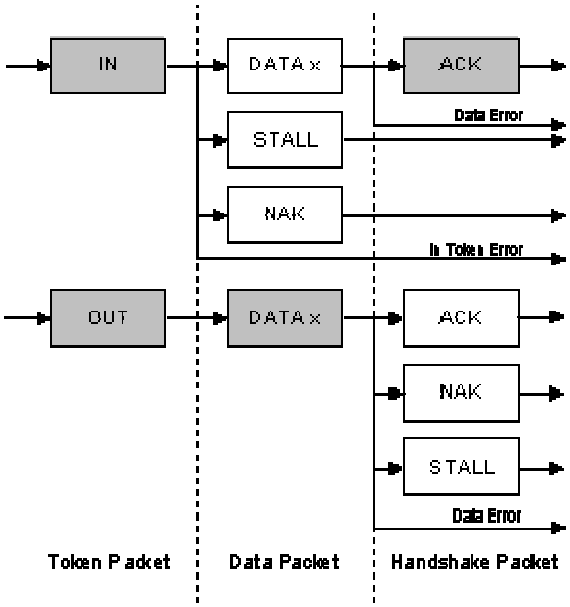
4.3.3 BULK transfers

Het bulk transfer type is ontworpen voor toestellen die grote hoeveelheden data moeten verzenden op verschillende tijdstippen waarbij het zoveel mogelijk bandbreedte gebruikt. (printer, scanner, …) Het gebruikt niet toegewezen bandbreedte die overblijft nadat alle andere transacties op de bus zijn toegewezen. Als de bus isochrone en/of interrupt transfers aan het uitvoeren is, zal de bulk transfer traag verlopen.

Bulk transfers zijn voorzien van foutcorrectie in de vorm van een CRC 16 veld op de data payload en een foutdetectie/herzend mechanisme dat de zekerheid geeft dat de data verzonden en ontvangen is zonder fouten. Een pipe in bulk transfer mode heeft volgende eigenschappen:

- De USB data verzenden en ontvangen als er genoeg bandbreedte op de USB vrij is
- Herzenden van overdrachten bij het falen of fouten van de bus
- Garantie bij het correct afleveren van de data
- Geen garantie over de beschikbare bandbreedte
- Mogelijke wachttijden

Bulk transfers zijn enkel mogelijk naargelang de beschikbare bandbreedte. Voor een USB met relatief veel bandbreedte zal de dataoverdracht vlot en snel verlopen. Bulk transfers ondersteunen enkel full of high speed toestellen. Bij full speed communicatie zijn de payload groottes beperkt tot 8, 16, 32 of 64 bytes. Voor high speed communicatie is het pakket 512 bytes groot. Een bulk transfer wordt afgehandeld als de exacte bulkgrootte is gevuld (high speed: 512 bytes). Wanneer er zich nu een payload aanbiedt met minder data, zal er een pakket worden verzonden kleiner dan de maximumgrootte (=512 byte). Er worden na dit kleinere datapakket geen volgende datapakketten meer bij het microframe gevoegd totdat er een nieuw SOF start en dus nieuw microframe wordt aangemaakt. [6]



Figuur 21: bulktransactie schema

Bovenstaande figuur toont het bulktransactie schema.

IN: Wanneer de ontvanger (PC) klaar is om data te ontvangen, zendt deze een IN Token. Als de zender dit token ontvangt met een fout, negeert hij dit pakket. Als het Token correct wordt ontvangen, zal de zender antwoorden met:

- DATA packet: bevat bulk data
- STALL packet: geeft aan dat ENDPOINT een fout bevat

- NAK: ENDPOINT is goed geconfigureerd maar heeft geen dµata om te zenden

OUT: De zender (PC-host) zal een OUT Token zenden gevolgd door een pakket dat de bulk data bevat. Het OUT token kan als volgt worden behandeld:

- foutief OUT Token: ontvanger negeert het data pakket. Als het halt-bit gezet is zal de ontvanger een STALL[12] pakket terugzenden
- ontvangstbuffer is leeg: data wordt ontvangen in de buffer en de ontvanger zendt een ACK naar de zender om de ontvangst te bevestigen
- ontvangstbuffer is niet leeg omdat ontvanger het vorig pakket nog aan het verwerken is: data kan niet worden ontvangen, de ontvanger zendt een NAK naar de zender om dit te melden

Sync	PID	ADDR	ENDP	CRC5	EOP
	8 bits	7 bits	4 bits	5 bits	

Figuur 22: USB TOKEN (IN/OUT)

Implementatie BULK transfer in het project

Voor het overbrengen van de data van de elementaire deeltjes data werd er gekozen voor een USB-FIFO van 2 maal 512 bytes. Dit geeft 1024 bytes wat voldoende is om een maximum eventpakket afkomstig van de QIE readout & trigger architectuur (zie 2.4 QIE readout & trigger architecture) van bvb. 600 bytes te kunnen verzenden. De overige 424 bytes in de ENDPOINT FIFO worden gebruikt om al een nieuw eventpakket te kunnen doorsturen, het resterende deel van dit tweede eventpakket wordt bewaard in een FIFO in de FPGA. (meer uitleg, zie 6.2.4.3 USB-IN : van PC naar USB-interface).

4.3.4 Beheer Bandbreedte

De host(PC) is verantwoordelijk voor het beheer van de bandbreedte. De bandbreedte is afhankelijk van de limitaties die bepaald zijn bij de enumeratie (isochrone, interrupt of bulk mode). Bij high speed communicatie is 80 % van het microframe voorzien voor het verzenden van data. (0.8 x 125 = 100 µs) De overige 20 % is voorzien voor control transfers. [6]

[12] Dit pakket geeft aan dat er bij het ENDPOINT een fout zit. Voorlopig is er geen datatransport op de USB.

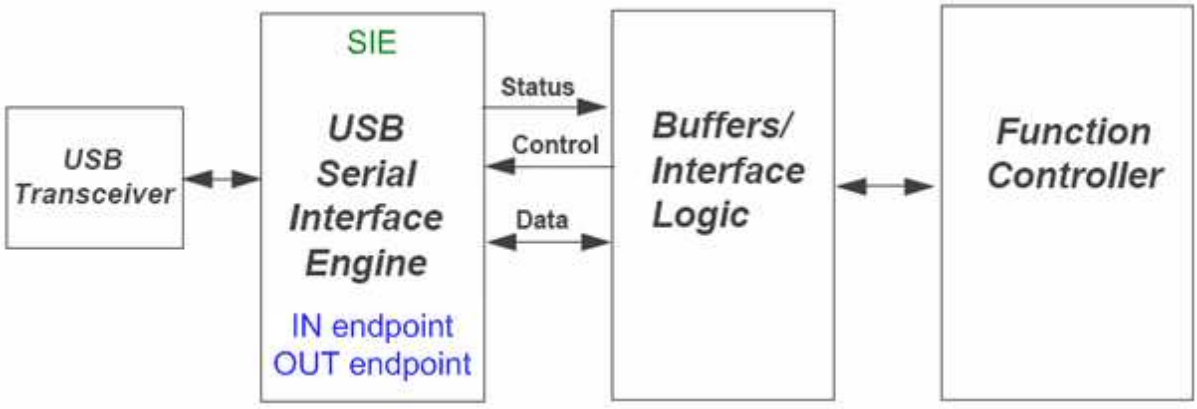
5 USB-interface: cypress cy7c68001 EZ-USB SX2

5.1 W erking USB-interface of USB-controller

Een USB I/O toestel is een combinatie van software en hardware, het dient als interface naar de reële wereld toe. Het toestel zal sensoren gebruiken om informatie zoals temperatuur, snelheid, kleur, dichtheid en grootte te bepalen. Aan de hand van deze informatie zal het toestel acties ondernemen (berekeningen, uitvoerende acties, …). Data die wordt verzameld via de sensoren(zender) wordt geplaatst in een lokale buffer, die OUT ENDPOINTS worden genoemd. De gebruiker (PC) zal deze data later gebruiken. IN ENDPOINTS zijn buffers voor data die naar buiten (PC, host, ontvanger) moet worden verzonden. Deze data wordt eveneens in een lokale buffer geplaatst. Een I/O toestel kan in theorie maximaal 16 IN en 16 OUT ENDPOINTS bevatten, maar in praktijk is dit aantal meestal minder maar moet wel minimum 1 OUT of IN ENDPOINT bevatten.

Het I/O toestel moet een CONTROL ENDPOINT of ENDPOINT 0 bevatten. Dit ENDPOINT is in tegenstelling tot de andere ENDPOINTS bidirectioneel en wordt gebruikt door het besturingssysteem om de identiteit en de mogelijkheden van het toestel te weten te komen. Dit CONTROL ENDPOINT wordt ook gebruikt voor de initialisatie van het toestel. Dit initialisatie-proces wordt in de USB-literatuur enumeratie genoemd en bevat een reeks van commando's die bij het programmeren van het toestel als eerste wordt ingelezen. Vermits het programmeren van de USB-interface van commerciële toestellen eenmalig is, zal er weinig aan deze enumeratie kunnen worden gewijzigd tenzij de USB-interface kan worden hergeprogrammeerd.

De minimale hardware componenten van de USB-controller worden in onderstaande figuur getoond. Een transceiver is vereist voor detectie van USB apparaten en USB-data. De Serial Interface Engine (SIE) is noodzakelijk voor bit timing van de bus en de omzetting van USB pakketten in geldige data-bytes voor de OUT ENDPOINTS buffers. Evenzo zal de SIE bytes in de IN ENDPOINT buffer omzetten in USB-pakketten. Buffers dienen als tussenopslagplaats voor verkregen of te zenden data. De 'Function Controller' verwerkt de data afkomstig van of bedoeld voor de USB-transceiver.



Figuur 23 USB HARDWARE interface (minimale opstelling)

Vermits de USB-interface gebruik moet maken van een CPU, wordt verondersteld dat de USB-interface de SLAVE is en de CPU de MASTER. De USB-interface zal dus reageren op requests van de CPU (MASTER). Dit kan gaan om setup request voor ENDPOINT 0 of data request voor ENDPOINT IN/OUT. De werking van de interface is vastgelegd door de MASTER. ENDPOINT IN data wordt naar

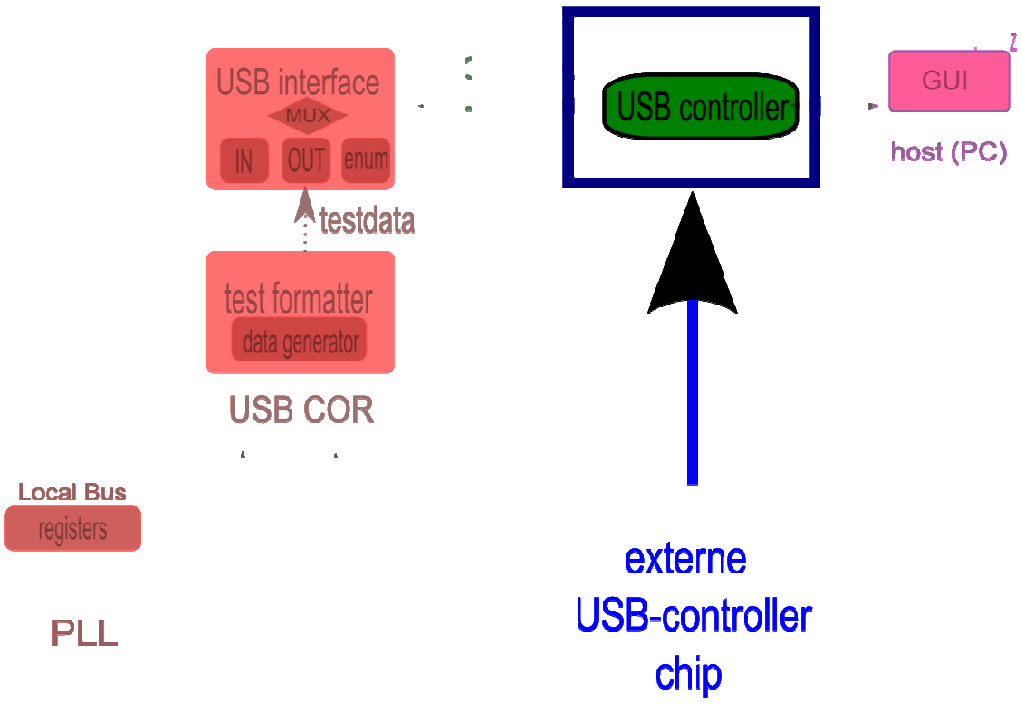
de buitenwereld gezonden en ENDPOINT OUT data wordt ontvangen en bewerkt voor ontvangst door de MASTER. Deze MASTER of CPU kan een microcontroller, microprocessor, DSP zijn of zoals in dit project geprogrammeerde logica..

De USB-interfaces kunnen worden onderverdeeld in twee groepen:
De eerste categorie omvat stand-alone interfaces die een geïntegreerde CPU bevatten. De tweede categorie omvat USB- interfaces die automatisch de USB-interacties uitvoeren, maar voor andere operaties beroep doen op een externe MASTER of CPU zodat de gebruiker zelf kan bepalen welke acties hij wil uitvoeren met de USB-interface (bvb. Cypres cy7c68001). [7]

5.2 gebruikte USB interface chip : cypress cy7c68001

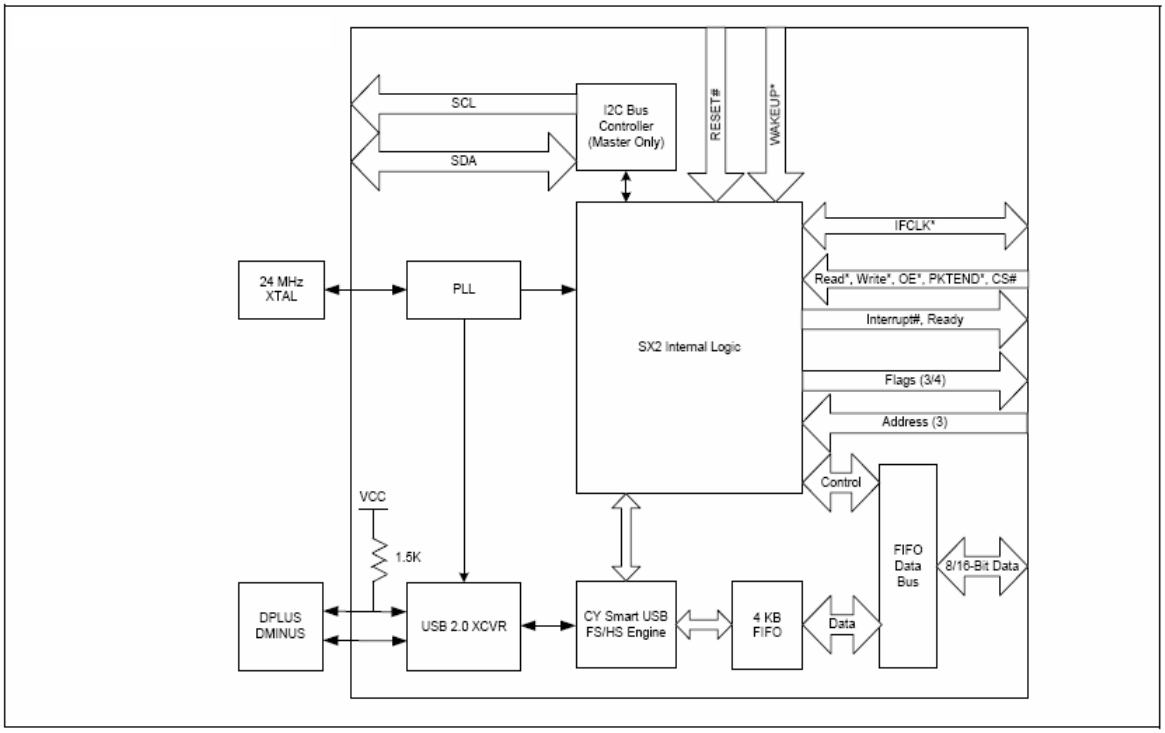
De EZ-USB SX2 USB-interface van Cypress is ontworpen om met eender welk master-toestel te kunnen werken. De SX2 voorziet de master van een USB v2.0 verbinding. De SX2 heeft een ingebouwde USB-transceiver, Serial Interface Engine (SIE) en een commando decoder, die wordt gebruikt voor het verzenden en ontvangen van USB data (= Interface Logic). De interface verzorgt ook de fysieke connectie naar de USB bus (PHY) .

Figuur 24: interfacechip cy7c68001



Figuur 25: Cy7c68001 in USBCOR

De SX2 bevat een control Endpoint (EP0), 4 gewone endpoints (EP 2, 4, 6, 8) en beschikt het over een 4 kB FIFO ruimte (dit zijn de buffers) voor flexibiliteit en vooral doorvoer van de data. De FIFO's in het project zijn dubbel gebufferd (2 x 512 bytes). Tot slot bevat de USB-interface van Cypress nog 3 adreslijnen (FIFOADR) die dienen voor de selectie van de endpoints en kan de databusbreedte (FD: 8 of 16 bit) ook worden ingesteld. Voor het project is er voor een 16-bit databus gekozen.



[7]

Figuur 26: Blokdiagram cy7c68001

5.2.1 Beschrijving belangrijkste registers, commando's en signalen

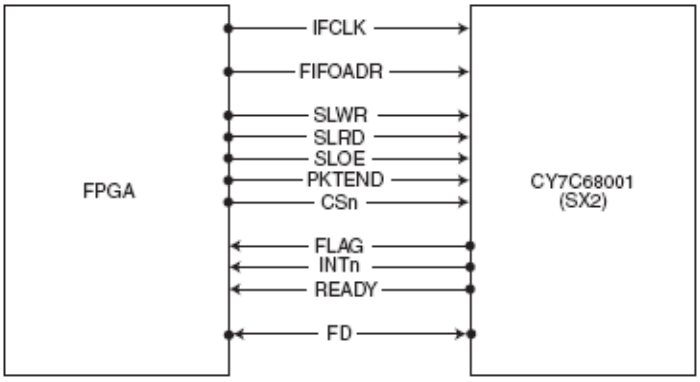
Dit wordt beschreven in bijlage G en biedt een overzicht van de gebruikte signalen, registers en commando's van de USB interface chip. Deze bijlage kan worden gebruikt voor een diepgaander onderzoek.

5.3 Communicatie met FPGA

In deze masterproef zal de communicatie tussen FPGA en USB-interface zéér belangrijk zijn. Daarom worden in dit deel van het hoofdstuk de verschillende communicatiemogelijkheden nader uitgelegd. Er zijn verschillende oplossingen om communicatie tussen FPGA en USB-interface te verwezenlijken.

5.3.1 FPGA (Altera) – USB controller

Dit is de eenvoudigste configuratie. Het doel is dat de communicatie tussen USB-controller en FPGA wordt ingesteld in bij de FPGA. De gebruiker maakt zelf code en bepaalt zelf het niveau (en dus ook de prioriteit) van deze delen. Omwille van het eenvoudig karakter is er dan ook gekozen om deze methode te gebruiken. Onderstaande figuur toont de verbindingen tussen FPGA en USB-Controller.

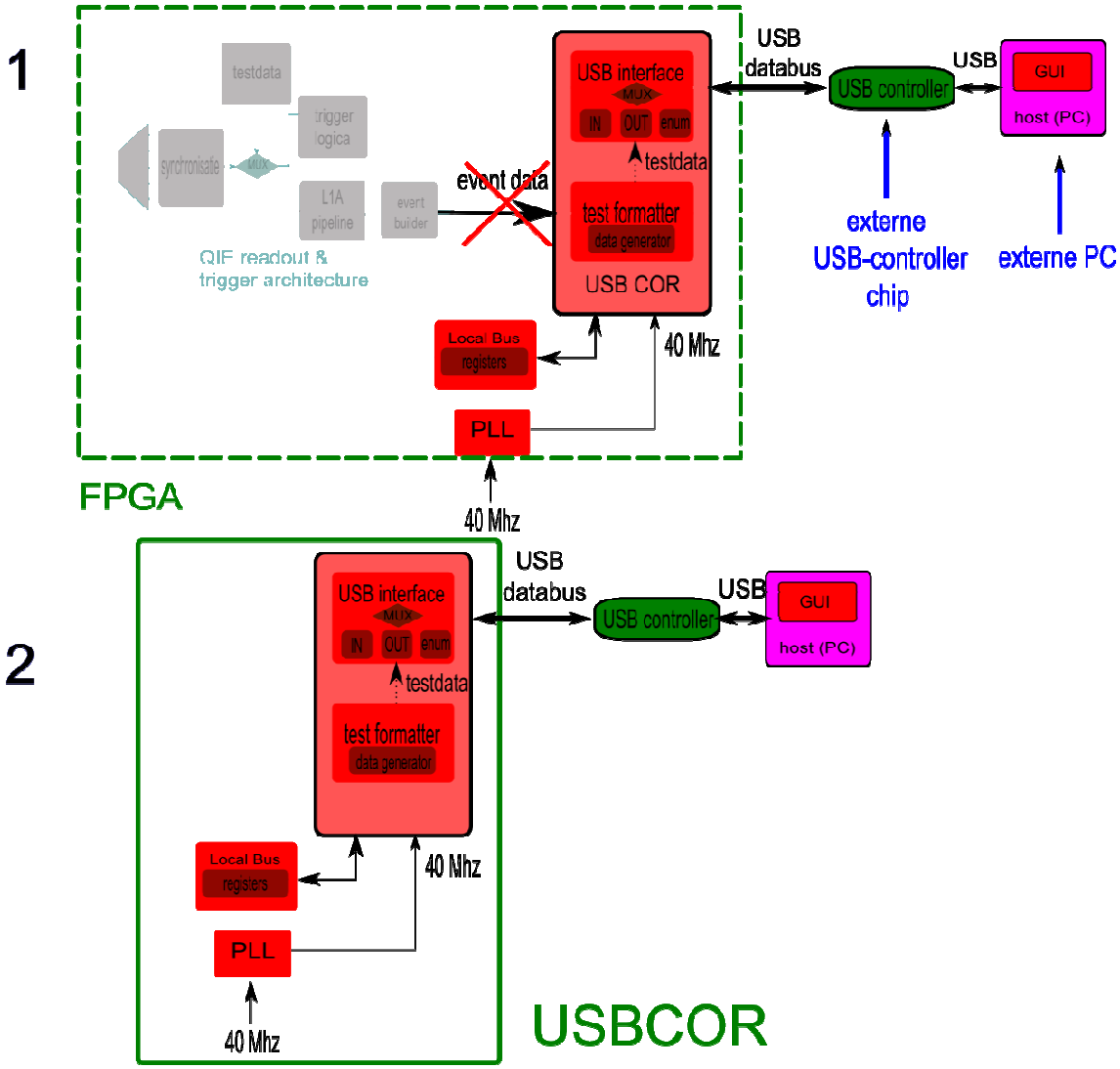


[2]

Figuur 27: verbindingen tussen FPGA en SX2

Het project in de FPGA wordt USBCOR of USB Castor optic receiver genoemd, dit is tevens de toplaag van het project. In deze toplaag zullen er 4 VHDL-entiteiten worden aangemaakt:

- USB interface: communicatie met USB, hierbij zijn er 3 taken van belang: enumeratie, data zenden en ontvangen
- PLL: geeft het kloksingaal voor alle entiteiten in het project namelijk 40 MHz
- dataformatter: genereert testdata (teller) voor het project
- verbinding met de lokale bus die wordt gerbruikt om specifieke data in op te slaan



Figuur 28: 1) gebruikte delen van USBCOR 2) USBCOR project + GUI

Hiermee is de globale structuur beschreven die wordt gebruikt in het project. De figuur toont waar het USBCOR project zich bevindt ten opzichte van de hele uitleesopstelling. USBCOR zorgt voor het overbrengen van de data naar de buitenwereld via USB v2.0. De implementatie van deze structuur kan gevonden worden in hoofdstuk 6: Implementatie van het USBCOR project in VHDL-code.

Voordeel: De programmeur begint vanaf nul te programmeren. Hij zal iedere stap die gemaakt wordt zelf hebben geprogrammeerd zodat hij weet welke functie waar zit. Dit is een voordeel bij fouten in de code. Er is enkele een nieuwe USB-controller chip nodig (de vorige was stuk, kostprijs: $11) , er was al een FPGA en USB uitlees PCB beschikbaar.

Nadeel: Het programmeren van de code vraagt veel tijd, alsook het testen van deze code.

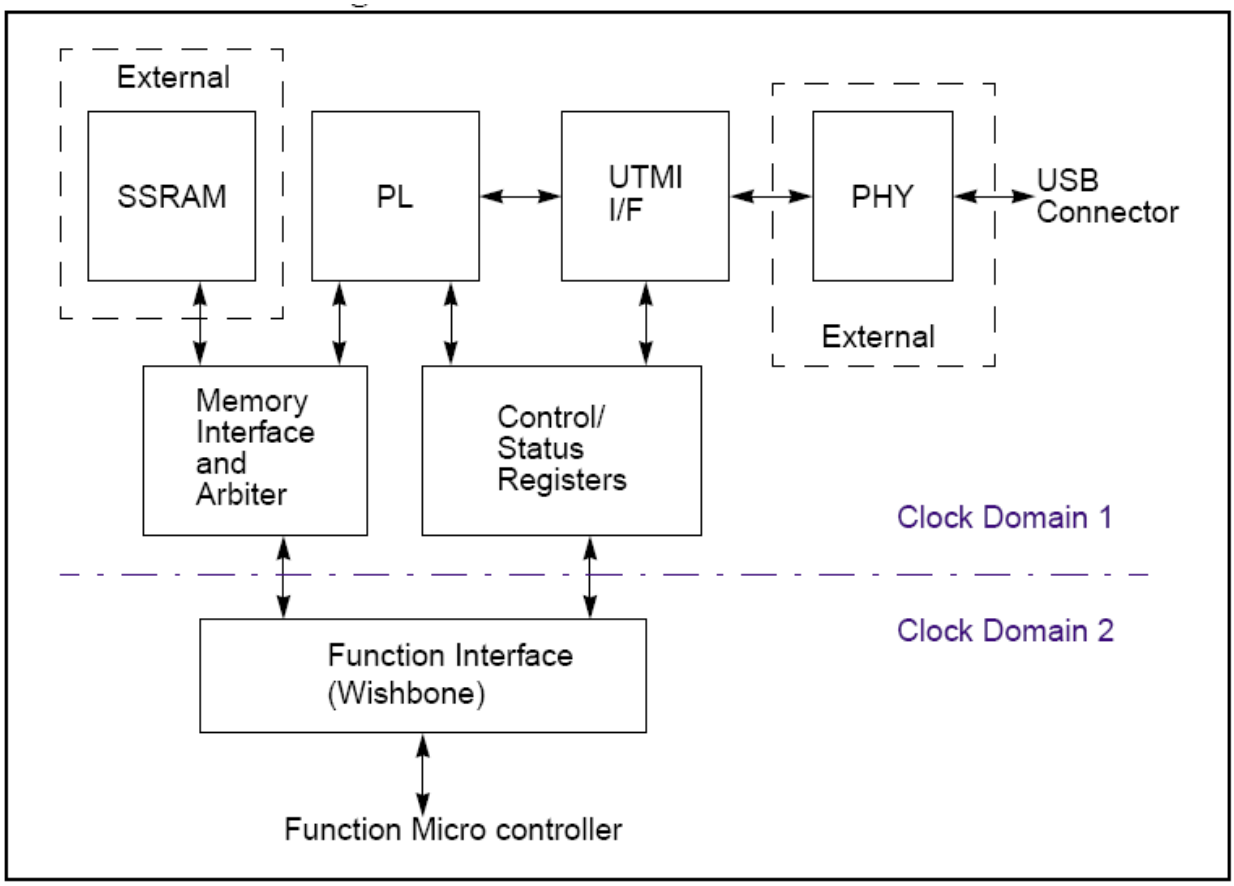
5.3.2 USB IP core

Een IP (intellectual property) core is een blok met logische data dat wordt gebruikt in het ontwerpen van een field programmable gate array (FPGA) of een Application-specific integrated circuit (ASIC) voor een product. Het IP is een essentieel element voor het hergebruik van het ontwerp. Voorbeelden van IP cores zijn Ethernet MAC's, JPEG 2000 en de Viterbi decoder. [16]

IP cores zijn relatief goedkope en krachtige oplossingen voor de eisen van de ontwikkeling van digitale ontwerpen, maar deze zijn niet gratis waardoor deze oplossing niet gekozen is. Er zijn verschillende IP cores beschikbaar op het internet:

- USB 2.0 Device IP core van Arasan (www.arasan.com/products/usb/USB_2-0_Device_IP_Core.pdf)
- USB IP cores van evatronix (www.evatronix.pl/products/product.html?id=5)
- USB 2.0 On-The-Go (OTG) IP Core (FS and HS) (www.hitechglobal.com/IPCores/usbotgfshs.htm)

Om de lezer een idee te geven van de werking van een USB IP core volgt een voorbeeld van een USB 2.0 IP core van openCores.org. De onderstaande figuur illustreert de algemene architectuur van de USB core. Het blok 'Function interface' voorziet een interface tussen de interne functie van de USB core en de functies van de microcontroller. Het 'memory interface' blok beheert het RAM geheugen voor de USB core en de microcontroller. Protocol Layer (PL) zorgt voor de interface met het UTMI interface blok. De UTMI (USB 2.0 Transceiver Macrocell Interface) is dan weer een interface met de externe USB-connector.

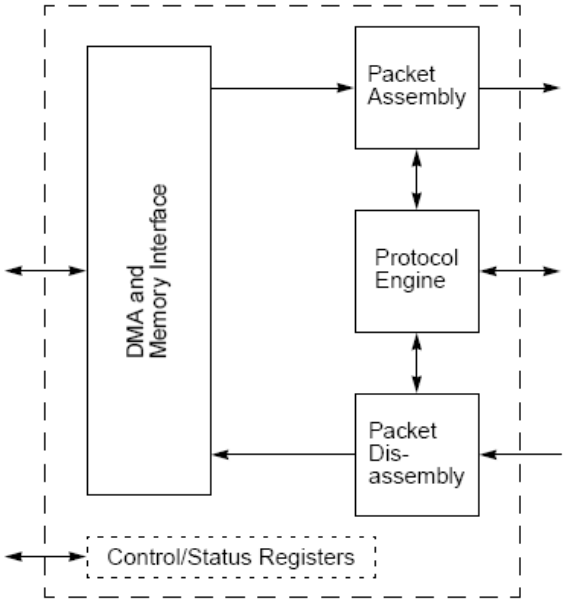


Figuur 29: architectuur USB IP core

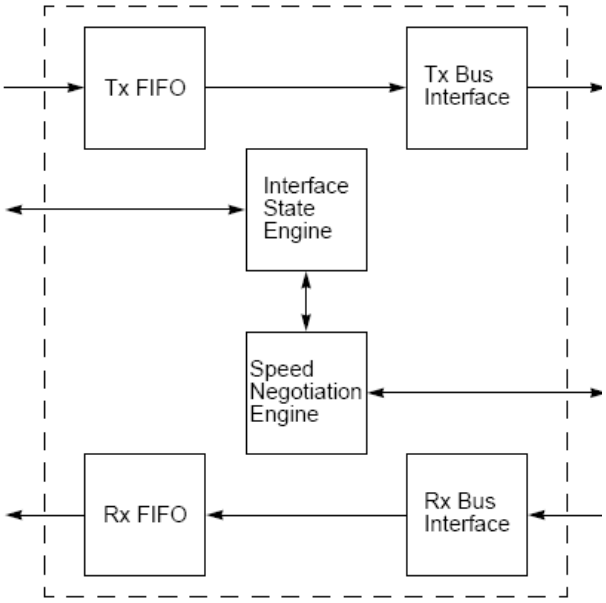
Alle taken van de USB interface (zie ook 5.1:W erking USB-interface of USB-controller) zitten nu vervat in de microcontroller zelf en niet in een externe USB-controller.

Het UTMI blok bevat een SIE en een zend –en ontvangst FIFO wat ook kan worden teruggevonden bij de externe USB-controler. Het 'speed negotation' blok beheert de USB snelheid en reset-detectie van de USB.

Verder zorgt de PL voor de verpakking van data, alsook het uitpakken van pakketten. De PL bevat ook een Protocol Engine die het alle USB protocollen afhandelt: handshake en controle protocollen. Dit zijn SOF tokens, bevestigingen van dataoverdrachten (ACK, NACK, NYET) en antwoorden op een PING singaal (zie ook bijlage F.d USB 2.0 Pakketten en hoofdstuk 4.3.3 BULK transfers).[18]



Figuur 30: Protocol Layer blok



Figuur 31: UTMU interface blok

Voordeel: de IP neemt een deel van de taken over van de USB-controller waardoor deze al niet hoeft worden aangeschaft. Er is een duidelijk overzicht van verschillende blokken. Er is ook reeds code beschikbaar bij aanschaf van een IP core.

Nadeel: Er moet nog steeds een PHY worden aangeschaft. Deze zorgt voor fysische interface tussen USB en microcontroller. IP cores zijn niet gratis. Vermist er al een bestaande PCB met USB-controller chip was werd deze gebruikt. Voor implementeren en begrijpen van de werking van een USB IP core is er ook enige tijd nodig dus werd er gekozen voor het ontwikkelen van enumeratie-code, schrijf en lees code voor de USB.

6 Implementatie van het USBCOR project in VHDL-code

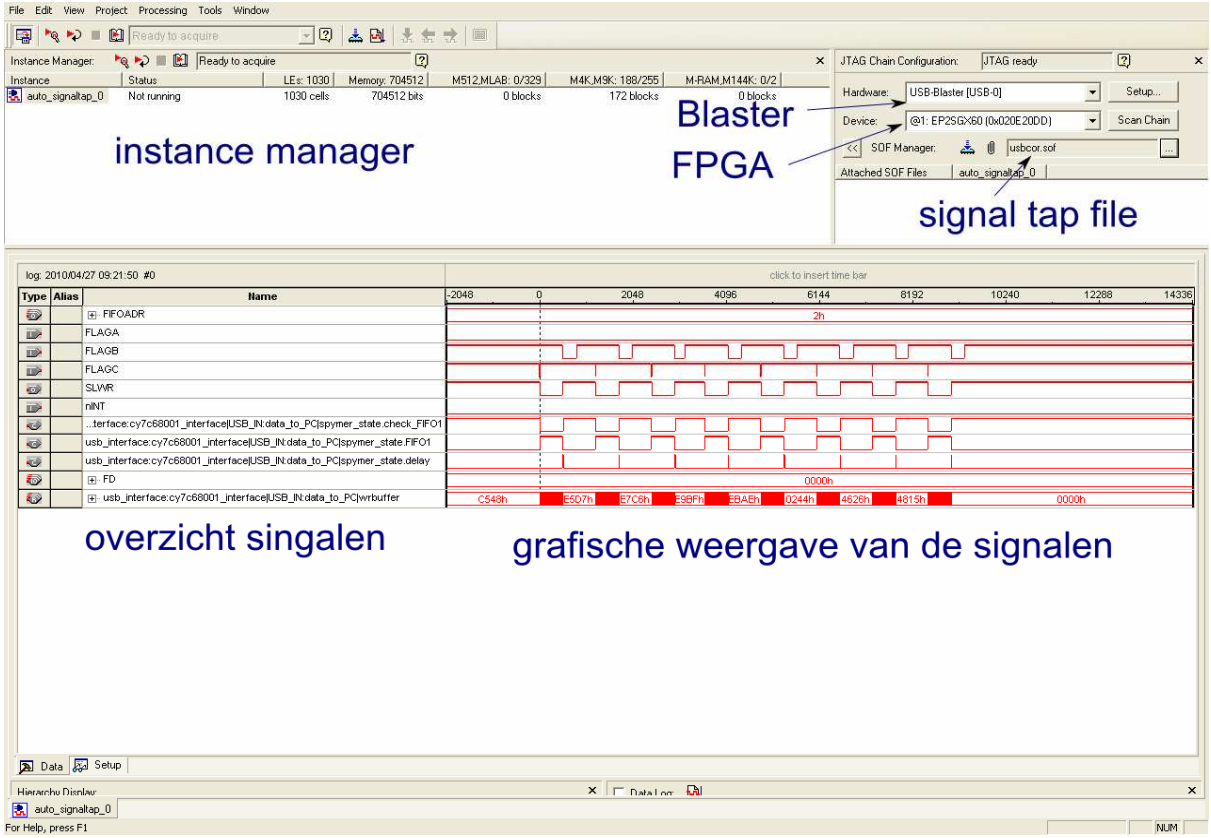
6.1 Gebruikte Software

6.1.1 Altera Quartus II 9.0

Altera Quartus II ontwerp-software biedt een design omgeving die zich aanpast aan de specifieke eisen van de designer. Het is een uitgebreide omgeving voor een system-on-a-chip programmeerbare (SOPC) ontwerp. Quartus II software omvat oplossingen voor alle fasen van de FPGA en CPLD13 en ontwerp.

6.1.1.1 Signaltap II Logic Analyzer

SignalTap II Logic Analyzer is een debug tool dat het real-time gedrag van de geprogrammeerde signalen registreert en toont. Hiermee kunnen de interacties tussen hardware en software worden waargenomen. Met Quartus II kan er gekozen worden welke signalen er moeten worden gedetecteerd. De signalen kunnen ook worden weergegeven op een externe oscilloscoop of analyser. De signalen worden gedetecteerd via een communicatiekabel met daarop een USB-blaster (gebruikt tijdens de stage), BytBlaster, MasterBlaster of EthernetBlaster. De verworven signalen worden vervolgens weergegeven in de SignalTap II Logic analyser als golfvormen (blokgolven).



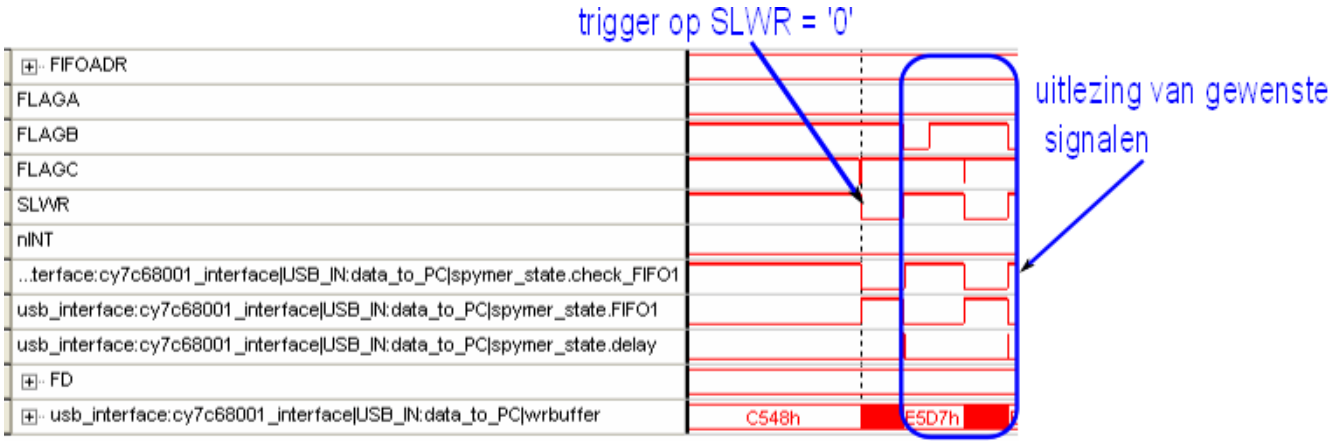
Figuur 32: grafische weergave Logic II SingalTap Analyzer

[13] complex programmable logic device

Om Logic II SignalTap Analyzer te gebruiken, moet er een SignalTap II-bestand (bestandsnaam.stp) worden aangemaakt die de configuratie-instellingen bevat en de signalen omzet in een golfvorm. De SignalTap II Logic Analyzer ondersteunt tot 2048 kanalen en 128K samples op één apparaat. Voor de stage werd er voornamelijk 16K tot 32 K samples gebruikt. Hoe groter deze waarde hoe meer geheugen het in beslag neemt op de FPGA.

Triggering

Met Signaltap II Logic Analyzer kan er op een signaal worden getriggerd, dit wil zeggen dat er gewacht wordt tot een bepaalde toestand van dit signaal wordt gedetecteerd (bvb. actief hoog, actief laag, positieve flank, negatieve flank, bepaalde waarde). Deze optie is zeer handig om de activiteit van hardware pinnen te controleren en voor debugging van het programma. Stel dat de toestand van het SLWR-signaal moet worden gecontroleerd. Door dit signaal in de singaltapfile te zetten kan de toestand van SLWR worden gecontroleerd.



Figuur 33: triggering met Signaltap

Klok

De singaltap-file zelf moet voorzien zijn van een kloksignaal die gelijkloopt over hoger is dan het interne kloksignaal van de FPGA. Zo zullen alle singalen van de FPGA kunnen worden gecontroleerd. In het project is dit kloksignaal gelijk aan 40 MHz. (= interne kloksignaal). Er is ook een kloksingaal van 200 MHz om signalen bij een groter intern kloksingaal (80 MHz bij QIE readout/trigger) te kunnen detecteren.

6.1.2 Modelsim PE Student Edition 6.5b

Deze software dient Als simulator voor de gemaakt VHDL ontwerpen. Na compilatie in Quartus II kan de code worden gecompileerd met ModelSim. Hierin kunnen er test worden uitgevoerd op de gecreëerde code. Doordat Quartus II beschikt over een eigen debugger, Signaltap analyzer is Modelsim weinig gebruikt tijdens de stageperiode bij het ontwikkelen van de programmeercode.

6.2 USBCOR

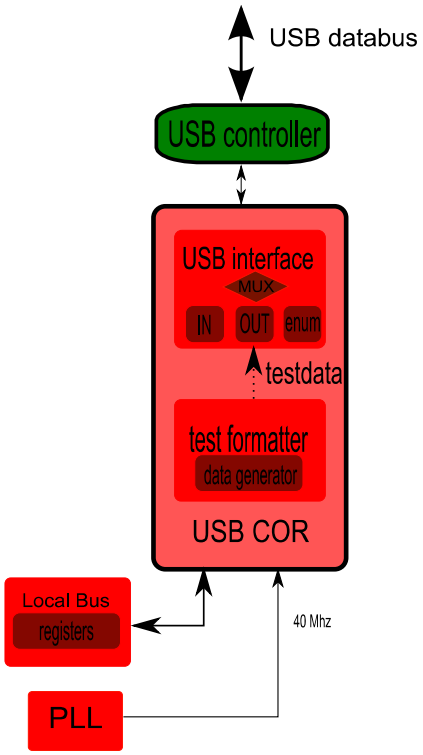
Doel: Het USBCOR-project wordt gezien als een testopstelling, dit wil zeggen dat er enkel testdata wordt verzonden en geen reële data afkomstig van de QIE-kaarten. Het doel is om deze testdata te zenden van de FPGA via USB naar een GUI op een extern systeem (hier is dat een PC). Via deze opstelling zullen de USB-instellingen (bvb. FIFOgrootte, transportmode: BULK) die worden aangebracht worden gecontroleerd. Ook zal de ontvangen USB-data kunnen worden geverifieerd worden met de verzonden testdata. Bij fouten in de ontvangen USB-data worden de USB instellingen aangepast.

Strategie:

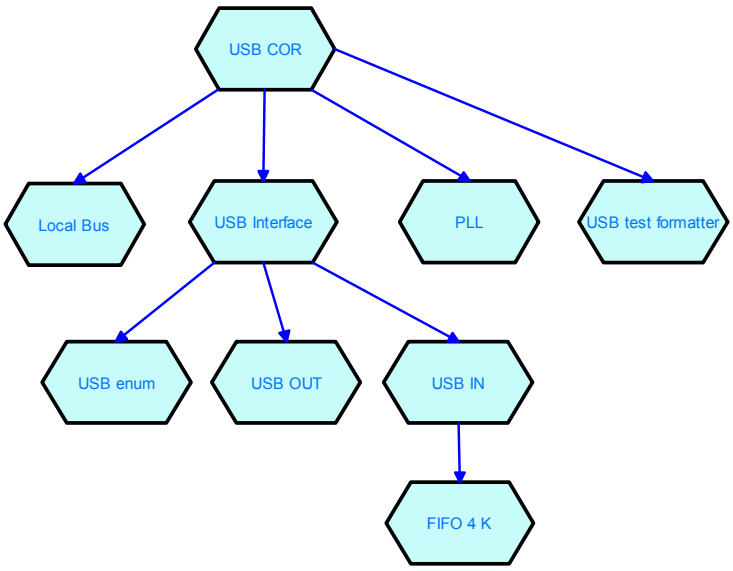
USB COR ofwel USB CASTOR OPTICAL RECEIVER is de top level code van het VHDL-project dat wordt gebruikt om de USB-interface te configureren en datatransport tussen PC en FPGA te realiseren.
In figuur 34 worden de vier delen van USBCOR weergegeven:

- connectie met LOCALBUS, extern
- connectie met PLL, extern
- USB test formatter, intern
- USB Interface, intern

USBCOR zorgt voornamelijk voor de verbinding tussen deze 4 blokken. Verder moet in de top-level entiteit ook de pinning tussen FPGA en USB-INTERFACE worden toegekend. De pinnen kunnen als ingang, uitgang of bidirectioneel geconfigureerd staan. Bij bidirectionele pinning moet er rekening worden gehouden met de bidirectionele functionaliteit van het signaal. Dit wil zeggen dat dit signaal afgehandeld moet worden met tri-sate voorwaarde. (zie 6.2.5 MUX – Multiplexer).



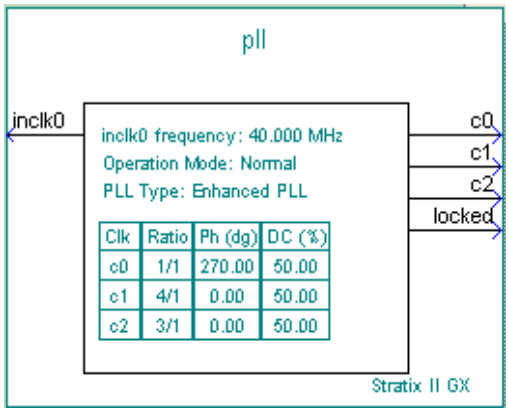
figuur 34: USBCOR project



Figuur 35: overzichtsdiagram USB COR

6.2.1 PLL

De phase locked loop voorziet het hele systeem van een intern kloksignaal(= 40 Mhz). De PLL zal zijn signaal halen van een QPLL[14] die op de PCB aanwezig is. Het voordeel van QPLL is dat deze een kleine range en lage jitter gebruikt. Deze lage jitter is van belang voor het decoderen van de snelle optische seriële signalen. In dit project wordt het 40 MHz signaal afkomstig van de QPLL (inclk0) door de PLL omgezet naar een kloksignaal van 40 (C0), 80 (C1) en 160 (C2) MHz . 40Mhz is de systeemklok voor de USB interface. De 80MHz klok wordt in de data formatter gebruikt om processen twee maal zo snel te laten verlopen. Met de Megawizard functie van Quartus II kan deze PLL worden aangemaakt en geconfigureerd.



Figuur 36: PLL configuratie screenshot

6.2.2 Local Bus

De Local Bus bevat lokale registers waarin data kan worden opgeslagen en uitgelezen. Dit wordt gebruikt om o.a. startsignalen, stopsignalen, ID's en configuratie informatie op te slaan.

De code voor de bus was al beschikbaar. De localbus communiceert met de verschillende entiteiten in het project. De local bus entiteit in het USBCOR project is dan ook een externe entiteit die niet lokaal wordt aangemaakt, maar voor verschillende projecten tegelijk beschikbaar is.

6.2.3 USB test formatter

Doel: Vermits het systeem geen gebruik kan maken van de echte data-formatter en het hier om een testopstelling gaat, zal er ook testdata moeten worden aangemaakt. Het voordeel van een test formatter is dat de uitgangsdata kan worden geoptimaliseerd om fouten eenvoudig te detecteren. Het datapakket bestaat uit payload met daarvoor een header en een trailer. De test data-pakketten worden via de USB-interface verstuurd naar de GUI op de PC.

De payload van het pakket bestaat uit een eenvoudige oplopende teller zodat het gemakkelijk is om data die verzonden is door de USB-interface te controleren op eventuele fouten (bitfouten, timing, herhalingen van bits). Door middel van de testdata uit te lezen via de USB v2.0 aan de GUI en deze data te vergelijken met de overeenkomstige gegenereerde testdata kunnen er eventuele fouten in de datastroom worden opgespoord.

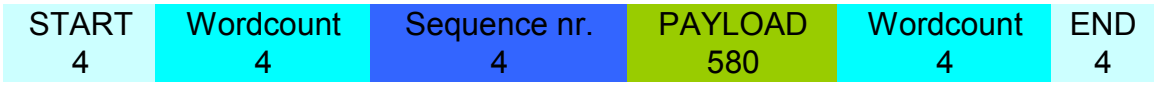
[14] Quartz Phase Locked Loop

Testdata gegnereerd door de testformatter	Uitgelezen data in de GUI	
AAAA	BAAA	Fout in MSB nibble
0121	2101	Byteswapping
5555	5555 5555	Herhaling van 16 bit woord

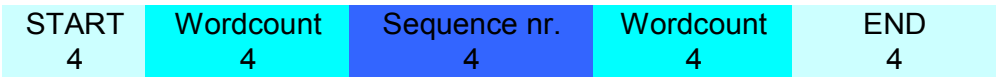
Figuur 37: voorbeelden van mogelijk fouten in de datastroom

Strategie:

Er zijn twee soorten pakketten: een volledig pakket en een leeg pakket. Het volledige pakket zal bestaan uit header, payload en trailer terwijl het lege pakket bestaat uit enkel een header en een trailer. Volle pakketten zullen worden verzonden wanneer er nog voldoende ruimte aanwezig is in de tussenliggende buffer (anders genoemd de FIFO in USB IN). Waneer er echter geen beschikbare ruimte meer is in deze FIFO, zal het systeem enkel lege pakketten versturen om dataoverhead te reduceren. Wanneer dan het PAYLOAD-veld wordt aangeroepen zal er geen data naar de tussenliggende FPGA FIFO worden verzonden.



Figuur 38: volledig pakket (veldtype en n bytes)



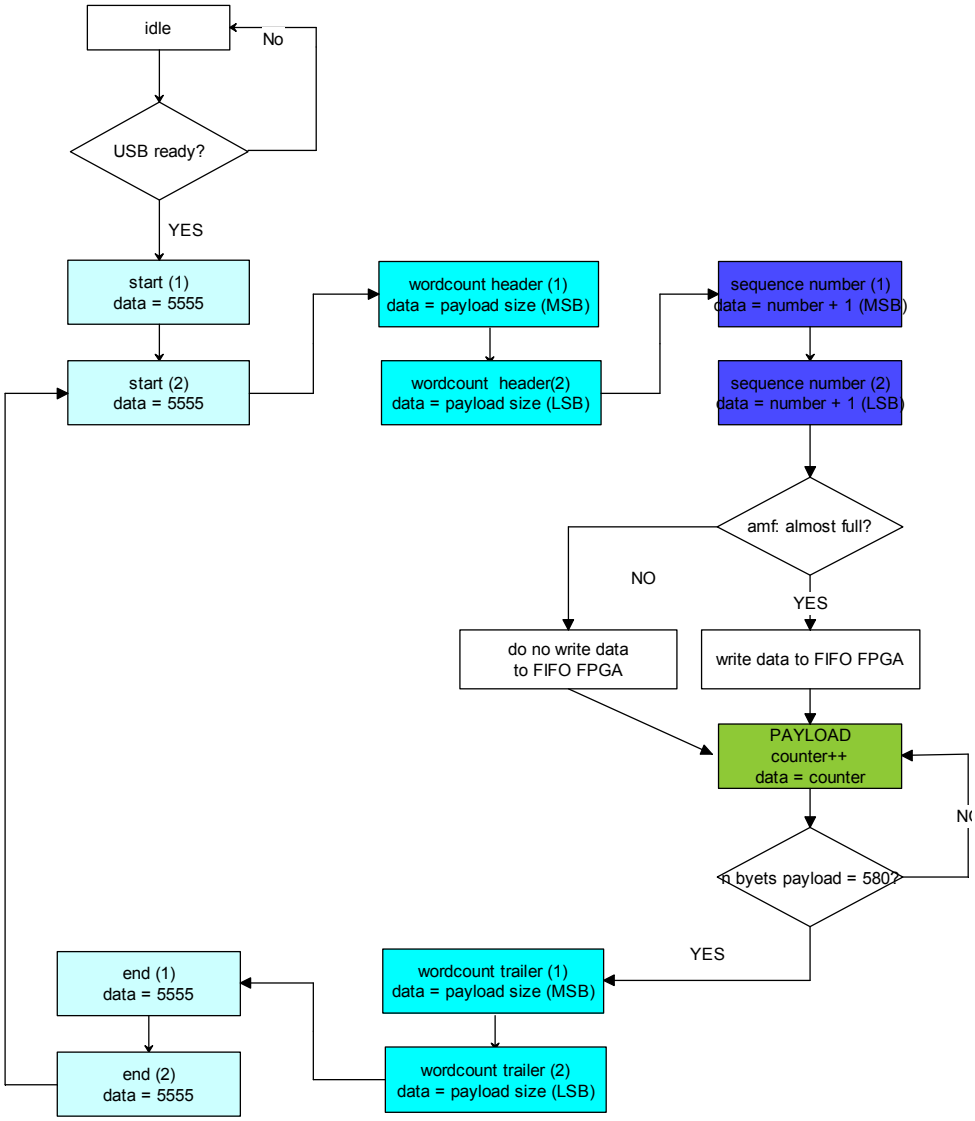
Figuur 39: leeg pakket (veldtype en n bytes)

De verschillende velden bevatten specifieke data:

VELD	DATA	INFO
START	5555 5555	deze sequentie geeft de start van het pakket weer
WORDCOUNT	600	Bevat de grootte van het pakket (typ. 600 byte)
SEQUENCE NUMBER	Number	ieder pakket wordt genummerd, een teller zal aan ieder nieuw pakket een nummer toekennen.
PAYLOAD	Counter	de eigelijk data afkomstig van eem oplopende teller
END	AAAA AAAA	deze sequentie geeft het einde van het pakket weer

Tabel 2:USB test pakketvelden en hun functie

Het finite state machine van de test formatter zal bij ieder veldtype de gewenste data aanmaken en doorsturen naar de USB IN. In onderstaande flowchart is het finite state machine van de USB test formatter weergegeven. Nadat de USB-controller gereed is zal de start sequentie, payload grootte en het sequence nummer worden verzonden. Vervolgens wordt de payload doorgestuurd (teller). Indien het signaal buffer allmost full (amf) actief is wordt er een “empty event “ gestuurd. Dit wil zeggen dat de FIFO in de FPGA bijna volledige vol is en dus geen volledige pakketten met payload kan opslagen. Na de payload wordt de payload grootte nog eens herhaald gevolgd door een eind sequentie.



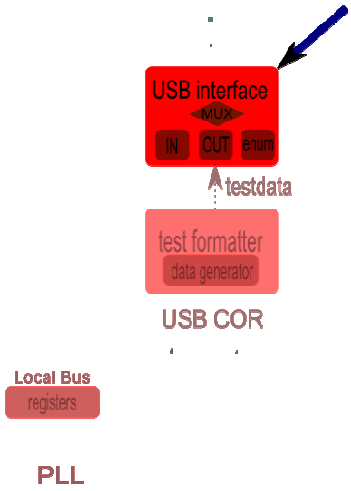
Figuur 40: Flowchart USB test formatter

6.2.4 USB interface

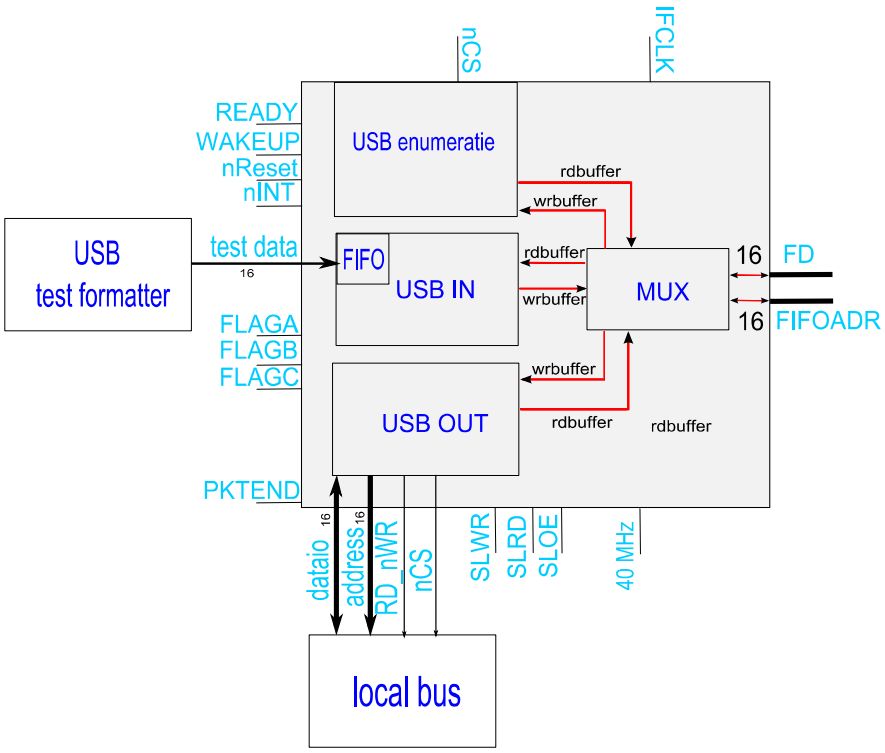
De entiteit USB interface bestaat uit 3 kleinere entiteiten en 1 proces:

- USB enum : enumeratie van de USB-interface
- USB OUT : verwerken van binnenkomende signalen in de USB-interface
- USB IN : verwerken van uitgaande signalen uit de USB-interface
- MUX (proces): multiplexer die FD-bus en FIFOADR met de juiste entiteit verbind

Dit blok zal voor de verbinding zorgen tussen het systeem en de buitenwereld (GUI van de host-PC). Bij het schrijven van de code werd er rekening gehouden met het feit dat dit blok snel moet werken en werd er ook getracht om zo weinig mogelijk tussenstappen te moeten uitvoeren tussen de blokken onderling. Bijvoorbeeld extra tussenliggende signalen of buffers tussen 2 entiteiten zorgen voor een vertraging van de data-uitwisseling tussen 2 blokken.



Figuur 41: positie USB-interface in het USBCOR project



Figuur 42: USB-interface

6.2.4.1 Enumeratie – initialisatie van de USB-controller

Doel: Het toekennen van bepaalde eigenschappen en functionaliteit aan de USB-controller chip (groen blok in Figuur 41) Deze initialisatie van de USB-interface wordt ook wel enumeratie genoemd. De enumeratie moet plaatsvinden voor er datastromen worden getransporteerd. Het is de eerste actie die wordt ondernomen bij het programmeren van de SX2-chip. Dit is een eenmalige actie en wordt uitgevoerd wanneer de USB-controller van spanning wordt voorzien. Door het heropstarten van de chip (even geen spanning en dan opnieuw spanning) zal de enumeratie terug worden uitgevoerd. Volgende eigenschappen van de USB-controller worden met enumeratie bepaald:

- het transfertype van de USB-verbinding: CONTROL, INTERRUPT, ISOCHROON of BULK
- de richting van de tussenliggende FIFO's: IN of OUT
- de buffergroote van de USB-FIFO's: bvb. 1024 bytes voor isochroon, 512 bytes voor bulk transport
- de hoeveel buffers de FIFO bevat: DOUBLE, TRIPLE, QUAD
  totale FIFO-grootte = buffergroote X buffertype
  bvb. Een FIFO met buffergrootte 512 byte, double buffered wil zeggen dat de FIFO bestaat uit 2 buffers van 512 bytes en geeft een totale FIFOgrootte van 1024 byte
- de functie van de FLAGS bepalen: de standaard functionaliteit van een FLAG wijzigen in een functie gedefinieerd door de gebruiker
  bvb. De empty-toestand van 'FIFO 1' moet worden gecontroleerd terwijl de full en empty toestand wordt gecontroleerd van 'FIFO 2'. FLAGA kan worden ingesteld als full flag van 'FIFO 1' en zo wordt deze toestand gecontroleerd terwijl 'FIFO 2' is geselecteerd via FIFOADR.

FLAGSAB								0x02
Bit #	7	6	5	4	3	2	1	0
Bit Name	FLAGB3	FLAGB2	FLAGB1	FLAGB0	FLAGA3	FLAGA2	FLAGA1	FLAGA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

FLAGx3	FLAGx2	FLAGx1	FLAGx0	Pin Function
0	0	0	0	FLAGA = PF, FLAGB = FF, FLAGC = EF, FLAGD = CS# (actual FIFO is selected by FIFOADR[2:0] pins)
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	Reserved
0	1	0	0	EP2 PF
0	1	0	1	EP4 PF
0	1	1	0	EP6 PF
0	1	1	1	EP8 PF
1	0	0	0	EP2 EF
1	0	0	1	EP4 EF
1	0	1	0	EP6 EF
1	0	1	1	EP8 EF
1	1	0	0	EP2 FF
1	1	0	1	EP4 FF
1	1	1	0	EP6 FF
1	1	1	1	EP8 FF

[7]

Tabel 3: registers die de functionaliteit van de FLAGs bepalen

Bovenstaande opgesomde lijst bevat de meest gebruikte functionaliteiten, vooral bepalend voor de USB-FIFO. Er zijn nog andere functies maar deze zijn niet noemenswaardig. De enumeratie verloopt via het CONTROL ENDPOINT, selecteerbaar met FIFOADR.

Strategie:

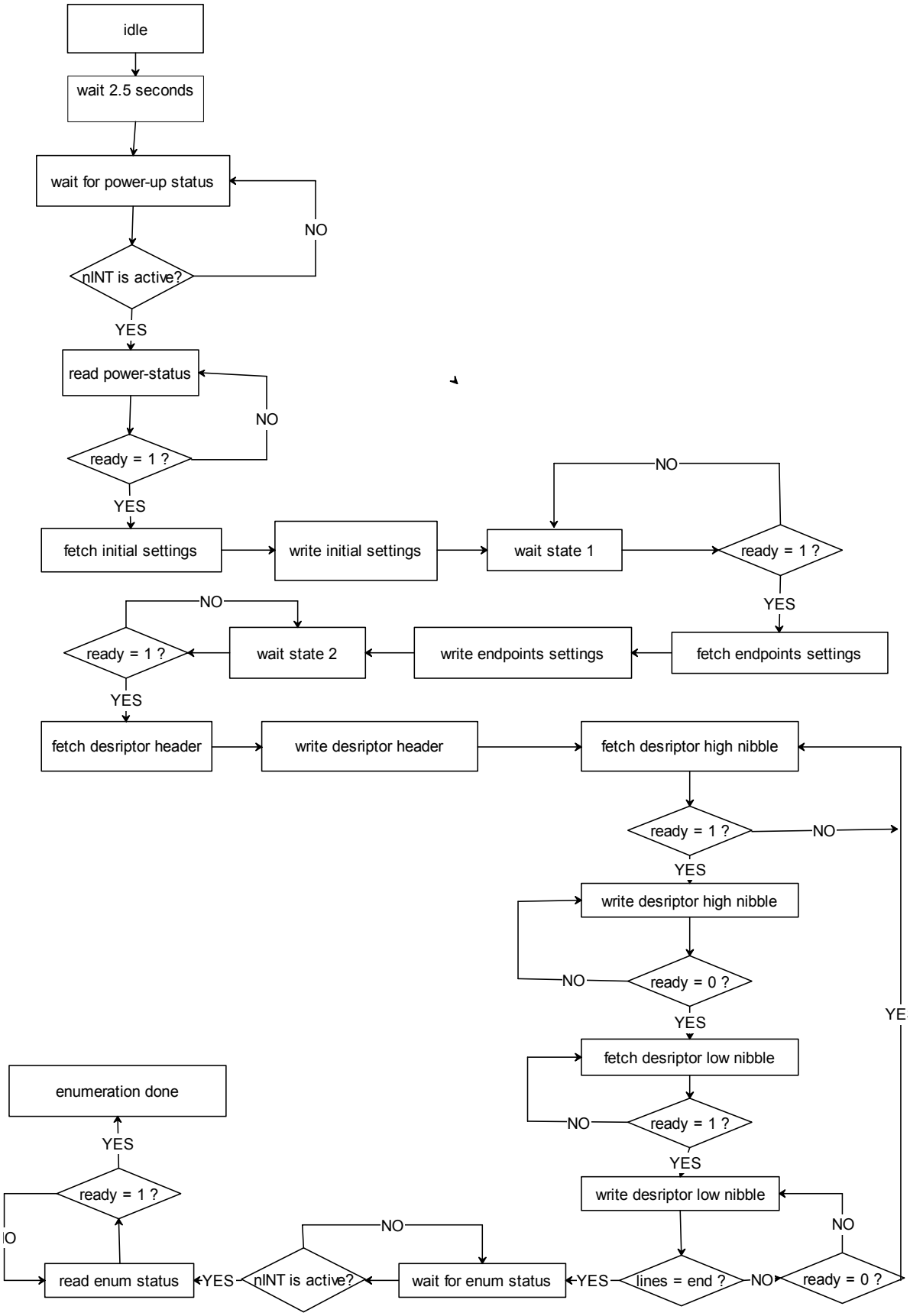
De flowchart geeft de structuur van de enumeratiecode weer. Vooraleer er control-commands kunnen worden geschreven, zal er eerst worden gecontroleerd of de USB-controller actief is (wait for power-up status). Moest dit niet zo zijn, kan de USB-interface, worden aangeroepen en uit low-power state worden gehaald met het WAKEUP-signaal.

Vervolgens zijn commando's onderverdeeld in 3 soorten:

- initiële settings: Interface Configuratie register
- endpoints settings: Initialisatie van de FIFO's en ENDPOINTS
- descriptor (high en low nibble): default descriptor die staat vermeld in de datasheet van Cy7C68001.

Er werd gekozen om eerst het commando te fetchen in de writebuffer en dan de data weg te schrijven: FETCH & WRITE.

Na het schrijven van de nodige commando's zal de toestand van de USB-controller worden gecontroleerd. Als deze stabiel is (READY = 1) mag worden aangenomen dat de USB-interface geprogrammeerd is en klaar voor gebruik.



Figuur 43: FSM enumeratie CY7C68001

Alternatieve oplossingen:

De enumeratie van de USB-controller is ook mogelijk met een voorgeprogrammeerde EEPROM op de PCB. Deze is geplaatst op de PCB maar functioneerde niet naar behoren en kon dus niet worden geprogrammeerd.

Voordeel : er moet geen code worden geschreven voor de enumeratie. Deze wordt nu via een snelle manier ingelezen via de EEPROM. De gebruiker kan onmiddellijk beginnen met data te versturen zonder eerst een lange enumeratiecode te moeten schrijven.

Nadeel: De code in de EEPROM is statisch en kan niet meer worden aangepast omdat in dit project er geen 'on board programming' is voorzien. Bovendien kan deze enumeratiecode bij een defect aan de EEPROM niet worden gebruikt, zoals in dit project dus ook het geval was.

Bijzonderheden die moesten worden onderkend:

Het was gepland om geen enumeratiecode te schrijven en deze te laden in de USB-controller vanuit een EEPROM op de PCB. Doordat deze EEPROM niet naar behoren functioneerde moest er zelf enumeratiecode geschreven worden. Vermits er al een soortgelijke code bestond voor dezelfde USB-Controller chip werd deze code verder uitgewerkt en geconfigureerd. Het initialiseren is complex en vrij essentieel omdat dit de basis is van de goede werking van een USB-controller.

Het tweede probleem was dat de geplaatste USB-interface chip van Cypress (Cy7C68001) niet naar behoren werkte, de temperatuur liep hoog op en bepaalde signalen van de chip functioneerde niet. Hiervoor is een nieuwe chip aangekocht en geplaatst op de PCB. Deze chip werkte wel naar behoren.

Het derde probleem waarop moest gelet worden zijn de connectie van de chip pinnen. Deze moeten allen verbonden zijn met de PCB en mogen niet aan elkaar vast hangen. Dit gaf verkeerde resultaten weer, zoals bv. bits in de databus die altijd 0 waren of 2 bits naast elkaar die elkaars waarde hadden. Door het analyseren van fouten in de communicatie tusen de FPGA en de USB konden de problematische pinnen worden aangeduid. Deze werden dan optisch en elektrisch (weerstands meeting) gecontroleerd en opnieuw gesoldeerd. Verbindingen die aan elkaar vast hingen werden doorgekrast of het overtollige soldeersel werd verwijderd met desoldeerdraad. Het controleren van de juiste spanningen op de pinnen werd gemeten met een meetprobe of een multimeter.

6.2.4.2 USB-OUT : van PC naar USB- interface:

Doel: Dit blok wordt gebruikt voor de toegang tot de local_bus entiteit via de GUI langs PC-zijde. Het doel is om via deze GUI een adres in te geven, wat vervolgens zal worden geraadpleegd via deze USB_OUT entity.

Er zijn nu twee mogelijkheden:

- er wordt een adres en data gezonden via de GUI: RAM actie of bewerking
- er wordt enkel een adres mee verzonden via de GUI: ROM actie

Bij de RAM actie zal data worden gestuurd vanuit de GUI zijde en deze zal worden opgeslagen op het opgegeven adres (dat ook is meegestuurd met de data). Het doel van de ROM-bewerking is om een adres aan te spreken en de data die op dit adres staat te tonen aan de gebruiker via de GUI.

Strategie:

Er zijn twee endpoint FIFO's voorzien: IN endpoint en OUT endpoint. De OUT endpoint FIFO zal data verzenden bij een RAM of ROM-actie, vanuit het standpunt van de GUI bekeken. De IN endpoint FIFO van de USB-controller zendt data naar de GUI bij een RAM-actie.

De ontvangst van de data verloopt via een OUT-FIFO en er zal worden gecontroleerd of deze FIFO data bevat via de empty flag van de respectievelijke OUT-FIFO. Wanneer de empty-flag niet actief is , zal er data aanwezig zijn in de OUT-FIFO wat een ROM of RAM actie aangeeft. Het doorsturen van de datastroom in de USB-FIFO bij het blok USB-IN (zie 6.2.4.3 USB-IN : van PC naar USB-interface) zal nu even onderbroken worden om de USB-OUT procedure volledig af te handelen. Het USB_OUTIN-signaal[15] wordt actief waardoor het USB OUT blok de ontvangen data kan verwerken (zie ook 6.2.5 MUX – Multiplexer). Met dit signaal is USB OUT blok gekoppeld met de USB databus (FD) en kan ook de FIFO adressen(FIFOADR) selecteren om het juiste endpoint te selecteren (IN of OUT). De twee eerste bytes in de USB FIFO-OUT worden gelezen en bewaard om ermee verder te werken:

	7 : MSB	6	5	4	3	2	1	0
adresbyte	ROM/RAM	open	open	adres	adres	adres	adres	adres
databyte	data	data	data	data	data	data	data	data

Tabel 4: : adres -en databyte bij USB-OUT

Vervolgens wordt het type data (ROM of RAM) bepaald aan de hand van de MSB van de adresbyte:

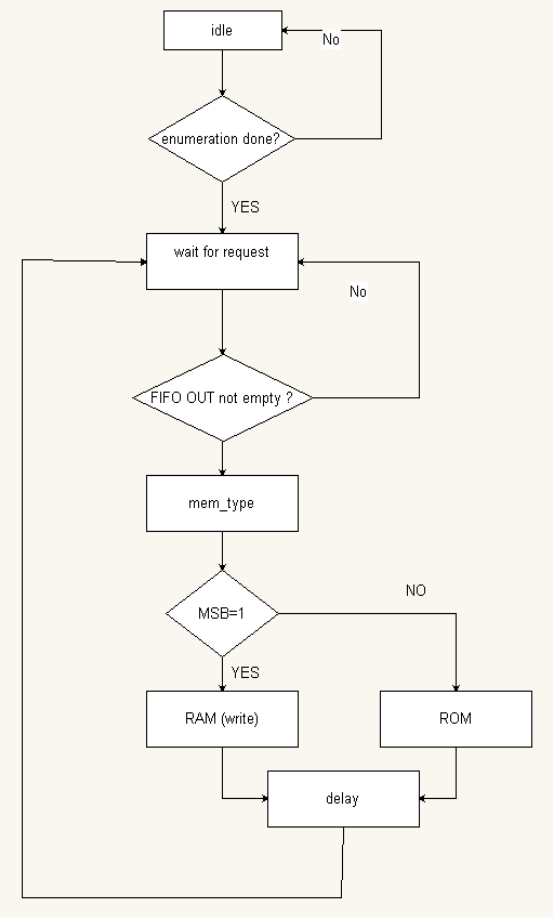
MSB	
0	ROM en RAM(read)
1	RAM (write)

Tabel 5: MSB van de adresbyte

[15] Wanneer USBOUT_IN actief is zal het USB OUT blok gekoppelde worden met de FD en FIFOADR bus, bij inactiviteit van USBOUT_IN wordt het USB IN blok met deze bussen gekoppeld.

Bij ROM mode wordt de data van het registeradres opgehaald uit de local bus en geplaatst in een aparte USB FIFO-IN (niet die van de datastroom maar een IN FIFO om kleine data te verzenden) zodat het terug naar de GUI kan worden gezonden. Met PKTEND wordt aangegeven aan de USB FIFO-IN dat de data mag worden verzonden.

Bij RAM mode[16] zal er zowel een adres als data op de local bus worden geschreven. Er wordt niets teruggestuurd naar de gebruiker of GUI. Een kleine wachttijd (=delay) op het einde zorgt ervoor dat alle lijnen terug stabiel zijn voor een volgende actie.



Figuur 44: flowchart USB OUT

<u>Bijzonderheden die moesten worden onderkend:</u>

De tijdsvereisten van de signalen voor het schrijven en lezen van data naar en van de USB-FIFO moeten worden gerespecteerd. Deze kunnen worden teruggevonden in de data sheet van de

[16] Met RAM mode wordt enkel de schrijf mode bedoeld, vermits het lezen van RAM gelijk is aan ROM.

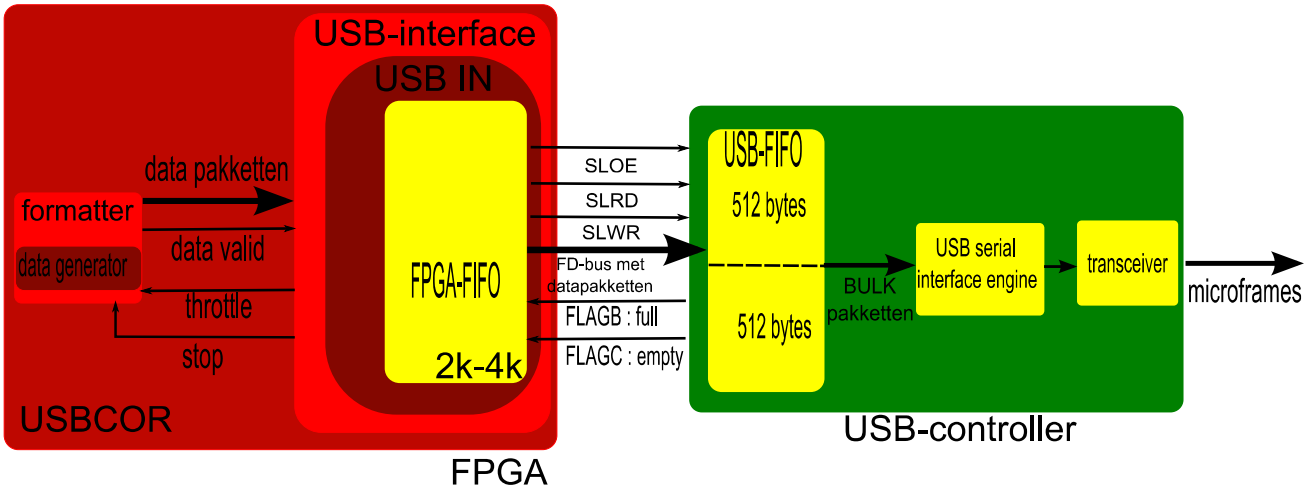
Cy7C68001 USB-controller[17].[7] Bij het lezen van data bijvoorbeeld moeten SLOE en SLRD (18.1 ns) beiden worden geselecteerd, zodat de uitgaande mode van de bidirectionele bus in de de USB-controller actief is bij het lezen van data. (zie ook 6.2.5 MUX – Multiplexer) De USB-controller moet nu uitgaande data kunnen versturen. Voor het schrijven moet SLWR minimaal 18.1 ns actief zijn, SLOE[18] moet dan inactief zijn. In het begin van de stageperiode werd met deze timings nog niet veel rekening gehouden wat voor ongewenste acties of geen actie van de USB-controller door te korte tijdsintervallen.

[17] Cy7c68001 datasheet is ook te vinden op www.cypress.com/?docID=17537
[18] SLOE = Slave Output Enable

6.2.4.3 USB-IN : van PC naar USB-interface

Doel: Data pakketten afkomstig van de formatter[19] plaatsen in de USB-FIFO (IN) en vervolgens zenden van deze data naar de host (GUI).

Strategie:



Figuur 45: USB IN met FPGA-FIFO en het opslaan de van de data in USB-FIFO in USB controller

Figuur 45 geeft het datatransport weer van FPGA naar de GUI. In het USBCOR-project heeft hiervoor 2 blokken nodig: een formatter met data generator en het USB-interface blok met het USB IN blok. Er wordt in de formatter data aangemaakt met behulp van een teller en verpakt in datapakketten. De teller zal enkel ophogen wanneer er effectief data moet worden geplaatst in de payload van het pakket. Het is ook mogelijk om voor een continue oplopende teller te kiezen. De pakketten worden doorgestuurd naar het USB IN blok in de USB-interface blok. De data (pakketten) wordt opgeslagen in de FPGA-FIFO.

Wanneer de FPGA-FIFO bijna vol is zal dit worden aangegeven met het throttle-signaal aan de formatter zodat deze enkel nog lege pakketten verstuurt. Een leeg pakket bestaat uit een header en trailer en geen dataveld. (zie 6.2.3 USB test formatter). Door het verzenden van deze lege pakketten weet de ontvanger dat er data is weggevallen en wordt de synchronisatie tussen zender en ontvanger behouden.

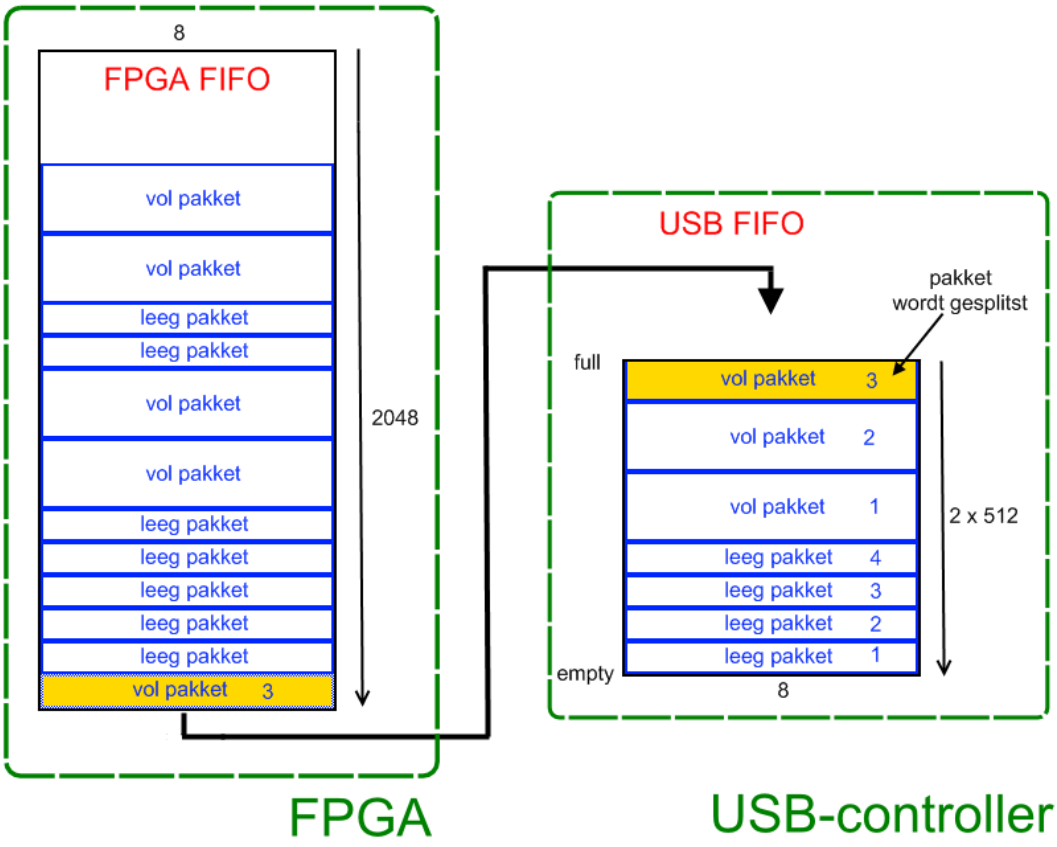
Als de FPGA-FIFO helemaal vol is wordt aangegeven met het stop-signaal zodat de formatter stopt met datapakketten te zenden. Deze overflow of error situatie wordt opgeslagen in een statusregister zodat het systeem weet dat er nu fouten zijn opgetreden, voornamelijk datadelen die wegvallen doordat de FPGA FIFO geen data meer kan opslaan.
Als er geen data in de FPGA-FIFO is wordt er geen data meer verstuurd over de USB. Pas wanneer er voldoende data in de FPGA-FIFO zit (> 1024 bytes) zal er terug data worden verzonden naar de USB-controller en via de USB naar de GUI.

De data die is opgeslagen in de FPGA wordt doorgestuurd naar de FIFO in de USB-controller, de endpoint USB-FIFO genaamd. Deze bestaat uit 2 buffers van telkens 512 bytes, wat een endpoint

[19] De formatter verpakt de data in datapakketten

FIFO grootte van 1024 geeft. Alle data die wordt verzonden (volle en/of lege datapakketten) wordt dus eerst opgeslagen in een tussenliggende FIFO in de USB-controller totdat de USB-FIFO vol is. Bij een nieuwe lege USB-FIFO wordt opnieuw data uit de FPGA-FIFO gelezen en opgeslagen in de USB-FIFO. Met het gebruik van de FPGA-FIFO zal de data afkomstig van de formatter niet rechtstreeks in de USB-FIFO worden gezet.

Het gevolg is wel dat er nu bvb. een datapakket in twee delen wordt opgesplitst. Als de USB-FIFO vol is zal het ene deel al worden verstuurd terwijl het andere deel nog in de FPGA-FIFO zit. Dit laatste 'wachtende' deel zal later worden verstuurd en de twee delen zullen terug worden samengesteld langs de PC zijde (GUI).



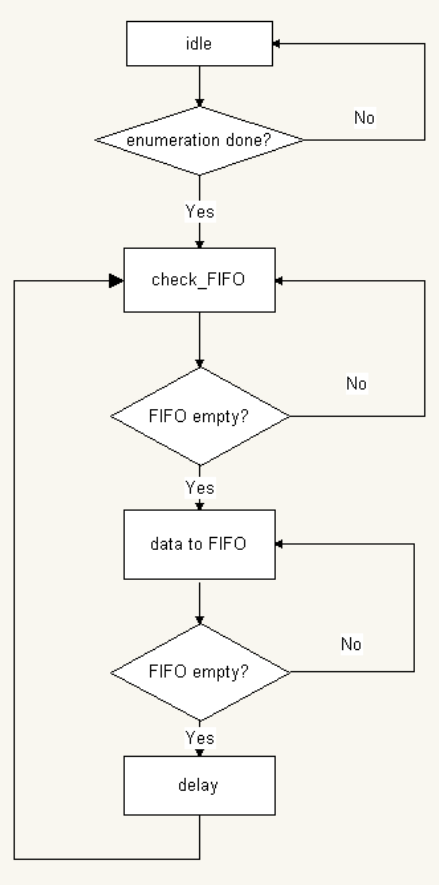
Figuur 46: pakketten van FPGA FIFO naar USB FIFO-IN in de USB-controller

De serial interface engine zorgt voor het omzetten van de bytes in USB datapakketten, in dit geval BULK datapakketten (zie 5.2 gebruikte USB interface chip : cypress cy7c68001). Deze BULK-pakketten worden in een microframe gezet en verzonden naar de GUI door de transceiver.

Finite sate machine: vullen van USB-FIFO

Met deze FSM zal de USB-FIFO gevuld worden met data. Eerst zal er worden nagegaan of de FIFO helemaal leeg is aan de hand van van de empty flag (FLAGC). Vervolgens wordt de FIFO gevuld met data totdat de volledige FIFO (alle buffers) gevuld zijn met data. Na deze actie wordt gewacht totdat de gevulde FIFO verzonden is naar de host en de inhoud gelezen is. Dit wordt gedaan door de empty flag van de FIFO te controleren. Vervolgens wordt de FIFO opnieuw gevuld met data. Dit proces

herhaalt zich voordurend. De FIFO's en hun buffers worden nu volledig gevuld, wat leidt tot een snellere dataoverdracht in tegenstelling tot het gebruik van PKTEND bij USB OUT (verlies bandbreedte).



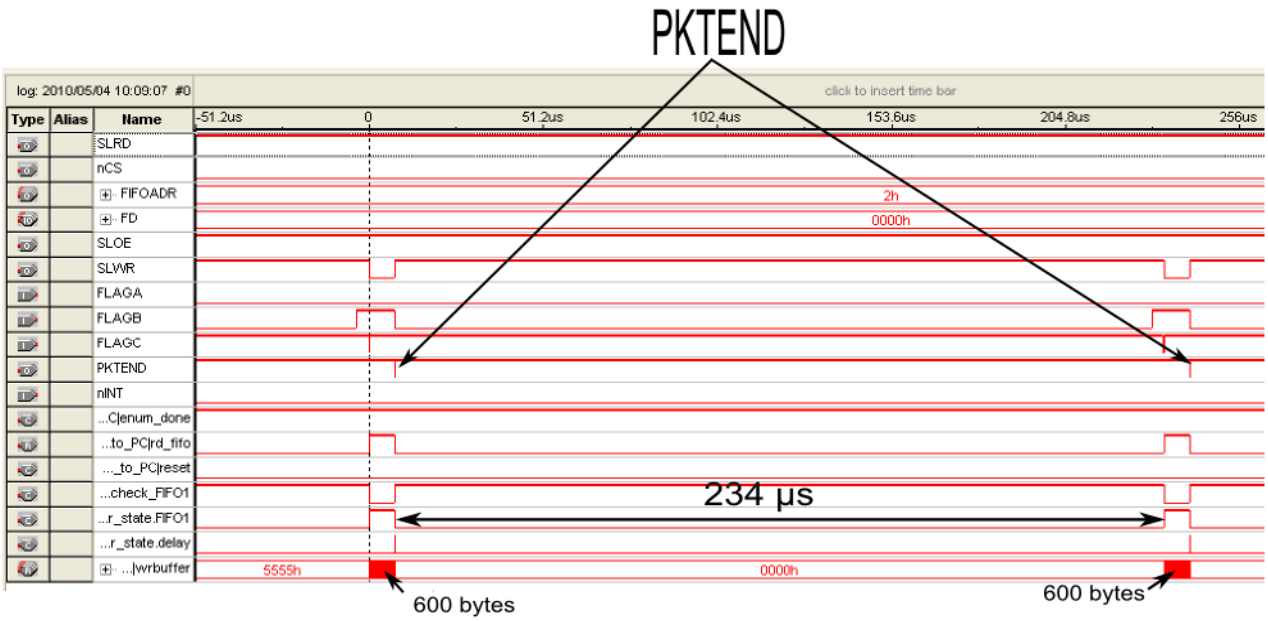
Figuur 47: flowchart USB IN

Alternatieve oplossingen:

A) PKTEND: er kan ook worden geopteerd om de USB-FIFO niet volledige te vullen met data en enkel volledige pakketten doorsturen zodat er aan de GUI zijde deze pakketten niet meer moeten worden samengesteld. Met PKTEND kan een niet volledig gevulde USB-FIFO van het endpoint worden verzonden naar de host. In Figuur 48 worden er telkens 600 bytes in de USB FIFO van het IN endpoint geschreven, gevolgd door een PKTEND ter bevestiging dat deze data met het microframe mag worden verzonden.

> Voordeel: er worden volledige pakketten doorgestuurd naar de host, de FIFO van het endpoint moet niet meer volledig worden gevuld.

Nadeel: het gebruik van PKTEND is zeer nefast voor de bandbreedte van de USB-connectie. Figuur 48 toont aan dat er tussen het verzenden van data een interval is van 234 microseconden. Dit geeft slechts een bandbreedte van 2.56 MB/s[20].



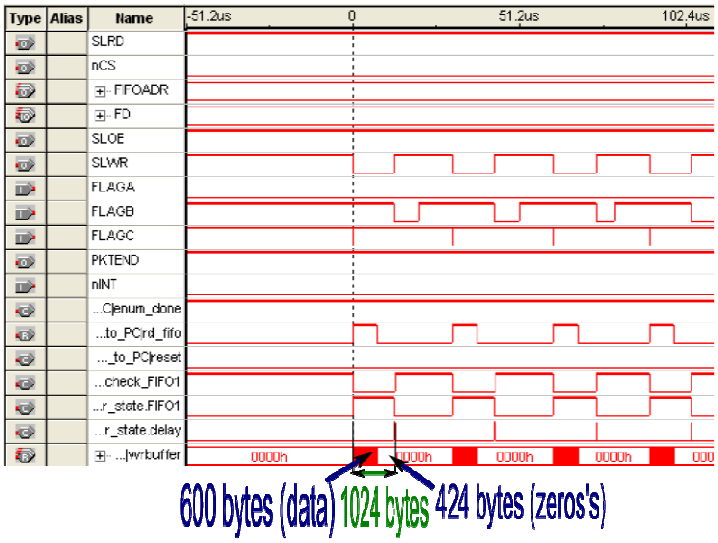
Figuur 48: verzenden van data met PKTEND methode

B) Zero-padding: Er kunnen ook volledige pakketten worden verzonden maar in plaats van. PKTEND te gebruiken kan de USB-FIFO helemaal worden opgevuld door middel van nullen toe te voegen. Hierdoor zal iedere FIFO volledig worden opgevuld.

Voordeel: De IN ENPOINT FIFO (USB FIFO IN) wordt nu telkens volledige gevuld alsook zal er meer data worden verzonden met de microframes. Dit leidt tot een normale USB-snelheid (20 MB/s, zie 6.3 USB overdrachtsnelheid).

Nadeel: Door het gebruik van zeropadding is er veel niet te gebruiken data aanwezig, de bandbreedte (van nuttige data) zakt hierdoor. Stelt dat er 600 bytes worden verstuurd en er 1024 bytes in een IN ENDPOINT-FIFO kunnen. Dan is 59 % van de data die wordt verzonden bruikbaar, de overige 41 % zijn nullen.

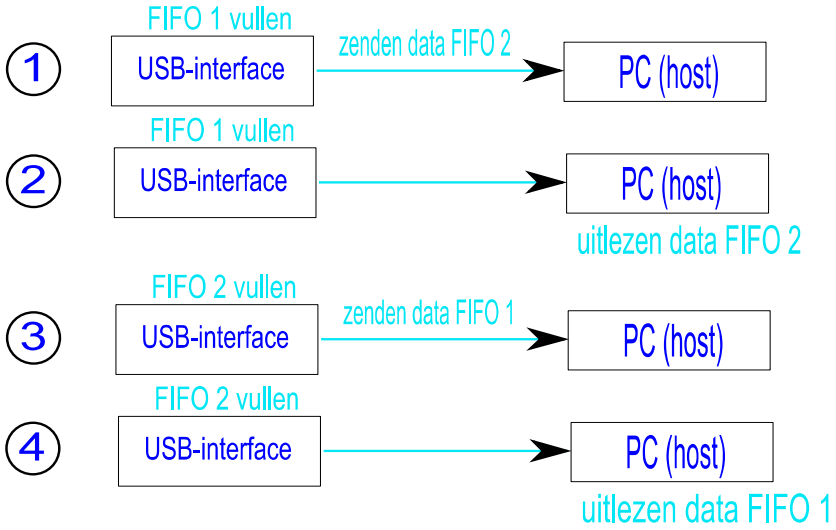
[20] 600 bytes . (1/234µs) = 2564103 bytes/s = 2.56 MB/s



Figuur 49: Verzenden van USB-data met zeropadding methode

C) Dubbele Endpoint FIFO's:

Bij het verzenden van data van USB-interface naar PC wordt in een enkele FIFO gebruikt. Het is ook mogelijk om een dubbele FIFO te gebruiken en deze afwisselend aan te sturen. Dat wil zeggen dat wanneer FIFO 1 wordt gevuld met data de data van FIFO 2 wordt verzonden naar de PC. Als de data van FIFO 2 is gelezen door de PC zal nu de data van FIFO 1 kunnen worden verzonden en FIFO 2 met data worden gevuld.



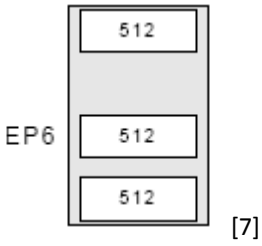
Figuur 50: werking dubbele FIFO

Het gebruik van 2 aparte FIFO's wil ook zeggen dat er 2 endpoints worden gebruikt, in dit project zijn dat enpoint 6 en 8. Hierdoor zijn er ook per endpoint aparte microframes voorzien. Het gebruikt van een dubbele FIFO is uitgeprobeerd maar het bleek al gauw dat bij het zenden van FIFO met data deze niet direct door de USB-controller van de PC wordt uitgelezen. Het duurt een tijd voordat een gevulde FIFO wordt uitgelezen en dan terug leeg is (ongeveer 2 seconden). Dit is zeer nadelig voor de USB-snelheid en er is daarom ook verkozen om met maar 1 endpoint en FIFO te werken. Tevens

wordt er per microframe slechts 1024 bytes verstuurd. Dit komt neer op een snelheid van 8.19 MB/s per endpoint of 16.37 MB/s voor beide endpoints zonder vertragingen van de USB.

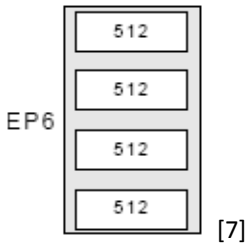
Het gebruikt van een dubbele FIFO kan wel worden toegepast mits uitgebreid onderzoek waarom het uitlezen van de beide enpoint FIFO's zo lang duurt en ook door het zenden van microframes met meer data, bvb. 3048 bytes of meer. Dit ligt aan de driver van de PC, die gestandaardiseerd is. Voor een betere ontvangst van USB-data zou de driver zelf moeten worden geschreven, al dan niet op een LINUX of WINDOWS systeem. Het zelf schrijven van een driver is niet toegepast omdat een werkende testopstelling prioritair was ten opzichte van een testsysteem met aangepaste driver om snelheidswinst te behalen. Het aanpassen van de USB-driver op een externe PC wordt dan ook beschouwd als een uitbreiding op de eigenlijke thesisopdracht.

D) Eén Endpoint FIFO een verschillend aantal buffers: triple en quad



[7]

Figuur 51: triple FIFO, FIFO endpoint met drie buffers



[7]

Figuur 52: quad FIFO, FIFO endpoint met vier buffers

Het is ook mogelijk om een FIFO te initialiseren met 4 of 3 interne buffers met een buffergrootte van 512 bytes. Voor de triple FIFO komt dit neer op een inhoudsgrootte van 1536 bytes, een quad buffer heeft een groote van 2048 bytes. Uit de testen bleek dat er nu wel meer data in een FIFO kan worden gestoken maar in het aantal bytes dat het read-blok van Labview kan lezen beperkt is tot de grootte van de FIFO.

Aantal buffers in FIFO	**FIFOgrootte (bytes)**	**Aantal bytes dat er kunnen gelezen worden met het READBLOK van Labview**	**USB transportsnelheid (MB/s)**		
			Gemeten	Signaltap	Theoretisch, 1 microframe
Double	1024	onbeperkt	+/- 20	20.23 = 8192/(405 µs)	40.96 = 10. 512 . (1/125 µs)
Triple	1536	1536	6.2	6.14 =1536/(250µs)	
quad	2048	2048	8.1	8.19 = 2048/250µs	

Tabel 6: USB transportsnelheid voor quad en tripple ENDPOINT FIFO's

Tabel 6 toont de gemeten en berekende transportsnelheid. De metingen van de transportsnelheid per FIFO-type zijn uitgevoerd met *Device Monitoring Studio 6.02*[21] Het valt op dat de gemeten

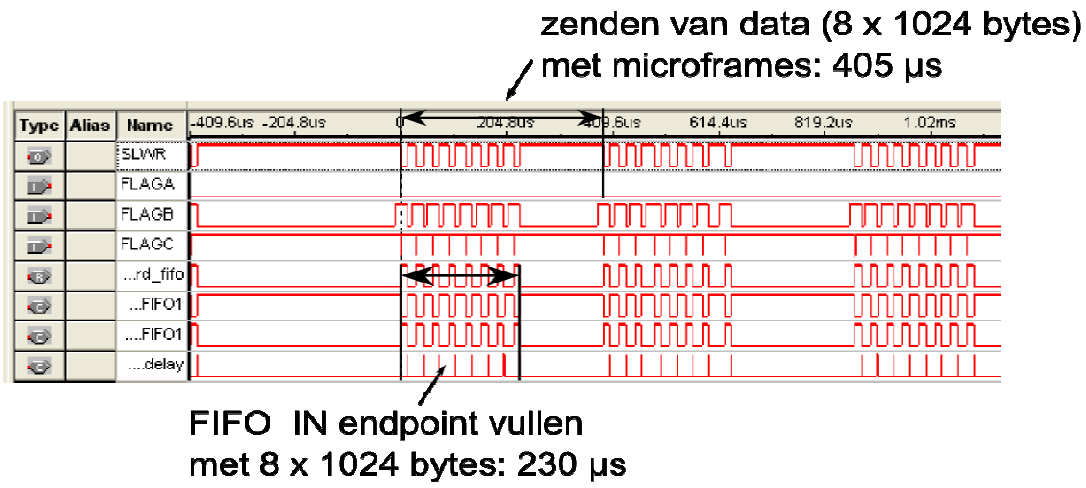
[21] http://www.hhdsoftware.com/Products/home/usb-monitor.html

snelheid van de double FIFO hoger ligt dan die van de triple en quad FIFO. De oorzaak hiervan is dat de GUI in labview enkel bij een double FIFO grotere waarden (bvb. 15360 bytes) kan uitlezen. Voor de triple en quad FIFO kan er slechts de FIFO grootte worden uitgelezen met de GUI (triple: 1536 bytes, quad: 2048 bytes). De theoretische snelheid van de USB is 40.96 MB/s. Er kunnen maximaal 10 BULK-pakketten (512 bytes) worden verzonden per microframe (zie ook 4.3.4 Beheer Bandbreedte).

Metingen Double-FIFO

Het valt meteen op dat er een verschil is tussen de gemeten (double FIFO = 20 MB/s) en de theoretische USB transportsnelheid (double FIFO = 40 MB/s), namelijk een halvering . Dit is te verklaren met door te gaan kijken naar de timings van de signalen door middel van signaltap (zie ook 6.1.1.1 Signaltap II Logic Analyzer). Figuur 53 toon het verzenden van data met een dubble FIFO (2 x 512 bytes). De FIFO ban de IN ENDPOINT worden telkens gevuld. De data in de ENDPOINT FIFO wordt door de SIE van de USB-interface chip in microframes gestoken en verzonden. Het vullen van de IN ENDPOINT FIFO duurt ongeveer 230 µs. Het zenden van de data naar de externe host duurt gemiddeld 405 µs. Dit verklaart tevens ook de gemeten snelheid: $\frac{8\,x\,1024\,bytes}{405\,\mu s} = 20.23\,MB/s$.

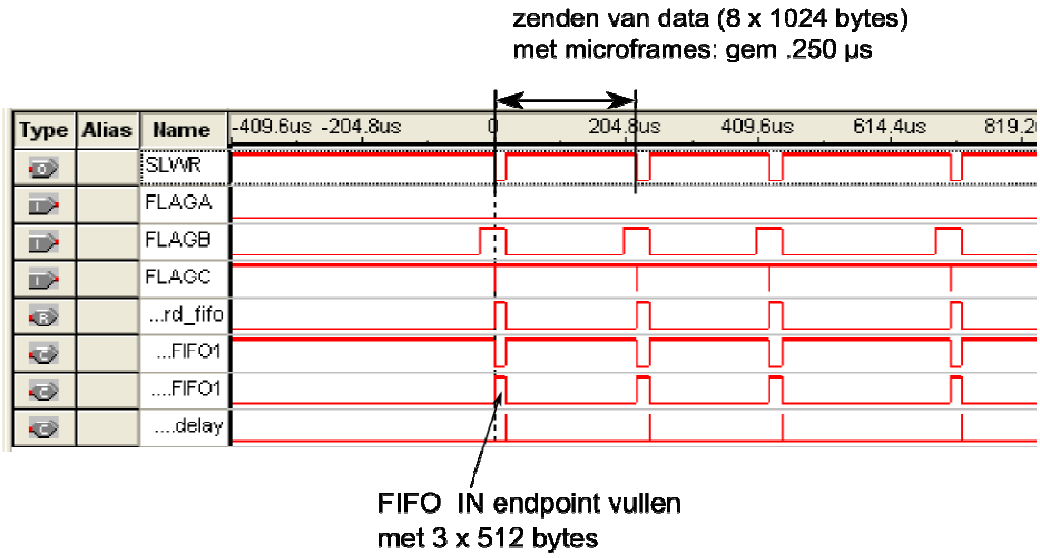
De verklaring waarom er geen continue pakketten worden doorgestuurd ligt vermoedelijk aan het uitlezen van de data langs host (PC) zijde. De geïnstalleerde software (Labview) op de PC geeft deze vertragingen.



Figuur 53: timings double FIFO

Metingen Triplle-FIFO

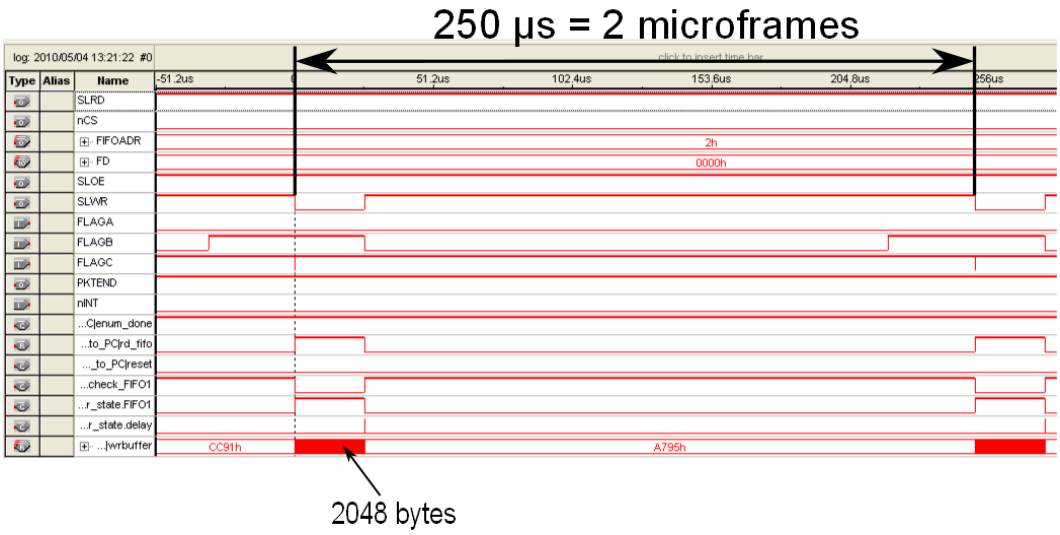
Een triple FIFO geeft een transportsnelheid van 6.2 MB/s die kan worden verklaard aan de hand van de signaaluitlezing met signaltap. In figuur 54 wordt er 1 volle FIFO van 3x512 bytes verzonden in een gemiddelde tijd van 250 µs. Dit geeft een transportsnelheid van (1536bytes/250µs =) 6.144 MB/s.



figuur 54: timings tripple FIFO

Metingen Quad-FIFO

Bij quad FIFO wordt een transportsnelheid gemeten van 8.1 MB/s. In de singaltap figuur (Figuur 55) wordt deze actie gepresenteerd. Hier worden er 4x512 bytes verzonden in 250 µs, wat een transportsnelheid geeft van 8,192 MB/s.



Figuur 55: Verzenden van USB-data met een quad-buffered ENDPOINT FIFO

<u>Bijzonderheden die moesten worden onderkend:</u>

Het grote probleem is de beperkte snelheid van de USB downlink, 20 MB/sec. De snelheidsbepaling staat beschreven in hoofdstuk 6.3: USB overdrachtsnelheid. De snelheid kan mits verdere studie nog worden geoptimaliseerd. Verhoging van de snelheid kan onder andere worden verkregen door gebruik te maken van een dubbele FIFO (zie alternatieve oplossing in 6.2.4.3 USB-IN : van PC naar USB-interface) en het schrijven van een aangepaste driver voor de externe PC. Hierdoor zal er meer en sneller data kunnen worden ontvangen langs PC-zijde. De verwerkingstijd van de externe PC is vooralsnog het grootste knelpunt in verband met het behalen van snelheidswinst op de USB.

6.2.5 MUX – Multiplexer

Doel:

Vermits er in de entity USB-interface 3 blokken zijn die allen gebruik maken van dezelfde USB-bus (= FD) en FIFOADR-bus, zal er gebruik gemaakt moeten worden van een multiplexer. Deze bepaalt aan de hand van de selectiesignalen welke van deze 3 entiteiten of blokken verbonden moet worden met FD of FIFOADR.

Strategie:
Er zijn twee zaken die volbracht moeten worden: selectie van de juiste entiteit en de omzetting van het bidirectioneel signaal in aparte signalen en omgekeerd.

Omzetting bidirectionele databus

De databus FD functioneert bidirectioneeel, dit wil zeggen dat de bus in IN of OUT mode kan staan maar niet in beide modes. Er moet met een signaal duidelijk gemaakt worden aan de FD-bus in welke mode de bus moet staan (= write to sx2). Het bidirectioneel gedrag van FD wortd geïmplementeerd door de bus aan een lees -of schrijfbuffer te verbinden. Per blok zijn er 2 buffers (lees en schrijf).

Bij het lezen moet de FD-bus in leesmode gezet worden. Uitgaande data moet worden ontvangen en worden opgeslagen in een leesbuffer. De ingaande zijde van de bidirectionele FD-bus wordt hoog impedant. Bij FD-bus in schrijfmode zal de FD-bus met de writebuffer verbonden zijn, de inkomende zijde van de FD (leesmode) wordt in tri-state gezet.

Er moet ook op gelet worden dat de directionele databus in USB-controller juist wordt aangestuurd. Dit gebeurt met het Slave Output Enable (SLOE) signaal. Bij het lezen van data zal er uitgaande data vanuit de USB-controller (output) worden verzonden naar de FPGA, het SLOE-signaal moet dus actief[22] zijn. Als er data wordt geschreven mag de bidirectionele databus niet in de uitgaande (output) mode staan, het SLOE-signaal moet nu gedeactiveerd worden.

Onderstaande VHDL-code illustreert de beschreven werkwijze voor het lezen en schrijven van data van en naar de USB controller.

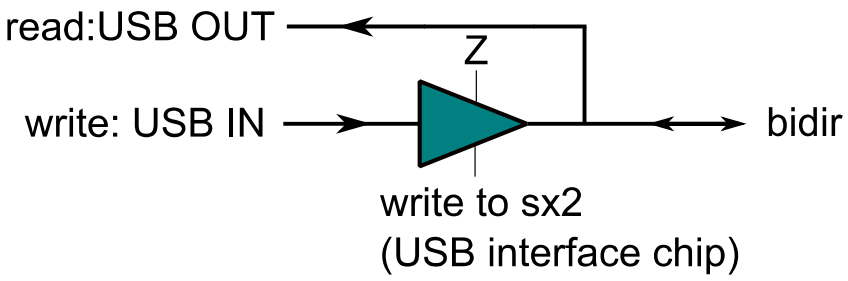
Lezen van data van de USB-controller:

```
SLOE          <=  '0';          -- output van USB-controller is geactiveerd, laag actief
write_to_sx2  <= '0';           --  er wordt geen data naar sx2 (USB controller) geschreven
rdbuffer      <= FD;            -- USB (uitgaande data) is gekoppeld aan de leesbuffer van
de FPGA
FD            <= (others => 'Z');       -- USB (ingaande data) is hoog impedant
```

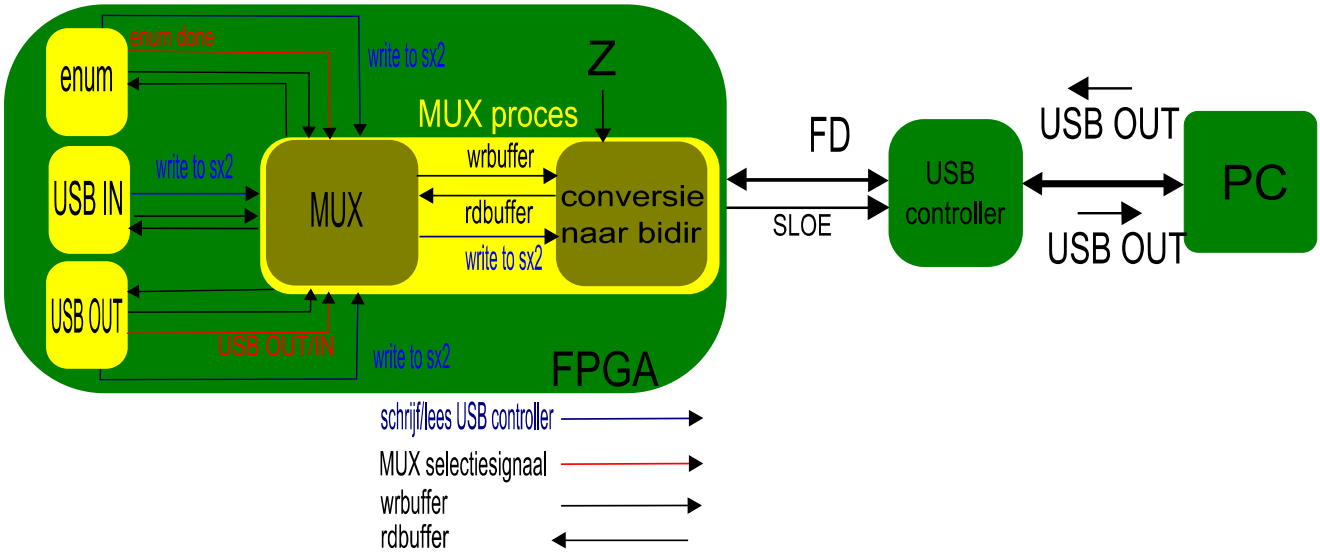
[22] SLOE is laag actief, '0' => actief, '1' => niet actief

schrijven van data naar de USB-controller

SLOE	<= ‘1’;	-- output van USB-controller is gedeactiveerd
write_to_sx2 actief	<= ‘1’;	-- er data naar sx2 (USB controller) geschreven, hoog
rdbuffer	<= (others => 'Z');	-- USB (uitgaande data) is hoog impedant
FD	<= wrbuffer;	-- data in de wrbuffer wordt gekoppeld met USB



Figuur 56: opsplitsing van één bidirectionele bus in lees –en schrijfbus



Figuur 57: positie van MUX-proces in de FPGA

Selectie van de juiste entiteit met een MUX:

Als tweede actie zal er een selectie moeten plaatsvinden tussen de 3 entiteiten. Dit hangt af van welke actie er moet worden uitgevoerd met met USB-controller: initialisatie van de UBS (enumeratie), leesactie van de USB (USB OUT) of een schrijfactie van de USB (USB IN).

Er zijn hiervoor twee selectiesignalen: enum_done (enumeration done) en USB_OUTIN. Enum done zal inactief (= 0) zijn als de USB-controller moet worden geïnitialiseerd. Dit signaal zal actief zijn als de initialisatie van de USB-controller is uitgevoerd. Na de enumeratie zijn er nog twee acties die kunnen worden uitgevoerd: het lezen en schrijven van USB-data. De keuze tussen deze twee acties worden

gemaakt door middel van het USB_OUTIN-signaal. Wanneer USB_OUTIN actief is zal de entieit USB_OUT worden geactiveerd en wordt er data gelezen van de USB. Bij USB_OUTIN inactief wordt er data verzonden of schreven naar de USB met de geactiveerd USB IN entieit.

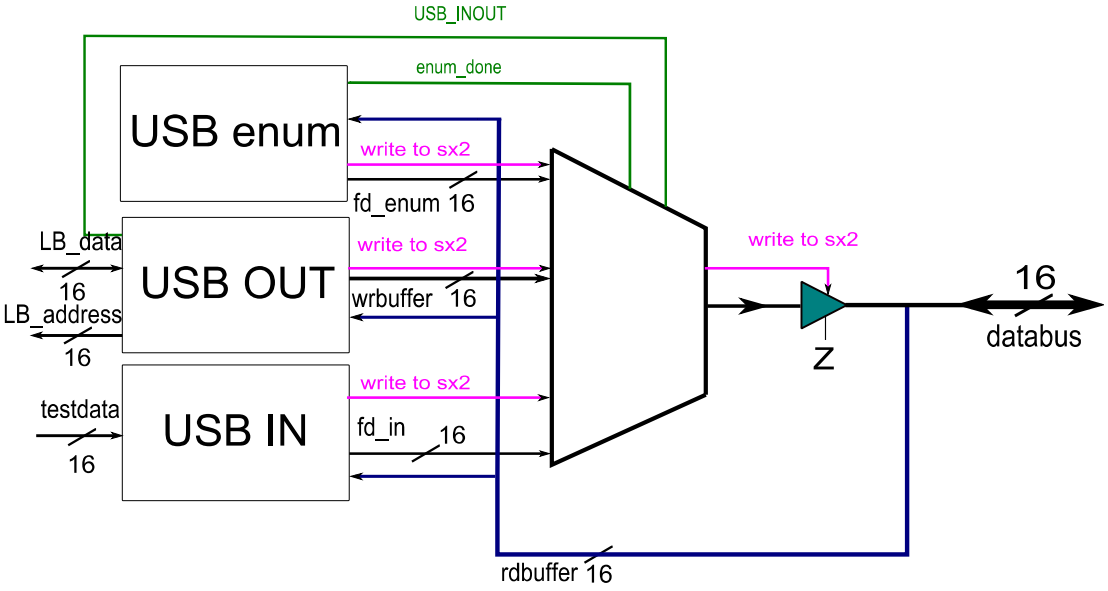
Een overzicht van de selectiesignalen en de toestand van de FD-bus en FIFOADR-bus

enum_done	USB_INOUT	write_to_sx2	FIFOADR			FD		
			eniteit			entiteit		
			enum (a)	USB OUT (	USB IN (c)	enum (a)	USB OUT (	USB IN (c)
0	0	0	OK	Z	Z	read	Z	Z
0	0	1				write	Z	Z
0	1	0	OK	Z	Z	read	Z	Z
0	1	1				write	Z	Z
1	0	0	Z	Z	OK	Z	Z	read
1	0	1				Z	Z	write
1	1	0	Z	OK	Z	Z	read	Z
1	1	1				Z	write	Z

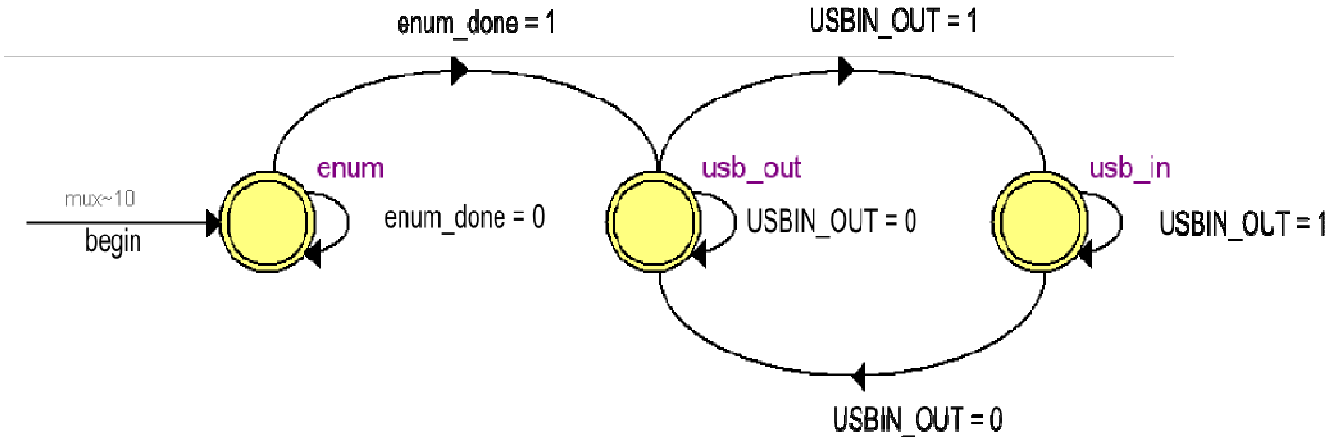
Tabel 7: waatrheidstabel MUX

<u>Toelichtend voorbeeld, aangeduid met groen in de tabel:</u>

De USB-controller heeft data ontvangen van de GUI via USB OUT. Het systeem merkt dit op en zet het signaal USB_OUTIN actief. De tabel toont aan dat het FIFOADR nu gekoppeld is aan USB OUT en de FD ook geactiveerd is. Naargelang de USB OUT entiteit wil schrijven of lezen zal de FD databus zich in deze toestand zetten met behulp van het selectiesignaal ‘write to sx2’. In dit geval wordt er data geschreven, de FD bus staat in schrijftoestand. Uit de tabel is het duidelijk dat wanneer er één van de 3 blokken niet actief is de overeenkomstige buffers (fifoadres en fd) hoog impedant is. (=Z).



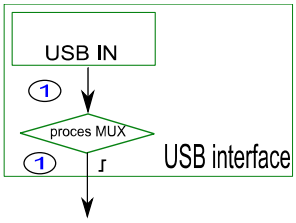
Figuur 58: grafische weergave MUX



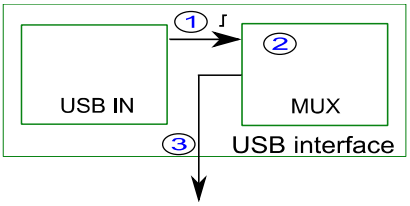
Figuur 59: MUX finit state machine

Alternatieve oplossing:

Er werd besloten om de MUX in een proces te beschrijven in het blok USB Interface. Hierdoor zal er een snelle overdracht zijn tussen het USB IN-blok en de buitenwereld. Er is slechts 1 klokperiode nodig om de data ter transporteren buiten het USB-interfaceblok. Het is ook mogelijk om het MUX-process in een VHDL-blok te zetten,hierdoor zijn er in totaal 3 klokperioden nodig voor datatransport naar de buitenwereld. (DATA IN blok → MUX blok; MUX blok → buiten USB inteface).



Figuur 60: MUX geïmplementeerd als proces

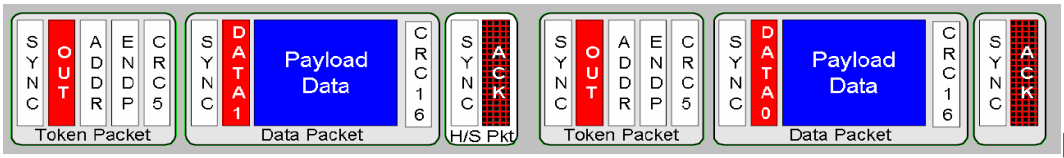


Figuur 61: MUX geïmplementeerd als VHDL-blok

Voordeel MUX-blok: duidelijk overzicht waar de MUX zicht bevindt en eenvoudiger om het MUX-blok te koppelen.
Nadeel MUX-blok: door een extra blok in de USB-interface te plaatsen, moet de data eerst nog langs het MUX-blok om vervolgens terug naar de USB-interface te gaan.

6.3 USB overdrachtsnelheid

USB 2.0 heeft een maximumsnelheid van 480 bps (= 60 MB/sec), hiervan mag niet van worden uitgegaan bij het ontwikkelen van USB-software. Er zijn verschillende redenen waarom het USB-toestel nooit de volledige bandbreedte zal kunnen gebruiken. De eerste oorzaak is dat de USB-bus vaak gedeeld wordt door verschillende toestellen. De kans is groot dat verschillende toestellen gebruik maken van dezelfde hotscontroller, zo moet het toestel de toegestane bandbreedte delen met de andere toestellen.



[13]

Figuur 62: BULK Transfer microframe

De tweede oorzaak is het BULK-protocol die pakketgroottes van 512 bytes toelaat. Ieder pakket bestaat op zich nog uit een header en een CRC aan het einde van het pakket voor foutcontrole. Ieder pakket vereist nog een ACK afkomstig van de andere zijde van de USB-link ter bevestiging dat het pakket is ontvangen. SOF-pakketen worden iedere 125 µs verzonden om synchroon te blijven met de USB. Het effect van deze zaken is dat de theoretisch bandbreedte van USB 13 pakketten per microframe zijn of 53248000 bytes/sec. Deze waarde is nog niet haalbaar doordat de meeste hotscontrollers 10 bulk pakketten/microframe kan ontvangen en 8 bulkpakketten/microseconden kan zenden. Tabel 8 toont een maximale ontvangstsnelheid voor de host-controller (PC) van 40.96 MB/sec.

		Bandbreedte (MB/sec)
zenden	8 . 512 bytes / 125µs	32,768
ontvangen	10 . 512 bytes / 125µs	40.96

Tabel 8: Bandbreedte USB

QIE kaarten

Voor de volledige opstellingen worden er 6 QIE-kaarten gebruikt. Per fiber worden er 3 kanalen (= 3PMT's) van de CASTOR uitgelezen en voor 1 QIE kaart (2 fiberuitgangen) geeft dit 6 uitgangskanalen. In totaal wordt er 64 bit (= 8 byte) getransporteerd per QIE kaart, zie tabel.

fiber 1																															
	PMT1										PMT2										PMT3										
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
dv	ER	CapId		Exp		mant sect0 EM0					ER	CapId		Exp		mant sect0 EM1					ER	CapId		Exp		mant sect0 H0					1
fiber 2																															
	PMT4										PMT5										PMT6										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
dv	ER	CapId		Exp		mant sect0 H1					ER	CapId		Exp		mant sect0 H2					ER	CapId		Exp		mant sect0 H3					1

Tabel 9: : uitgangsdata van de QIE-kaart

In tabel zien ze de data die wordt verzonden vanuit de QIE-kaart. Er zijn 2 fibers weergegeven die elk 32 bit verzenden. Er kunnen in totaal 10 samples worden verzonden (6 pre, 1 huidige, 3 post) in een pakket.

n bytes	veldtype	QIE kaart
0	header	
8		
16		
24		
32		
40	pre sample 6	QIE1
48	pre sample 6	QIE2
56	pre sample 6	QIE3
64	pre sample 6	QIE4
72	pre sample 6	QIE5
80	pre sample 6	QIE6
88	pre sample 5	QIE1
96	pre sample 5	QIE2
104	pre sample 5	QIE3
…	…	…
136	pre sample 4	QIE1 - QIE6
184	pre sample 3	QIE1 - QIE6
232	pre sample 2	QIE1 - QIE6
280	pre sample 1	QIE1 - QIE6
328	pre sample 0	QIE1 - QIE6
376	sample	QIE1 - QIE6
424	post sample 1	QIE1 - QIE6
472	post sample 2	QIE1 - QIE6
520	post sample 3	QIE1 - QIE6
568	trailer	
576		
584		

Figuur 63: structuur van samples in een QIE datapakket

n QIE	header	trailer	data (bytes)	data x samples	totaal	bytes/sec
1	40	24	8	48	112	11200
2	40	24	16	96	160	16000
3	40	24	24	144	208	20800
4	40	24	32	192	256	25600
5	40	24	40	240	304	30400
6	40	24	48	288	352	35200

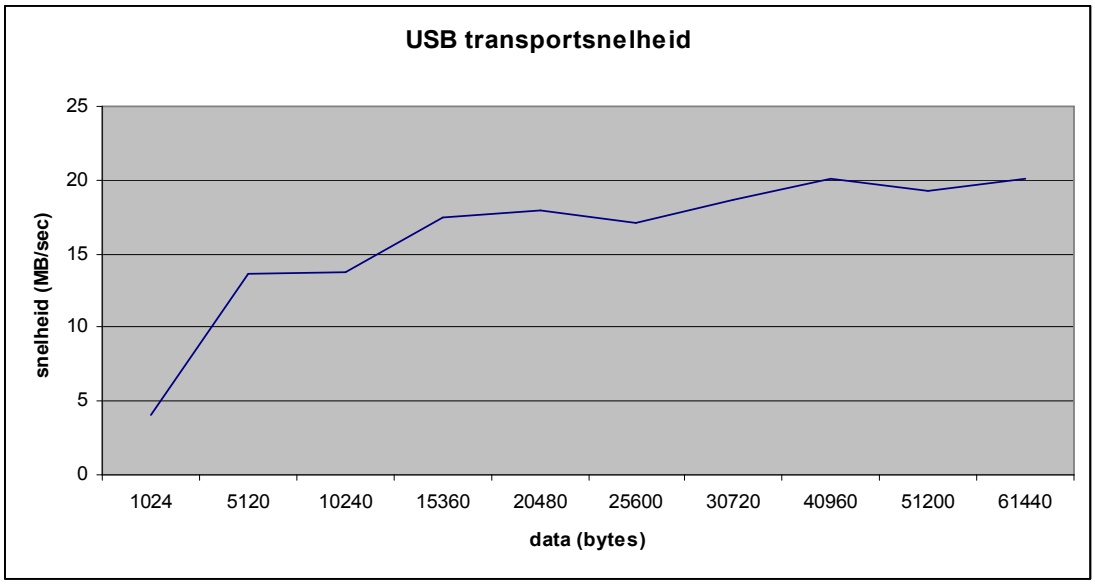
samples = 6
trigger (kHz) 100
1 QIE-kaart = 64 bit of 8 byte

Tabel 10: snelheden per QIE kaart

In het project is er gekozen voor zes samples (3 pre, 1 sample, 2 post) van de tien en worden er 6 QIE-kaarten gebruikt. De uitgangsfibers van de QIE hebben zo een transportsnelheid van 35.2 MB/sec.

Metingen

Via de GUI kan de gevraagde hoeveelheid data worden ingesteld in bytes. Het VISA-read blok buffert dit aantal vooraleer de data wordt verwerkt (opslaan). Naargelang de hoeveelheid data die wordt opgevraagd zal de snelheid van de USB ook groter worden. Dit wordt weergegeven in onderstaande grafiek. Als er in Labview 40960 of 61440 bytes worden gebufferd zal de USB overdrachtsnelheid 20 MB/sec bedragen. Dit is 57.1 %[23] van de uitgangssnelheid van de 6 QIE-kaarten. Niet alle data afkomstig van de QIE zal dus kunnen worden verzonden over de USB. (zie USB IN).



Figuur 64: USB transportsnelheid[24]

Opmerkelijk is dat de 20 MB/sec die er wordt verkregen slecht 62.5 %[25] van de praktisch haalbare bandbreedte bij USB 2.0 bij het zenden door de hostcontroller (=32,768 MB/sec) bevat. Wat gebeurt er met de overige 37.5 % aan bandbreedte? Zoals al eerder vermeld kan dit zijn door het gebruik van meerdere apparatuur op één hotscontroller of door overhead van de ontvanger. Bij de testopstelling is ervoor gezorgd dat er geen extra apparaten bij de host-controller van de USB-interface staken. De muis van de PC bijvoorbeeld is via een andere host-controller aangesloten.

Onderstaande figuur toont het verzenden van 2 microframes vanuit de USB-controller naar de PC. Per microframe wordt er 4096 byte verzonden. De USB-FIFO wordt dus per microframe vier maal gevuld (4 x 1024 bytes). De serial interface engine (SIE) van de USB-controller zal deze FIFOdata verpakken in BULK-pakketten en in het microframe plaatsen. De twee microframes hebben een tijdspannen van 250µs en bevatten in totaal 8192 bytes payload. Door de overhead van het besturingssysteem, labview en de CPU zal er een tijd nodig zijn vooraleer alle data kan worden

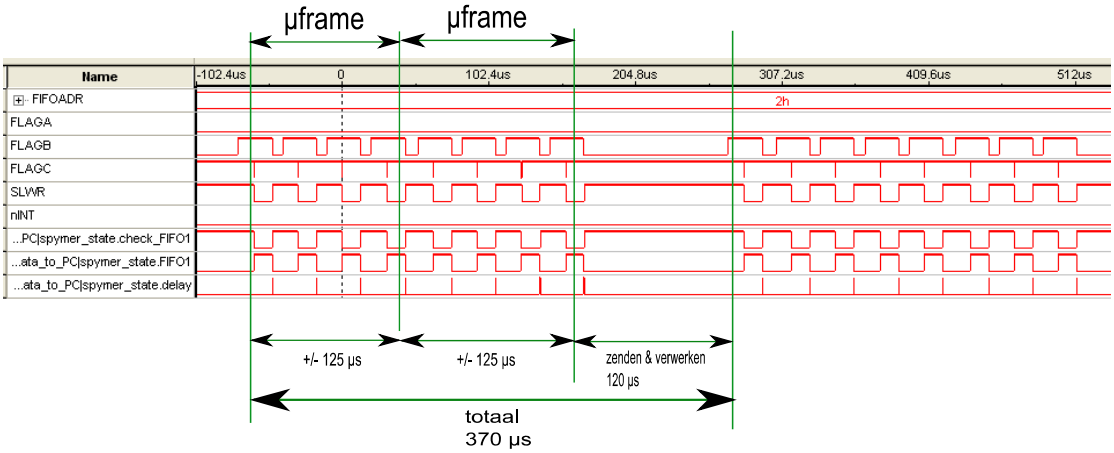
---

[23] $\frac{20\ MB/s}{35.2\ MB/s} = 0.571\,x\,100 = 57.1$

[24] De USB snelheid is gemeten met het programma Device Monitoring Studio 6.02 ®

[25] $\frac{20\ MB/s}{32\ MB/s} = 0.625\,x\,100 = 62.5$

uitgelezen van de microframes en worden verwerkt, dit is 120 µs. In totaal wordt er dus 8192 bytes verzonden in 370 µs, wat een bandbreedte geeft van 21 MB/sec[26] die nagenoeg gelijk is aan de gemeten bandbreedte (20 MB/s) in Figuur 64: USB transportsnelheid.



Figuur 65 : datatransfer tussen USB-controller en PC

Oplossing

Vermits er maximaal 40 MB/sec kan worden overgedragen en er slechts 20 MB in praktijk wordt gehaald is er nog ruimte om dit te optimaliseren. De data in de Cypress USB-Controller wordt met een kloksnelheid van 40 MHz behandelt (SIE) en verzonden. Deze kloksnelheid kan nog worden opgehoogd naar zijn maximale waarde van 50 MHz. Dit zou een snellere kloksnelheid geven (+ 20%)[27]. Er zal nu sneller data kunnen worden verzonden van de USB-Controller naar de externe PC. Een andere oplossing is het aanpassen van de PC-driver zodat het mogelijk is om met dubbele uitgaande IN FIFO's te werken. Labview heeft problemen om twee ENDPOINT FIFO's om de beurt uit te lezen. Een andere GUI programma zoeken, bvb. een GUI in C-code schrijven kan een oplossing zijn. Hierdoor moet minder rekening worden gehouden met de statische grafische blokken die moeilijk te configureren zijn in Labview voor specifieke vereisten.

Bijzonderheden die moesten worden onderkend:

Wanneer er in de GUI van Labview data naar een file wordt weggeschreven zal dit invloed hebben op de USB snelheid. Dit komt doordat er data wordt weggeschreven en dit verwerkingstijd van de CPU vraagt. Dit is wel opgelost door eerst de data in een array te plaatsen en bij het beëindigen van het schrijven naar data zal deze array naar een file worden weggeschreven.

[26] $\frac{8192bytes}{370\mu s} = 21\frac{MB}{s}$

[27] $1 - \frac{40}{50} = 1 - (0.8 * 100) = 20\,\%$

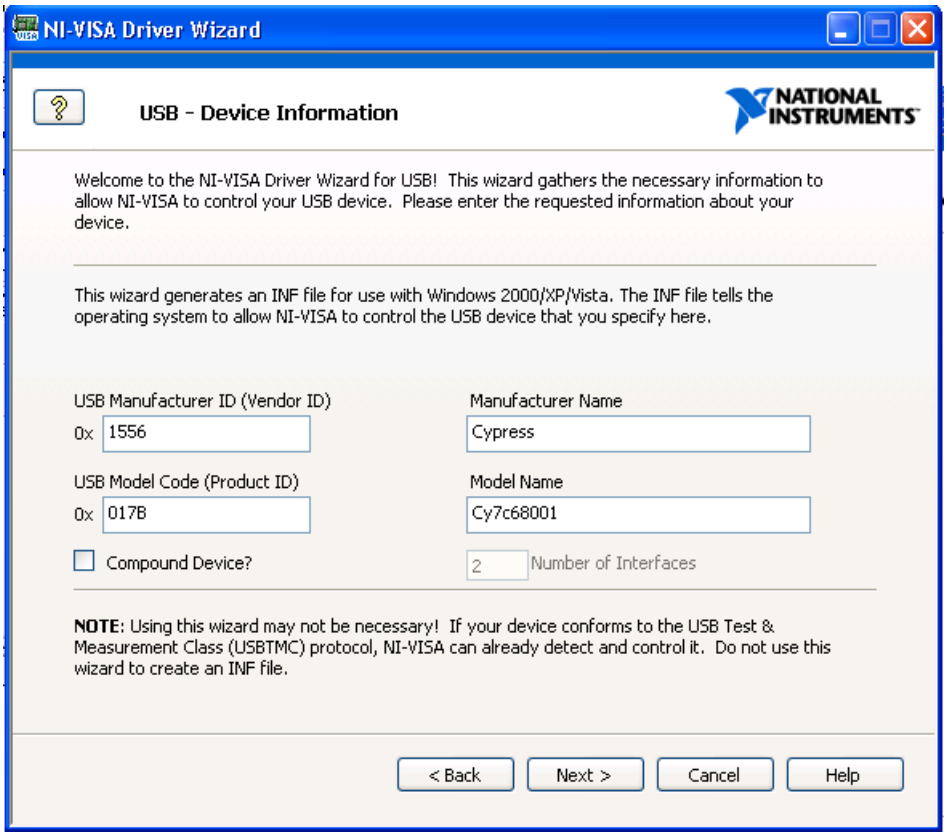
7 Implementatie van de GUI in Labview

7.1 Driver voor USB-controller IC Cy7c68001

Tijdens de stageperiode zijn er twee drivers van producenten gebruikt: Cypress en National Instruments. Deze zijn beide eenvoudig te configureren door het instellen van het PID (product ID) en de VID(vendor ID). Beide driver kunnen worden teruggevonden in de bijlagen.

VISA driver – bijlage B

Deze driver wordt geconfigureerd door een Driver Wizard door het invullen van de VID, PID, fabrieksnaam en het model.



Figuur 66: NI-VISA Driver Wizard

Cypress driver – bijlage C

De driver wordt ingesteld door de VID, PID en benamingen in te geven in de setup information file (.INF)

```
[Device]
;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX
%VID_1556&PID_017B.DeviceDesc%=CyUsb, USB\VID_1556&PID_017B

[Strings]
CYUSB_Provider     = "Cypress"
CYUSB_Company      = "Cypress Semiconductor Corporation"
```

```
CYUSB_Description = "Cypress Generic USB Driver"
CYUSB_DisplayName = "Cypress USB Generic"
CYUSB_Install     = "Cypress CYUSB Driver Installation Disk"
VID_1556&PID_017B.DeviceDesc="Cypress USB Generic Driver (3.4.1.20)"
CYUSB.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
CYUSB_Unused      = "."
```

Figuur 67: aanpassingen in de tekstdriver van Cypress Cy7c68001

7.2 Ontwerp GUI

Doel: met behulp van de GUI zal de gebruiker data kunnen ontvangen, bekijken en opslaan. Verder zal de host ook data kunnen zenden naar de USB-conctroller.

LABVIEW:

Voor de GUI langs de host-zijde werd gebruik gemaakt van labVIEW (Laboratory Virtual Instrument Engineering Workbench).

LabVIEW is een volledig grafische programmeeromgeving die ontwikkeld is door National Instruments (NI). Volledig grafisch wil zeggen dat, niet alleen de gebruikersinterface grafisch is opgebouwd, maar dat ook de eigenlijke programmacode grafisch opgebouwd wordt. LabVIEW is dus geen programmeertaal, maar eerder een programmeeromgeving.

Het toepassingsdomein van LabVIEW is vrijwel onbegrensd. Bij voorkeur echter, komen toepassingen in aanmerking waarbij gegevens dienen verzameld, geanalyseerd en gepresenteerd te worden. LabVIEW wordt daarom vooral toegepast voor technische, wetenschappelijke en industriële toepassingen.[1]

Door zijn grafische omgeving leek labVIEW het beste programma om de GUI in op te bouwen. Verder was het al gekend van in de opleiding PB-EI(MCT) te KdG.

VISA: Virtual Instrument Software Architecture

Het is in de praktijk niet altijd even eenvoudig om de software te ontwikkelen voor al de specifieke hardware die moeten worden aangestuurd. Vandaar dat er steeds getracht is om de basiscommunicatiesoftware te standaardiseren zodat de gebruiker zich enkel hoeft te concentreren op de toepassing die hij zelf maakt en niet op de hardware die hij gebruikt.
VISA biedt hiervoor de oplossing. VISA is een standaard I/O API (APlication Interface) voor het programmeren van meet –en stuurinstrumenten. VISA ondersteunt communicatiehardware zoals GPIB, seriële, USB, Ethernet, PXI- en VXI-instrumenten. Met VISA moeten de specifieke protocols niet meer worden gekend en geïnstalleerd. Deze basiscommunicatie zit reeds vervat in de VISA-functies.

Strategie:

Door gebruik te maken van een finite state machine in labview zullen de volgende processen worden doorlopen:

- **Idle**: rusttoestand waarin de GUI wacht op een actie van de gebruiker
- **Open**: de gebruiker klikt op start en zal hiermee een USB-connectie met de USB-interface opzetten m.b.v. 'VISA OPEN'. De resource geeft het gewenste USB toestel aan.

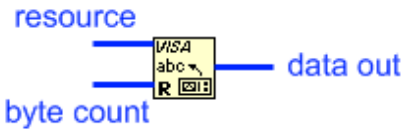
Figuur 68: VISA OPEN

- **Start**: de GUI zendt nu een start signaal naar de USB-interface. Er zal een localregister adres worden verzonden met data zodat de FPGA weet dat de GUI klaar is om data te ontvangen.

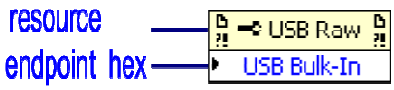
	local register adres	data
start	USB_startstop = 0x9B	0000 0001
stop	USB_startstop = 0x9B	0000 0000

Tabel 11: start -en stopcode voor USB-data trransport

- **DATA IN**: In deze state wordt de data afkomstig van de USB-interface gelezen d.m.v. het VISA-READ blok. Het aantal uit te lezen bytes wordt ingesteld met de 'byte count' waarde. Het gewenste endpoint van het USB toestel wordt geselecteerd (hexadecimaal) met de poperty node in USB Bulk-In mode gezet.



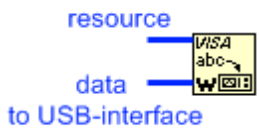
Figuur 69: VISA READ



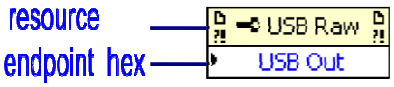
Figuur 70: property node: USB Bulk-In

In werkelijkheid kunnen er maar maximaal 3072 bytes per leesactie worden uitgelezen (= grootte van 1 microframe). Als er nu via de 'byte count' van het VISA read blok meer data is gevraagd dan zal Labview deze 3072 bytes eerst bewaren in een buffer tot het gewenste aantal is bereikt
Stel dat de bytecount is ingesteld op 15360 byte. Er 5 maal 3072 bytes worden opgeslagen in de labview-buffer, en bij een waarde van 15360 bytes wordt de volledige inhoud (15360 bytes) naar buiten gebracht via 'data out' van het VISA Read blok.

- **DATA OUT**: Deze toestand wordt gebruikt om toegang te krijgen tot de Local BUS registers vanuit de GUI. Er kunnen acties plaatsvinden: RAM en ROM. Deze state zal dus data schrijven naar de USB-interface met behulp van het VISA-WRITE blok. Het property node blok (mode USB Out) selecteerd het gewenste endpoint OUT FIFO van het USB toestel (resource).



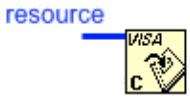
Figuur 71: VISA write



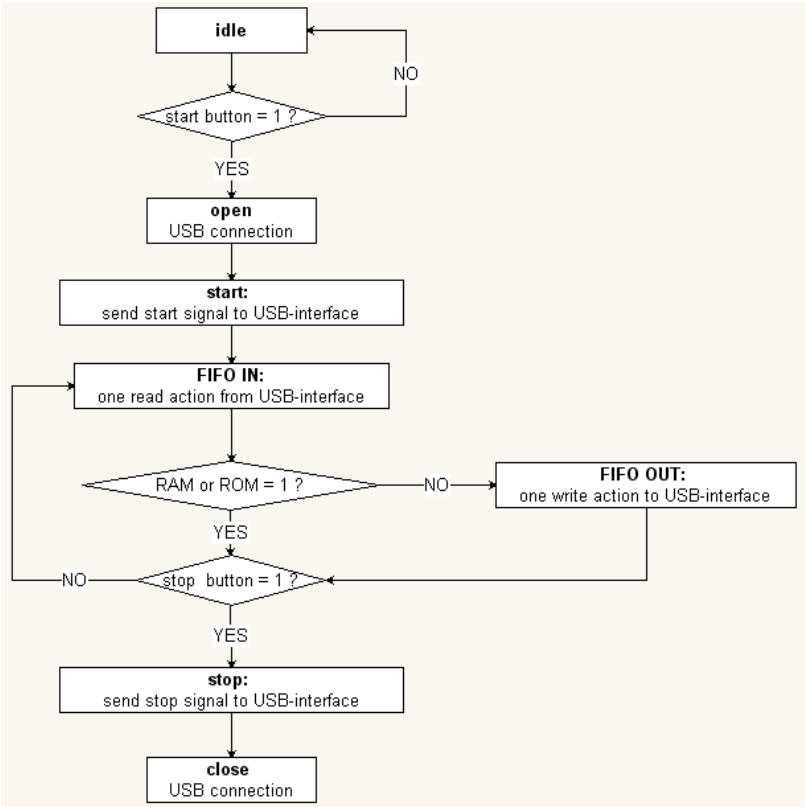
Figuur 72: property node: USB out

- **Stop:** de gebruiker activeert de stop button. De GUI zendt stop data naar de USB-interface entiteit in de FPGA, bijvoorbeeld 9B 00[28] (zie Tabel 11). Wanneer de USB OUT entiteit deze code ontvangt weet deze dat ze moet stoppen met data te zenden.

- **Close: D**e USB-connectie tussen host en USB-interface wordt gesloten met het VISA CLOSE blok.



Figuur 73: VISA CLOSE



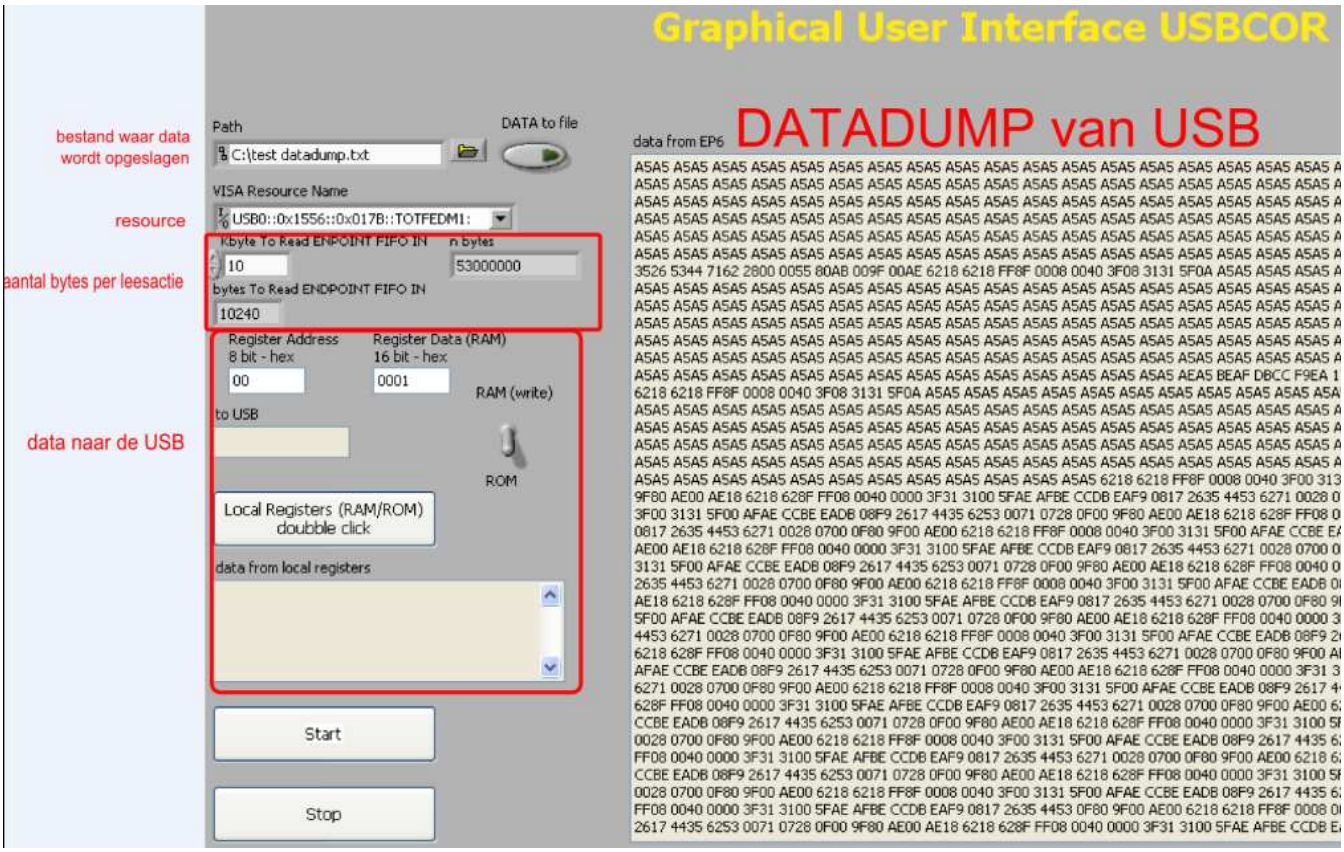
Figuur 74: flowchart GUI – labview

Grafisch

Een eenvoudig grafisch venster geeft de nodige knoppen en labels weer voor de gebruiker:

[28] 9B is het startstop register van de lokale bus, 00 is de data en geeft aan dat het zenden van data gestopt is

- Start: starten van de USB-connectie
- Stop: stoppen van de USB-connectie
- Resource name: selectie van het juiste USB-toestel bestaande uit VID[29] en PID[30]
- Adres en data label: toegang meegeven van het local bus adres en de gewenste data
- Local register (RAM/ROM): toegang tot de local registers van de FPGA
- Data from USB CONTROLLER: data dump afkomstig van de USB-interface



Figuur 75: Grafisch Venster in labview

Alternatieve oplossingen:

Er werd gebruik gemaakt van labview om zo een snel resultaat te verkrijgen. Labview biedt kant en klare VISA blokken om acties uit te voeren met een USB-toestel. Hoewel dit vrij eenvoudig lijkt, heeft labview ook enkele nadelen. De VISA blokken zijn redelijk statisch en de gebruiker is strikt gebonden aan deze blokken. Het systeem wordt getest met een Windows XP besturingssysteem. Een oplossing kan zijn om als besturingssysteem een LINUX-versie te gebruiken en dan ook een andere GUI te gebruiken. Er zijn ook commando's in C-programmeertaal waarmee er een USB uitleesmechanisme kan worden gemaakt. In onderstaand tabel worden er enkele voorbeelden van C-code gegeven waarmee er aan USB communicatie kan worden gedaan.

[29] VID = Vendor ID

[30] PID = Product ID

Configuratie van de USB-interface `int usb_set_configuration(struct usb_device *dev, int configuration);`	

CONTROL transfer (device, endpoint)
 	
`pipe=usb_sndctrlpipe(dev,endpoint)`
`pipe=usb_rcvctrlpipe(dev,endpoint)`
	
	Zenden van controle commando's naar de USB
	Ontvangen van controle commando's van de USB

BULK transfer (device, endpoint)
 	
`pipe=usb_sndbulkpipe(dev,endpoint)`
 	
`pipe=usb_rcvbulkpipe(dev,endpoint)`
	
	Zenden van bulk-data naar de USB
	Ontvangen van bulk-data van de USB

Tabel 12: c-code voor USB-communicatie

Bijzonderheden die moesten worden onderkend:

Labview is een gebruiksvriendelijk programma dat tijdens de stageperiode nauwelijks problemen gaf. Zoals eerder vermeld zal de snelheid van de USB dalen tijdens het schrijven van data naar een bestand op de harde schijf. Dit is dan ook opgelost met de data in een tussenliggende array op te slaan en bij het beëindigen van de schrijfactie naar een bestand door de gebruiker de inhoud van de array naar een document te zetten.

8 Implementatie van QIE-test

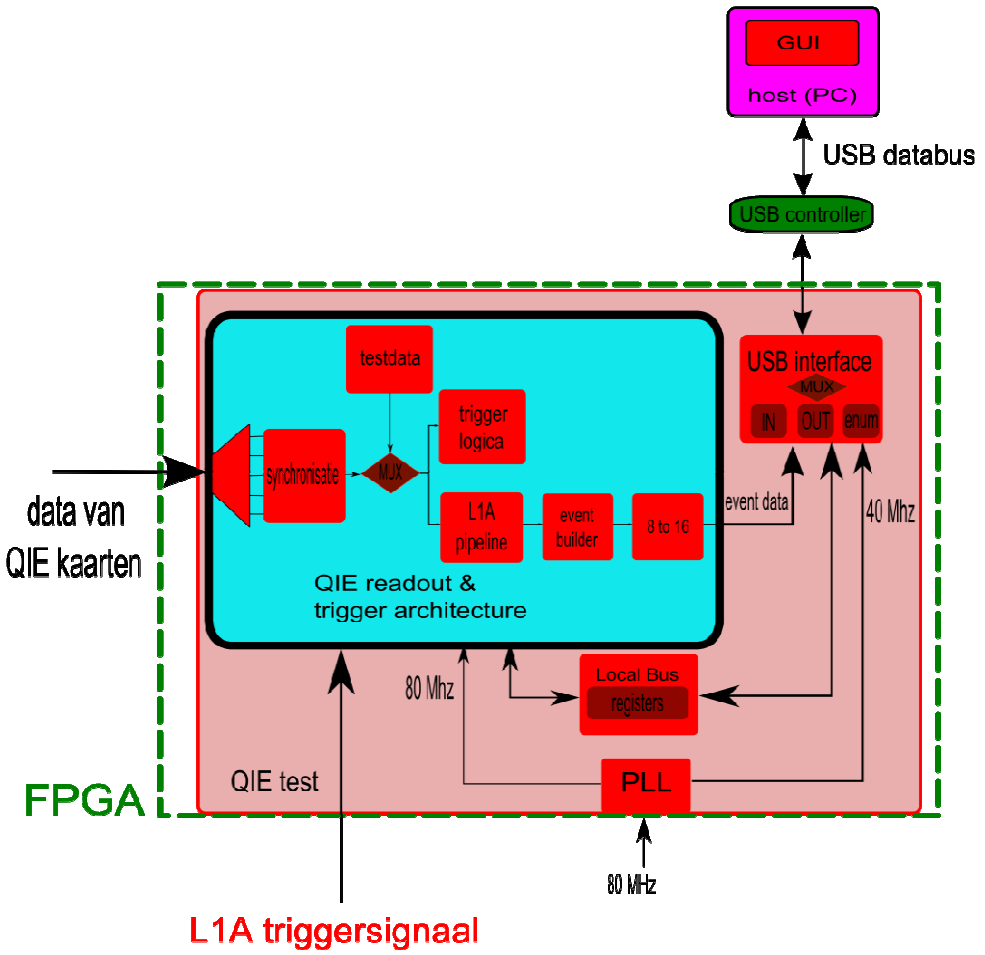
Doel: In het USBCOR-project wordt enkel de USB instellingen getest. Bij het QIE-test project zal het USB-interface blok aan het QIE readout & trigger worden gekoppeld. Er wordt nu testdata verzonden door dit QIE uitleesmechanisme via de USB-interface naar de GUI op een externe computer. De opzet van deze testopstelling is om datafouten tussen de het QIE uitleesmechanisme en de GUI op te sporen. Enkel het opzetten van de QIE-testopstelling valt binnen de stageopdracht. Het opsporen van datafouten hoort niet bij de doelstelling van de thesisopdracht.

Strategie:
Dit project omvat de QIE readout & trigger architectuur en het USB interfaceblok als één geheel. Doordat VHDL gebruikt maakt van entiteiten of blokken programmeercode is dit vrij eenvoudig door de juiste verbindingen in de toplaag te maken. Onderstaande figuur geeft een grafisch overzicht van het QIE-test project. Er wordt nu testdata gegenereerd in het QIE readout & trigger architectuur blok. De testdata zal nu ook worden getriggerd, verpakt en verzonden via USB. De databus in het QIE readout-trigger blok is 8 bit en van het USB-interface is 16 bit. Een eenvoudige 8-to-16 conversie proces in de top level zorgt voor deze omzetting. Doordat het QIE readout/trigger blok een interne klok van 80 MHz en USB-interface met een 40 MHz klok werkt is deze 8/16 mogelijk. Tabel 13 toont dit aan.

eniteit	klok	datasteam									
QIE readout/trigger	80 Mhz	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
USB interface	40 Mhz	D0&D1		D2&D3		D4&D5		D6&D7		D8&D9	

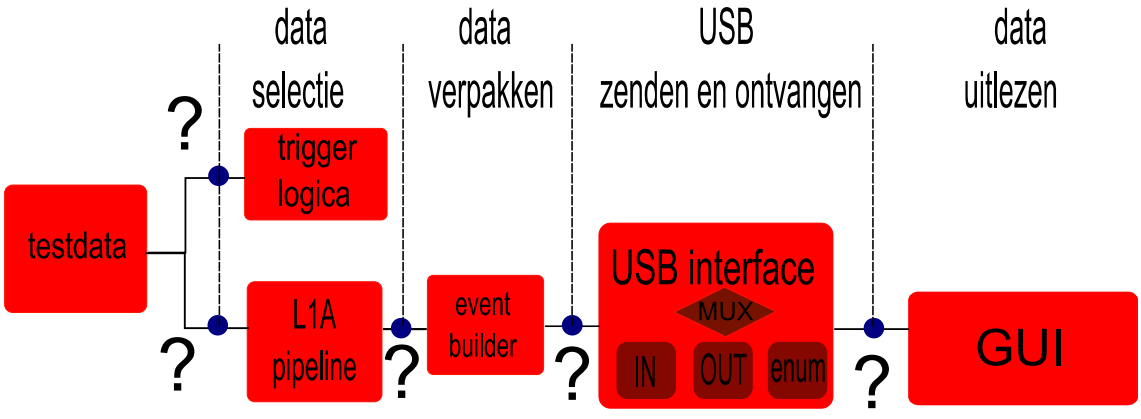
Tabel 13: werking 8 to 16 bit converter

Met de GUI kan de verzonden data gecontroleerd worden op datafouten.



Figuur 76: QIE-test

Bij een datafout kunnen er vijf plaatsen zijn waar er fouten zitten: trigger logica, L1A pipeline, fornatter (= event builder) ,de USB-interface of de externe GUI. Als de doorgestuurd testdata gelijk is aan de data in de GUI is zowel het triggeren en verpakken van data als het zenden via de USB-interface in orde .



Figuur 77: posities in QIE-test waar er zich fouten kunnen voordoen

9 Besluit

De primaire taak met name het configureren van een USB-controller om COR (CASTOR optische receiver) via USB uit lezen is een moeilijke opdracht omdat er weinig documentatie hier voor handen was. Toch is het nuttige taak omdat via de GUI kan getest worden of er geen foute data in het hele uitleessysteem van de CASTOR vervat zitten. Er kan nu test data aan het begin van het systeem worden aangebracht die dan alle VHDL-blokken doorloopt en uiteindelijk aankomt in een GUI via de USB-link. Fouten in de QIE-kaarten, trigger architectuur of op een andere plaats kunnen op deze wijze worden vastgesteld. Het uitleessysteem via de USB-link is dus een belangrijke bijdrage tot de hele COR module.

Lijst met figuren

Lijst met tabellen

Bibliografie

[1]. Adriaan, B. (2008). *Labview 8.5.* Geel, België: Campina Media vzw.

[2]. Arbab Jolfaei, F., Mohammadizadeh, N., Sadegh Sadri, M., & FaniSani, F. (2009). High Speed USB 2.0 Interface for FPGA Based Embedded Systems. Isfahan, Iran.

[3]. CERN. (2008). *A global endeavour*. Opgehaald van CERN: http://public.web.cern.ch/public/en/About/Global-en.html

[4]. CMS. (2008). *CMS detector overview*. Opgehaald van The Compact Muon Solenoid Experiment: http://cms.web.cern.ch/cms/Detector/FullDetector/index.html

[5]. CMS. (2010, maart 31). *Event Displays from the high-energy collisions at 7 TeV on 30th March 2010*. Opgehaald van CMS: http://cms.web.cern.ch/cms/Media/Images/EventDisplays/7_0TeVCollisions/index.html

[6]. Compaq, HP, Intel, Lucent, Microsfot, NEC, Philips. (2000, April 27). Universal Serial Bus revision2.

[7]. Cypress Semiconductor Corporation. (2009, Juli 7). EZ-USB SX2™ High Speed USB.

[8]. Esquenet, A. (2006, augustus 31). *USB 2.0, Hi Speed USB FAQ*. Opgehaald van everything USB: http://www.everythingusb.com/usb2/faq.htm

[9]. Haevermaet, H. V. (2009). Ontwikkeling van de Castor reconstructie software. Wilrijk, Antwerpen, België: UA.

[10]. Kennislink. (2009). *Deeltjesfabriek CERN gaat knallen*. Opgehaald van KennisLink: http://www.kennislink.nl/publicaties/deeltjesfabriek-cern-gaat-knallen

[11]. Mechelen, P. V. (2009, Januari 16). Opbouw van een deeltjesdetector.

[12]. P, V. (2009). enumeration Cy7C68001. CERN.

[13]. Peacock, C. (sd). *USB in a nutshell, Making sense of the USB standard*. Opgehaald van Beyond Logic: http://www.beyondlogic.org/usbnutshell/usb1.htm

[14]. Peterson, W. D. (1988). *The VME Handbook.* V F E A International Trade Association.

[15]. RTL Nieuws NL. (2010, Maart 31). Experiment deeltjesversneller geslaagd. *RTL NIEUWS* .

[16]. TechTarget. (sd). *IP core*. Opgehaald van whatis?com: http://whatis.techtarget.com/definition/0,,sid9_gci759036,00.html

[17]. UA. (2010). *Particle Physics group*. Opgehaald van UA - Elementaire Deeltjes Fysica: http://www.ua.ac.be/main.aspx?c=.EDF

[18]. Usselmann, R. (2002, januari 27). USB Fucntion IP Core. West-Samoa: opencores.org.

[19]. Wikipedia. (2010). *CERN*. Opgehaald van Wikipedia: http://nl.wikipedia.org/wiki/CERN

[20]. Wikipedia. (2010). *Fotomultiplicator*. Opgehaald van Wikipedia: http://nl.wikipedia.org/wiki/Fotomultiplicator

[21]. Wikipedia. (2009). *Large Hadron Collider*. Opgehaald van Wikipedia: http://nl.wikipedia.org/wiki/Large_Hadron_Collider

[22]. Wikipedia. (2010). *VMEbus*. Opgehaald van Wikipedia: http://en.wikipedia.org/wiki/VMEbus

Bijlagen

De bijlage bevat extra informatie zoals USB bits (bijlage A), de gebruikte drivers (bijlage B en C), de code van het project (D) en weggelaten delen uit de thesis die niet direct bijdragen tot de eigenlijke thesisopdracht maar wel kan dienen als de lezer meer informatie wenst. De weggelaten hoofdstukken zijn een overzicht van de USB versies(bijlage E), USB v2.0 specificaties(bijlage F) en een beschrijving van de USB-interface chip pinnen, registers en commando's(bijlage G).

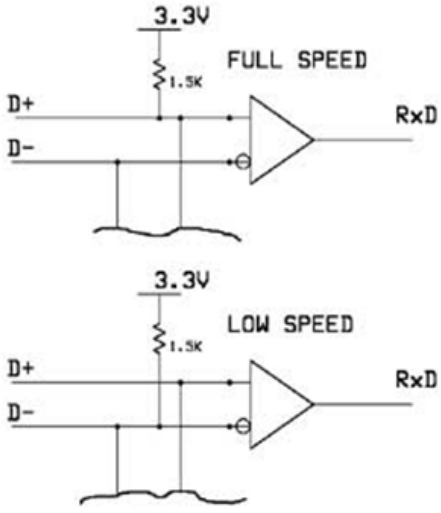
A. overzicht van USB bits

Bus State	Levels
Differential '1'	D+ high, D- low
Differential '0'	D- high, D+ low
Single Ended Zero (SE0)	D+ and D- low
Single Ended One (SE1)	D+ and D- high
Data J State: Low-speed Full-speed	 Differential '0' Differential '1'
Data K State: Low-speed Full-speed	 Differential '1' Differential '0'
Idle State: Low-speed Full-speed	 D- high, D+- low D+ high, D- low
Resume State	Data K state
Start of Packet (SOP)	Data lines switch from idle to K state
End of Packet (EOP)	SE0 for 2 bit times followed by J state for 1 bit time
Disconnect	SE0 for >= 2us
Connect	Idle for 2.5us
Reset	SE0 for >= 2.5 us

[13]

Figuur 78: USB bits

Het verschil tussen low –en full speed is de positie van de pull-up weestand die gekoppeld is aan D+ bij full speed en aan D- bij low speed. Dit verklaart ook het verschil van de USB bits (Figuur 78: USB bits) tussen de twee snelheden. High speed mode gebruikt dezelfde USB bits als die van full speed mode.



Figuur 79: pull-up weerstand bij low -en full speed

B. NI-VISA driver van Cy7c68001 (USB controller IC)

```
;=====================================================================
=====
;  This file was generated using:
;  NI-VISA Driver Development Wizard version 4.1
;=====================================================================
=====
;
;  This file is for use with Windows XP/2000. This will not work on
Windows
;  Vista.  This Windows Setup Information File contains the information
;  NI-VISA needs in order to access your instrument.  Do not modify the
;  contents of this file unless you are sure about what you are doing.
;
;=====================================================================
=====
;
;  Installation:
;  To apply the contents of this file to a system's settings, copy this
file
;  to %SystemRoot%\inf and reboot the computer.  If the device was
;  installed before this file is copied, the device will need to be
removed
;  from the "Unknown Devices" class in the Windows Device Manager.
;
;=====================================================================
=====
;
;  Removal:
;  Remove this file and its associated .PNF file (if it exists) from
;  %SystemRoot%\inf.  Either reboot the computer, or perform a
;  "Scan for hardware changes" from the Windows Device Manager.
;
;=====================================================================
=====

[Version]
Signature=$WINDOWS NT$
Class=visaUsbDevice
ClassGUID={A3330EDF-239D-4206-833B-1D58952613D5}
Provider=%Vendor0%
DriverVer=03/22/2010,1.0
CatalogFile=driver_cy7c68001.cat

[DefaultInstall.nt]
CopyFiles=NIVIUSBK_Device_CopyFiles_Inf

[DestinationDirs]
NIVIUSBK_Device_CopyFiles_Inf = 17

[NIVIUSBK_Device_CopyFiles_Inf]
driver_cy7c68001.inf

[SourceDisksNames]

[SourceDisksFiles]
```

```
[ClassInstall32.nt]
AddReg=AddClass_AddReg

[AddClass_AddReg]
HKR,,,0,%DeviceClassString%
HKR,,Icon,,"-20"
HKR,,IconPath,0x00010000,"%11%\setupapi.dll,-20"

[Manufacturer]
%Vendor1%=USBList

[USBList]
%USB\VID_1556&PID_017B.DeviceDesc%=NIVIUSBK_Inst, USB\VID_1556&PID_017B

[PreCopySection]
HKR,,NoSetupUI,,1

[NIVIUSBK_Inst]
AddReg=NIVIUSBK_Inst_AddReg

[NIVIUSBK_Inst.Services]
Addservice = NIVIUSBK, 0x00000002, NIVIUSBK_AddService

[NIVIUSBK_AddService]
DisplayName     = %NIVIUSBK.SvcDesc%
ServiceType     = %SERVICE_KERNEL_DRIVER%
StartType       = %SERVICE_DEMAND_START%
ErrorControl    = %SERVICE_ERROR_NORMAL%
ServiceBinary   = %12%\NIVIUSBK.sys

[NIVIUSBK_Inst_AddReg]

[Strings]
Vendor0="Cypress"
Vendor1="Cypress"
USB\VID_1556&PID_017B.DeviceDesc="CY7C68001"
DeviceClassString="NI-VISA USB Devices"
NIVIUSBK.SvcDesc="NI-VISA USB Driver"

SERVICE_BOOT_START = 0x0
SERVICE_SYSTEM_START = 0x1
SERVICE_AUTO_START = 0x2
SERVICE_DEMAND_START = 0x3
SERVICE_DISABLED = 0x4

SERVICE_KERNEL_DRIVER = 0x1
SERVICE_ERROR_IGNORE = 0x0
SERVICE_ERROR_NORMAL = 0x1
SERVICE_ERROR_SEVERE = 0x2
SERVICE_ERROR_CRITICAL = 0x3
```

C. x86 driver van Cypress voor Cy7c68001 (USB controller IC)

```
; Installation INF for the Cypress Generic USB Driver for Windows XP
; Processor support for x86 based platforms.
;
; (c) Copyright 2009 Cypress Semiconductor Corporation
;

[Version]
Signature="$WINDOWS NT$"
Class=USB
ClassGUID={36FC9E60-C465-11CF-8056-444553540000}
provider=%CYUSB_Provider%
CatalogFile=CYUSB.cat
DriverVer=06/05/2009,3.4.1.20

[SourceDisksNames]
1=%CYUSB_Install%,,,

[SourceDisksFiles]
CYUSB.sys = 1

[DestinationDirs]
CYUSB.Files.Ext = 10,System32\Drivers

[ControlFlags]
ExcludeFromSelect = *

[Manufacturer]
%CYUSB_Provider%=Device,NT,NTx86,NTamd64

;for all platforms
[Device]
;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX
%VID_1556&PID_017B.DeviceDesc%=CyUsb, USB\VID_1556&PID_017B

;for windows 2000 non intel platforms
[Device.NT]
;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX


;for x86 platforms
[Device.NTx86]
;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX


;for x64 platforms
[Device.NTamd64]
;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX


[CYUSB]
CopyFiles=CYUSB.Files.Ext
AddReg=CyUsb.AddReg

[CYUSB.HW]
AddReg=CYUSB.AddReg.Guid
```

```
[CYUSB.Services]
Addservice = CYUSB,2,CYUSB.AddService

[CYUSB.NT]
CopyFiles=CYUSB.Files.Ext
AddReg=CyUsb.AddReg

[CYUSB.NT.HW]
AddReg=CYUSB.AddReg.Guid

[CYUSB.NT.Services]
Addservice = CYUSB,2,CYUSB.AddService


[CYUSB.NTx86]
CopyFiles=CYUSB.Files.Ext
AddReg=CyUsb.AddReg

[CYUSB.NTx86.HW]
AddReg=CYUSB.AddReg.Guid

[CYUSB.NTx86.Services]
Addservice = CYUSB,2,CYUSB.AddService

[CYUSB.NTamd64]
CopyFiles=CYUSB.Files.Ext
AddReg=CyUsb.AddReg

[CYUSB.NTamd64.HW]
AddReg=CYUSB.AddReg.Guid

[CYUSB.NTamd64.Services]
Addservice = CYUSB,2,CYUSB.AddService


[CYUSB.AddReg]
; Deprecating - do not use in new apps to identify a CYUSB driver
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,CYUSB.sys
; You may optionally include a check for DriverBase in your application
to check for a CYUSB driver
HKR,,DriverBase,,CYUSB.sys
HKR,"Parameters","MaximumTransferSize",0x10001,4096
HKR,"Parameters","DebugLevel",0x10001,2
HKR,,FriendlyName,,%CYUSB_Description%

[CYUSB.AddService]
DisplayName    = %CYUSB_Description%
ServiceType    = 1                  ; SERVICE_KERNEL_DRIVER
StartType      = 3                  ; SERVICE_DEMAND_START
ErrorControl   = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary  = %10%\System32\Drivers\CYUSB.sys
AddReg         = CYUSB.AddReg
LoadOrderGroup = Base

[CYUSB.Files.Ext]
CYUSB.sys
```

```
[CYUSB.AddReg.Guid]
HKR,,DriverGUID,,%CYUSB.GUID%

[Strings]
CYUSB_Provider    = "Cypress"
CYUSB_Company     = "Cypress Semiconductor Corporation"
CYUSB_Description = "Cypress Generic USB Driver"
CYUSB_DisplayName = "Cypress USB Generic"
CYUSB_Install     = "Cypress CYUSB Driver Installation Disk"
VID_1556&PID_017B.DeviceDesc="Cypress USB Generic Driver (3.4.1.20)"
CYUSB.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
CYUSB_Unused      = "."
```

D. code

De gebruikte en gemaakte code kan teruggevonden worden op www.hep.ua.ac.be/castor/detector/TrigDaq/cdtc/usbreadout/ of op de bijgeleverde CD-ROM.

E. USB: extra informatie

a. USB-versies

USB heeft verschillende versies, aangegeven door een versienummer. Hogere USB versienummers beschrijven de USB interface met meer functionaliteit en een hogere snelheid.

Er zijn 4 officiële varianten van USB:

USB	Snelheid (Mbit/s)	Benaming	Jaar
1.0	**1,5**	**Low speed**	**1996**
1.1	**12**	**Full speed**	**1998**
2.0	**480**	**High speed**	**2000**
3.0	**4800 (= 4.8 Gbit/sec)**	**Super speed**	**2008**

Tabel 14: USB versies

De USB interface is al meer dan 15 jaar in ontwikkeling. Versie 0.7 van de USB interface definitie werd openbaar gemaakt in november 1994, en de eerste "echte" definitie van USB, USB 1.0 kwam in januari 1996 uit. Het was een gezamenlijke inspanning van enkele grote spelers op de markt om een nieuwe algemene apparaat interface te definiëren voor computers. Belangrijke stuwers van het project waren *Compaq*, *Intel*, *Microsoft* en *NEC*. De snelheid van USB 1.0 werd nog verbeterd wat leidde naar USB 1.1 versie (full speed, 1998)).

De USB 2.0-specificatie werd uitgebracht in april 2000 en werd gestandaardiseerd door de USB-IF[31] aan het einde van 2001. Hewlett-Packard, Intel, Lucent Technologies (nu Alcatel-Lucent na de fusie met Alcatel in 2006), NEC en Philips namen gezamenlijk het initiatief en dit leidde tot hogere data transfer rate dan de 1.1-specificatie (480 Mbit/s versus 12 Mbit/s).

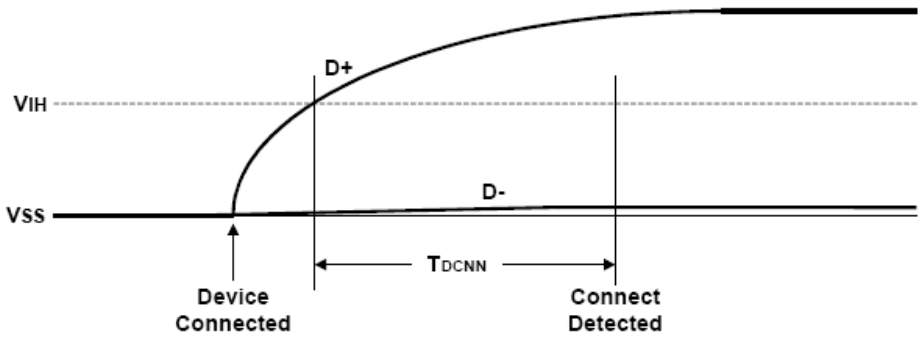
USBv3 is vooral bedoeld om apparaten (bvb. Blu-Ray) de bandbreedte te geven die ze nodig hebben. USBv3 genaamd SuperSpeed USB is de derde incarnatie van de seriële bus standaard met snelheden tot 4.8Gb/s wat tot tien keer sneller is dan de USB2-standaard. Deze kabel heeft 10 aders in plaats van 4 bij de vorige versies. USB 3 is compatibel met USB 2 en USB 1.1.

[31] **USB Implementers Forum**

F. USB v2 – high speed

a. Connectie

USB maakt connectie als volgt. Het D- signaal blijft net boven Vss lopen. Wanneer D+ boven Vih (tresshold spanning) loopt voor een bepaalde tijd (Tdcnn) zal er connectie met het USB-apparaat zijn.

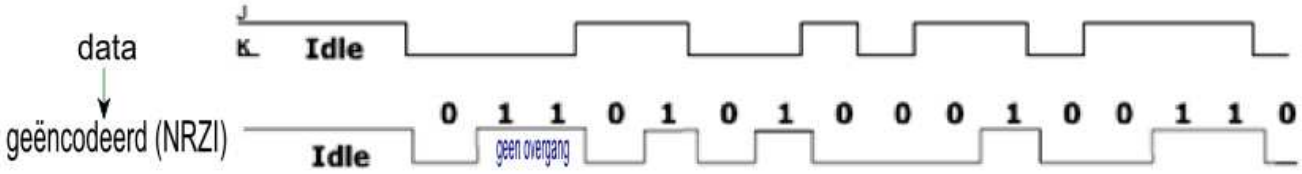


(Universal Serial Bus revision2, 2000)

Figuur 80: USB Connectie

b. Data encoding/decoding

Voor het vervoer van pakketten gebruikt USB NRZI[32]-encoding. Een '1' representeert geen verandering in de datastroom, een '0' geeft verandering in de datastroom weer. Een reeks nullen geeft dus een periode van wisselende bits weer(= toggeling bits). Een reeks enen toont een periode waarin er geen verandering van niveau is. In onderstaande figuur wordt dit weergegeven.



Figuur 81: NRZI encoding bij USB v2.0

De J en K bit wordt verkregen door differentiële spanning tussen de D+ en D- lijn. De J-toestand wordt bekomen door D+ gelijk aan hoog actief en D- gelijk aan laag actief. De K-bit is het invers van de J-bit, D+ is laag actief en D- hoog is actief. Meer informatie over de USB bits kan worden gevonden in Bijlage A: overzicht van USB bits.

c. IRP: I/O request packets

Een software client vraagt langs PC-zijde met behulp van I/O Request Packets (IRP) om data te versturen via de geselecteerde pipe. Vervolgens zal de client wachten totdat de dataoverdracht voldaan is of tot er een fout optreedt om eventueel een volgende IRP te zenden. Als er geen

[32] NRZI = Non Return to Zero inverted

wachtende IRP's zijn of IRP's bezig zijn met dataverwerking bevindt de pipe zich in rusttoestand. De IRP's zijn gedefinieerd met nummers en buiten data IRP heb je ook nog IRP_MJ_CLOSE, IRP_MJ_WRITE, IRP_MJ_DEVICE_CONTROL,…

IRP's vervoeren pakketten met een voorgedefinieerde grootte die bepaald is door de enumeratie[33] van het toestel. IRP buffers kunnen deze pakketten in 2 toestanden vullen:

- De buffergrootte is variabel. Bij een pakket kleiner dan deze buffergrootte zal het pakket toch verzonden worden
- De buffergrootte heeft een vaste waarde. Pakketten die kleiner of groter zijn dan de buffergrootte worden gezien als een error. De IRP wordt afgebroken en de pipe wordt in stall mode[34] gezet.

Een ENDPOINT kan laten weten dat het bezig is met het uitvoeren van acties en bewerkingen (= busy mode) door gebruik te maken van Negative-acknowledge character (= NAK). NAK wil echter niet zeggen er een fout is opgetreden maar enkel dat het endpoint in 'busy mode' staat.

d. USB 2.0 Pakketten

Hieronder volgt de beschrijving van belangrijkste velden van het USB-pakket: Sync, PID, ADDR, ENDP, CRC en EOP.

Sync

Alle pakketten moeten starten met dit sync-field. Het sync field is 32 bits lang en wordt gebruikt om de klok van de ontvanger te synchroniseren met die van de zender.

PID

PID staat voor 'Packet Indentifier' en geeft weer over welk type pakket het gaat. De PID bestaat uit 4 bits en om er zeker van te zijn dat de PID correct wordt ontvangen, worden deze 4 bits bij het zenden herhaald. Dit resulteert in een PID van 8 bits. Het resultaat wordt weergegeven in Tabel 16.

[33] Enumeratie = initialisatie van een toestel

[34] Stall mode = toestand die optreedt wanneer het endpoint van een toestel vastloopt door fouten meestal door verkeerde instelwaarden, time-outs of externe factoren

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

[13]

Tabel 15: USB pakket PIDs en zijn functies

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

[1

Tabel 16: 8 bit PID

ADDR

Dit veld geeft aan voor welk apparaat het pakket is voorzien. Het veld bevat 8 bits, wat ons (2^8 - 1 =) 127 mogelijke apparaten geeft. Adres 0 is niet geldig en wordt gebruikt door apparatuur die nog geen adres heeft. Dit soort apparatuur zal pakketten met adres 0 beantwoorden.

CRC: Cyclic redundancy check

Token CRC

Een vijf-bit CRC veld is voorzien voor token en beschermt de ADDR en ENDP velden van IN, SETUP en OUT tokens. De PING en SPLIT tokens bevatten ook een 5 bit CRC veld. De veelterm is:

$$G(X) = X(5) + X(2) + 1$$

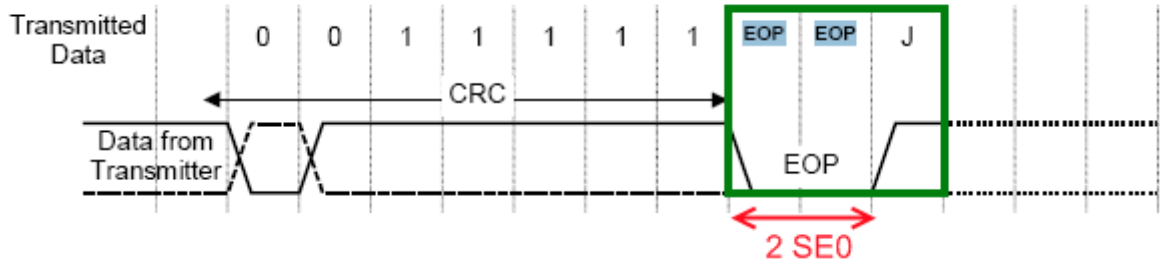
Data CRC

Het data CRC is een 16 bit veld dat heel het data veld van het datapakket beschermt tegen fouten. De polynomial of veelterm is:

$G(X) = X(16) + X(15) + X(2)$

EOP = End Of Packet

Bestaande uit 2 SE0 (Single Ended Zero), hierbij zijn zowel D+ als D- kleiner als 0.3 V. De 2 SE0 bits worden gevolgd door J-bit (D+ hoog; D- laag) voor 1 bit periode.



Figuur 82: EOP

Zoals eerder vermeld (4.2USB - Communicatie) heeft USB vier verschillende soorten pakketten. Onderstaande tekst zal deze pakketten verklaren en de mogelijk types die kunnen worden gebruikt weergeven. Deze types worden bepaald door de PID van het pakket.

Token: definieert welk transactietype er moet worden gevolgd.

- IN: informeert het USB-apparaat dat de gebruiker data wil lezen
- OUT: informeert het USB-apparaat dat de gebruiker data wil schrijven
- Setup: gebruikt om control transfer te starten

Sync	PID	ADDR	ENDP	CRC5	EOP

Figuur 83: formaat van een token pakket

Data: bevat de payload met een maximale payloadsize van 1024 bytes. Er zijn 4 soorten datapakketten.

- DATA0, DATA1, DATA2: 3 data pakketten van 1024 bytes geeft de max. grootte van 1 microframe (3072 bytes).
- DATAM: controle van pakketten bij isochroon OUT transport

Sync	PID	Data	CRC16	EOP

Figuur 84: formaat van een data pakket

Handshake: acknowledging van data en rapporteren van fouten

- ACK: bevestiging van een ontvangen pakket
- NAK: device kan tijdelijk geen data zenden of ontvangen.
- STALL: device heeft interventie van host nodig

Sync	PID	EOP

Figuur 85: formaat van een handshake pakket

SOF: Start van een nieuw frame. Bevat een 11-bit framenummer en wordt om de 125µs verzonden bij high-speed. Bij full speed zal het frame om de 1 ms worden verzonden.

Sync	PID	Frame Number	CRC5	EOP

Figuur 86: formaat van een SOF pakket

G. Beschrijving belangrijkste registers, commando's en signalen

FD – Field Data: De databus tussen SX2 logica en de FIFO. Deze bus kan 8 of 16 bit breed zijn. De bus werkt bidirectioneel.

FIFOADR – FIFO address: Dit signaal selecteert de gewenste FIFO. Er zijn standaard 5 FIFO's: 1 COMMAND (EP0), 2 OUT (EP2 en EP4) en 2 IN (EP6 en EP8) FIFO's. De breedte van de adreslijn is 3 bits. Er kan slechts 1 FIFO tegelijk geselecteerd worden.

Address/Selection	FIFOADR2	FIFOADR1	FIFOADR0
FIFO2	0	0	0
FIFO4	0	0	1
FIFO6	0	1	0
FIFO8	0	1	1
COMMAND	1	0	0
RESERVED	1	0	1
RESERVED	1	1	0
RESERVED	1	1	1

Tabel 17: FIFO adreslijnen

SLWR - Slave Write: Commando dat gebruikt wordt om data te schrijven in de interne IN-FIFO's van de USB-controller.

SLRD - Slave Read: Commando dat gebruikt wordt om data te lezen uit de interne OUT-FIFO's van de USB-controller.

SLOE – Slave Output Enable: Vermits de databus (FD) bidirectioneel is, zal dit signaal de richting van de databus bepalen. Bij het activeren van SLOE zal de databus in leestoestand staan. In niet-actieve toestand staat de databus in schrijftoestand.

PKTEND – Packet End: De USB transporteert zijn pakketten in microframes die om de 125 microseconden worden verzonden. In BULK-communicatie zal 1 microframe maximaal 6 BULK-pakketten van 512 bytes kunnen vervoeren. Deze BULK-pakketten worden eerst bewaard in de tussenliggende USB-FIFO. Als er BULK-pakketten kleiner de maximale waarde worden verstuurt (<512 byte) zal er gebruik worden gemaakt van het PKTEND-signaal. Het kleinere pakket wordt bevestigd en het microframe wordt vervolgens verzonden. PKTEND wordt voornamelijk gebruikt voor het verzenden van een klein aantal bytes in tegenstelling tot de datastromen die BULK-communicatie typeert. Zie ook 4.3.3 BULK transfers voor meer informatie omtrend BULK-communicatie.

WAKEUP: Bij geen busactiviteit of hervatting van de busactiviteit zal de USB zich in low power modus zetten. (= laag vermogenverbruik). Bij USB control-mode (control Endpoint) zal mits low-power toestand de USB-controller eerst terug geactiveerd moeten worden met het WAKEUP-signaal.

READY: Dit signaal geeft aan dat de USB-controller klaar is voor commando's of dataoverdracht. Het signaal wordt gebruikt bij control-modus van de USB-interface.

nRESET: Signaal dat de USB-controller reset.

nINT: Interruptsignaal, laag actief.

IFCLK: Klok ingang voor de SX2 (= USB-interface)

FLAGA: geeft de status weer van de Program Flag en wordt actief als de geselecteerde FIFO groter of kleiner is dan een bepaalde waarde die ingesteld kan worden door de gebruiker.

FLAGB: geeft de status weer van de Full Flag en wordt actief als de geselecteerde FIFO vol is.

FLAGC: geeft de status weer van de Empty Flag en wordt actief als de geselecteerde FIFO leeg is.