TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

Entwicklung eines Videosystems für das Furushiki2 Projekt

Ausgeführt am Institut für

Mechanik und Mechatronik

der Technischen Universität Wien

unter der Anleitung von

o.Univ.Prof. Dipl.-Ing. Dr. Dr.h.c.mult. Peter Kopacek

und

Dipl.-Ing. Dr. Bernhard Putz

durch

Lukas Wallentin

Bahnlände 43-45, 1100 Wien

Wien, am 18. November 2007

Danksagung

Mein Dank gilt meiner Familie, meinen Freunden und allen, die mich bei dieser Arbeit unterstützt haben.

Kurzfassung

Eine mögliche zukünftige Technik zur Erschaffung großer Strukturen im All sieht den Einsatz von Netzen als Basis vor. Die Netze werden im All gespannt und Roboter errichten auf diesen Netzen die eigentliche Konstruktion. Im Rahmen des Projektes Furoshiki2 soll das Verhalten der Roboter auf einem frei im All schwebenden Netz beobachtet werden. Dazu ist es notwendig die Roboter mit einem Videosystem auszustatten mit dem sich die vorliegende Diplomarbeit befasst. Es werden verschieden Panoramasysteme auf ihrer Verwendbarkeit in den Robotern untersucht, wobei schließlich die Wahl auf ein System mit vier Kameras gefallen ist. Aufgrund der Tatsache, dass die Roboter die Videostreams nicht direkt an die Erde senden können, wird die Rakete die die Roboter transportiert, als Relais verwendet. Sie kann jedoch nur das Equipment für die Übertragung eines analogen Videostreams beherbergen. Im Rahmen dieser Arbeit wird daher ein in Hardware realisierter digitaler Videokontroller vorgestellt, der zwei bzw. vier Videostreams nach dem Time Division Multiplexing Prinzip in Echtzeit zu einem vereint. Ebenso wird eine dazu passende Software vorgestellt, welche die Videodaten von einer Framegrabberkarte abgreift und diese getrennt darstellt. Weiterns ist die Software in der Lage die einzelnen eingebetteten Stream wieder getrennt abzuspeichern.

Abstract

One concept to build huge structures in space is to use nets as base for these constructions. The idea is to deploy a net in space and to use robots, which can move on the net, to build the construction on it. The goal of the Furushiki2 project is to learn more about the behavior of the robots while moving on the net in space and also about the reaction of the net. Therefore it is necessary to equip the robots with a video system. This thesis deals with such a video system. It starts with a comparison of different panorama systems by discussing the applicability of these systems on the robots. The result is that a system with four cameras would be most suitable. The problem is that it is not possible to transmit the video streams directly from the robots to earth. The rocket can carry the equipment for one analog video transmitter to earth. Hence it will be used as repeater for the video signals. This thesis presents a digital video multiplexer which can multiplex two and four signals to one using time division multiplexing. This way it is possible to transmit the video signals from the robots over the one channel of the rocket. It also presents a software which grabs the video stream from a frame grabber card and shows the pictures from the different cameras separately. With this software it is also possible to safe the extracted video streams

Abbildungs und Tabellenverzeichnis

Inhaltsverzeichnis

1. Einleitung

1.1 Ziele und Aufbau der Diplomarbeit

Im Sommer 2006 wurde das Institut für Handhabungsgeräte und Robotertechnik zu einem Workshop in Andøya eingeladen, um die mögliche Verwendung freier Nutzlastkapazitäten einer Höhenforschungsrakete für die Fortsetzung des Furoshiki Projektes zu untersuchen. Im Rahmen dieses Workshops wurde eine Mission ausgearbeitet und es wurde rasch klar, dass die Roboter, die verwendet werden sollen, mit einem Videosystem ausgestattet werden müssen.

Die vorliegende Arbeit befasst sich mit eben diesem Videosystem. Ziel ist es, ein Videosystem zu entwickeln, welches den Ansprüchen der Mission genügt und möglichst auch in weiteren Missionen verwendet werden kann. Nach einer Beschreibung der Gegebenheiten werden die Anforderungen analysiert und darauf aufbauend ein Videosystem entworfen. Anschließend werden kurz einige Videogrundlagen erklärt, worauf eine genaue Beschreibung des entwickelten Videocontrollers erfolgt. Danach wird noch die entwickelte Software erörtert. Die Arbeit schließt mit einem kurzen Ausblick auf mögliche Weiterentwicklungen.

1.2 Furoshiki

Ein Konzept zur Konstruktion großer Strukturen im All ist die Verwendung eines Furoshikisatelliten sowie der Einsatz von Robotern [Kay04]. Der Furoshikisatellit entfaltet mit Hilfe mehrerer kleinerer Satelliten ein Netz oder eine dünne Membran, auf der sich die Roboter bewegen können. Diese Roboter verwenden das Netz um verschiedene Module daran zu montieren und so die eigentliche Konstruktion zusammen zu setzen. Mit Hilfe dieser Technik ist es vorstellbar, Antennen mit mehreren hundert Meter Durchmesser zu erschaffen oder die Konstruktion eines „Solar Power Sattelite“ zur Energieerzeugung.

Am 22. Jänner 2006 fand der erste praktische Test eines solchen Systems statt [Sum06]. Im Rahmen dieses Testes wurde ein Parabelflug durch eine japanische Höhenforschungsrakete des Typs S310 durchgeführt. So wie die Rakete schwerelos war, wurde der Drall der Rakete auf Null reduziert. Anschließend wurde mit Hilfe dreier kleinerer Satelliten ein Netz aufgespannt, in dessen Zentrum sich die Rakete befand. Zur Beobachtung des Netzes war die Rakete mit zwei Kameras ausgestattet, eine dritte Kamera befand sich in einem der drei Satteliten. Nach dem das Netz gespannt war, fuhren die Roboter auf das Netz. Da allerdings das ganze Konstrukt nach dem Entfalten des Netzes zu schwanken begann, konnten die Roboter kaum beobachtet werden weil die Kameras nicht mehr richtig ausgerichtet waren und die Videoübertragung durch die Schwankungen einige Male abriss. Trotzdem konnte Aufgrund der Telemetrie-

daten festgestellt werden, dass sich ein Roboter zumindest für 25 Sekunden am Netz bewegte. Der andere Roboter bewegte sich rund fünf Sekunden.

2 Furoshiki2

In der Fortsetzung des Furoshiki Projekts sollen die Roboter verbessert und ihr Verhalten im Weltall ausführlicher studiert werden. Im Gegensatz zum ersten Projekt soll das Netz, auf dem sich der Roboter befindet, von der Rakete losgelöst sein und die Videodaten sollen direkt vom Roboter geliefert werden. Ein weiterer sehr interessanter Aspekt dieses Experimentes ist daher die Entwicklung und der anschließende Test eines Netzes, welches sich, auf Befehl des Roboters, selbst entfalten kann.

Das Furoshiki2 Projekt sollte zusammen mit anderen Experimenten mit der Höhenforschungsrakete HotPay2 im Jänner 2007 durchgeführt werden. Am 1. Juli 2006 stürzte die Rakete HotPay1 bereits kurz nach dem Start ab. Da die meisten Experimente an Bord der HotPay1 mit den Experimenten der HotPay2 in Zusammenhang stehen, wurde der Start von HotPay2 auf einen unbekannten Zeitpunkt verschoben. Die Durchführung des Furoshiki2 Experimentes an Bord dieser Rakete ist in Folge unsicher, da es, im Vergleich zu den Messungen die während des Fluges stattfinden sollen, ein hohes Risiko darstellt.

Wenn das Furoshiki2 Experiment an Bord einer anderen Rakete stattfindet, ist anzunehmen, dass der Versuch ähnlich ablaufen wird und daher ein Videosystem auf den Robotern auf jeden Fall Sinn macht. Bei so einem System kann der für das Experiment interessante Teil des Netzes nicht aus dem Blickfeld der Kameras gelangen, wie es beim ersten Experiment der Fall war. Für die Entwicklung des Videosystems wurde daher von einer Durchführung des Experiments auf der HotPay2 ausgegangen. In Kapitel 2.1 wird nun die Rakete HotPay2 sowie weitere Gegebenheiten erörtert. Anschließend werden die einzelnen Teile des Versuchsaufbaues kurz beschrieben und abschließend wird der Ablauf des Furoshiki2 Experimentes behandelt.

2.1 Gegebenheiten

HotPay2 ist die zweite Höhenforschungsrakete des eARI [ARR04] Projekts, die ursprünglich im Jänner 2007 gestartet werden sollte, um vor allem wissenschaftliche Messungen vorzunehmen. Die Leitung dieses Projekts wurde von der Andøya Rocket Range übernommen, wo auch die so genannte „Hotel Payload“, die Spitze der Rakete, welche alle Instrumente sowie Service Module beinhaltet, fabriziert und die Rakete schließlich auch gestartet wird. Diese Spitze ist in Abbildung 2-1 zu sehen.

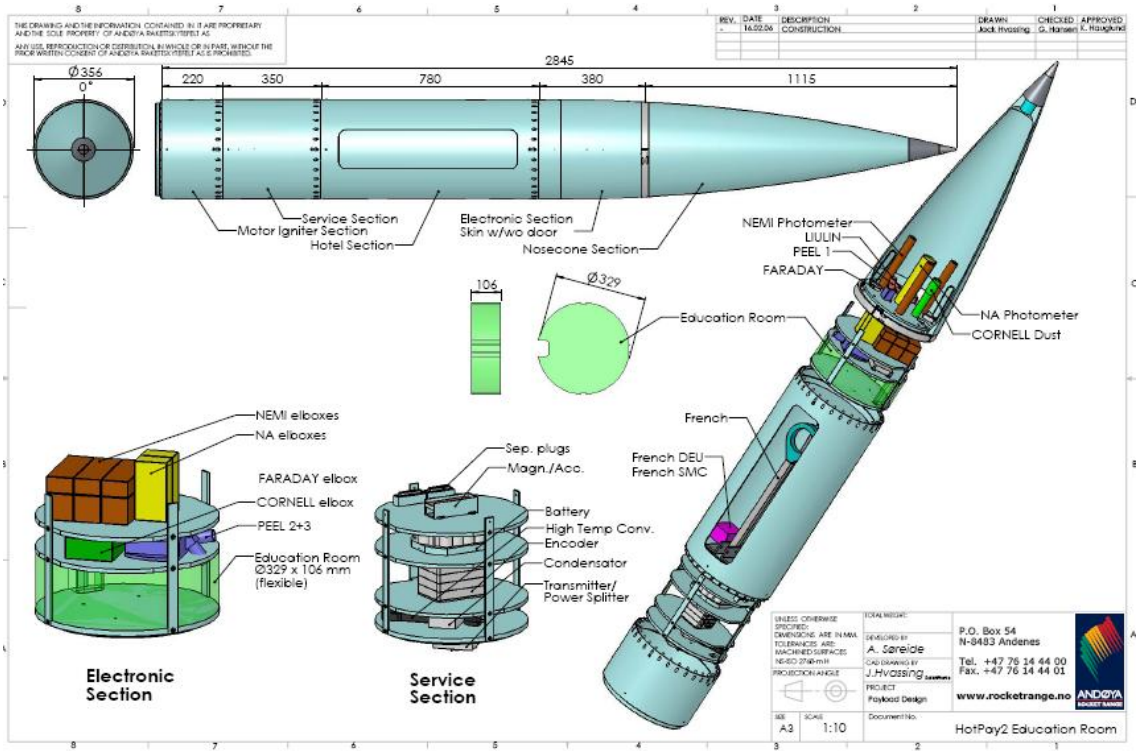

Abbildung 2-1 - HotPay2 Hotel Payload mit Education Room [ARR06]

Bei HotPay2 handelt es sich um eine zweistufige Feststoffrakete, die die in der 2,8 Meter langen Spitze (der Durchmesser beträgt 356mm) untergebrachte Nutzlast auf eine Höhe von 420km befördern soll. Dabei wird eine Spitzengeschwindigkeit von 2500m/s und eine maximale Beschleunigung von 20g erreicht. Da einige der Instrumente, die sich vor allem in der Nase der Rakete befinden, während des Fluges auf einen fixen Punkt ausgerichtet sein sollen, wird die Rakete mit Hilfe eines Dralles stabilisiert, der wahrscheinlich bei ungefähr 3 Umdrehungen pro Sekunde liegen wird.

Der für das Furoshiki2 Projekt zur Verfügung stehende Raum (grün eingezeichnet) befindet sich direkt über der „French Section“. Die Höhe des Raumes ist variierbar, 106mm ist nur ein Richtwert. Eine alternative Möglichkeit wäre die Platzierung eines Roboters in der Spitze der Rakete. Als maximales Gewicht für das Experiment wurde 5 kg genannt, so dass zwei Roboter verwendet werden können.

Einige Messungen, die während des Flugs durchgeführt werden sollen, erfordern besondere Bedingungen, welche auch auf das Furoshiki2 Experiment Auswirkungen haben. Wie bereits erwähnt, müssen einige Meßinstrumente möglichst während der kompletten Messung auf einen Punkt ausgerichtet werden. Es ist daher nötig die Rakete zu stabilisieren, was durch einen Drall bewerkstelligt wird. Die genaue Umdrehungszahl ist vor allem von einem Experiment abhängig, welches in der bereits erwähnten „French Section“ stattfindet und im Weiteren als

das französische Experiment bezeichnet wird. Dieses Experiment sieht unter anderem vor, dass zwei Messinstrumente mit Hilfe von zwei Teleskoparmen ausgefahren werden. Bei einem Test hielten die Arme jedoch nur einem Drall unter 4,5 Umdrehungen pro Sekunde stand und rissen bei einer höheren Umdrehungsgeschwindigkeit ab, worauf beschlossen wurde den Drall der Rakete von sechs auf die bereits erwähnten drei Umdrehungen pro Sekunde herabzusetzen.

Ein weiter Einfluss durch das französische Experiment auf das Furoshiki2 Experiment ist dahingehend gegeben, dass sich der Roboter, sobald das französische Experiment bei zirka 300 km Höhe beginnt, mindestens dreißig Meter von der Rakete entfernt haben sollte. Der Grund dafür ist der, dass das Experiment unter anderem eine Messung mit einem Magnetometer beinhaltet. Die Roboter bestehen allerdings aus zwei Teilen, die mit Magneten zusammengehalten werden. Hinzu kommt noch, dass auch die Motoren Magnetfelder erzeugen. Daher ist es notwendig, dass die Roboter vor Beginn des französischen Experimentes weit genug entfernt sind, um nicht mit der Messung zu interferieren. Die Funkanbindung der Roboter an die Rakete, auf die später noch genauer eingegangen wird, stellt aufgrund der hohen Frequenz des Funksignals kein Problem dar.

Zwei weitere Experimente benötigen des Weiteren einen möglichst dunklen Himmel. Dadurch ist auch der ursprünglich geplante Starttermin im Jänner zu erklären, da es zu dieser Zeit, auf Grund der geographischen Lage, in Andøya den ganzen Tag dunkel, sowie der Mond nicht sichtbar ist. Deshalb ist es nötig, das Netz im Blickfeld der Kameras auszuleuchten. Zu Problemen mit diesen lichtempfindlichen Experimenten sollte es nicht kommen, da die Sensoren, die sich in der Nase der Rakete befinden, nach vorne ausgerichtet sind und nur einen sehr schmalen kegelförmigen Bereich für die Messung verwenden. Die Roboter werden sich zur Zeit der Messung neben der Rakete befinden.

Eine letzte Einschränkung wäre noch, dass die Roboter, um eine andere Messung nicht zu stören, erst dann von der Rakete losgekoppelt werden dürfen, wenn diese die Höhe von 130 Kilometer überschritten hat.

Andere Bedingungen die für die anderen Experimente gegeben sein sollen, wie zum Beispiel das Vorhandensein eines stabilen Aurora Armes oder klares Wetter, haben keinen Einfluss auf das Furoshiki2 Experiment.

2.2 Versuchsaufbau

Im Weiteren wird nun konkret auf die einzelnen Teile des Projekts eingegangen.

2.2.1 Roboter

Als Roboter wird eine Weiterentwicklung der Roboter verwendet werden, die bereits erfolgreich beim ersten Furoshiki Projekt [Sum06] verwendet wurden. Er wird, wie bereits sein Vorgängermodell, aus zwei Hälften bestehen, die durch Magnete zusammen gehalten werden. Es berühren sich nur die Raupen der beiden Hälften, so dass es nun für den Roboter möglich ist, sich auf einem sich zwischen den beiden Teilen befindlichen Netz zu bewegen.

Im Gegensatz zum Vorgängermodell wird das Unterteil des Roboters bei diesem Experiment ausgebaut. Es wird ein Videosystem beherbergen, wodurch das komplette Netz beobachtet werden kann, sowie das dazugehörige Sendemodul mit den Antennen. Da, wie bereits bei den Gegebenheiten erwähnt, das Experiment in beinahe vollkommener Dunkelheit stattfindet, wird der Unterteil auch mit LEDs ausgestattet sein, um das Blickfeld der Kameras auszuleuchten.

Eine weitere Aufgabe des Unterteils wird die Aktivierung des Entfaltungsmechanismuses des Netzes sein, sobald der Roboter und das Netz sich in ausreichend großer Entfernung von der Rakete befinden. Solange sich der Roboter in der Rakete befindet, wirkt aufgrund des Dralls eine Zentrifugalbeschleunigung auf ihn ein, die mit einem Beschleunigungssensor gemessen werden kann. Sobald der Roboter allerdings von der Rakete losgekoppelt ist, wirkt sie nicht mehr auf ihn ein, so dass das erfolgreiche Entriegeln durch eine signifikante Änderung des durch den Beschleunigungssensor gemessenen Wertes erkannt werden kann. Nun kann ein Zeitmesser gestartet werden, der, falls nun für eine gewisse Zeitspanne keine Beschleunigung gemessen wird, den Entriegelungsmechanismus für das Netz aktiviert. Der Einsatz des Beschleunigungssensores bietet den Vorteil, dass falls aus irgendwelchen Gründen es nicht zu einer Entkopplung des Roboters von der Rakete kommt, dies erkannt werden kann und in Folge das Netz nicht entfaltet wird. Zur Sicherheit wird der Roboter noch mit Kontakten mit der Rakete verbunden, so dass das Verlassen der Rakete durch eine Unterbrechung der Kontakte zusätzlich noch erkannt werden kann. Für die Energieversorgung des Unterteils wird auch noch eine Batterie eingebaut.

2.2.2 Release Mechanismus

Der Release Mechanismus ist der wahrscheinlich kritischste Part an dem kompletten Experiment, da, falls sich ein Roboter nur teilweise von der Rakete löst, es zu einer geringen Verlagerung des Schwerpunktes kommen kann. Das hätte einen Einfluss auf den Drall der Rakete, was möglichst vermieden werden sollte. Wie bereits erwähnt kann es aber trotzdem nicht zu einem unerwünschten Entfalten des Netzes kommen.

Für den Release Mechanismus selbst ist geplant, die Zentrifugalkraft zu verwenden. Das hat den Vorteil, dass ein relativ einfacher Mechanismus konstruiert werden kann, wodurch die Gefahr eines Fehlschlages minimiert wird. Er wird im Grunde darauf beruhen, dass entweder ein Roboter samt Netz und ein Gegengewicht, oder dass zwei Roboter mit ihren Netzen losgelassen werden. Allein durch die Zentrifugalkraft werden die Roboter aus der Rakete gezogen, von der sie sich mit konstanter Geschwindigkeit wegbewegen. Die Geschwindigkeit würde abhängig von der Umdrehungsgeschwindigkeit der Rakete zwischen 3.4 und 6.7 m/s liegen, wobei aufgrund des angestrebten Dralles von der Umdrehungen pro Sekunde ein Wert nahe 3.4 m/s wahrscheinlicher ist.

Die Verwendung zweier Roboter hätte den Vorteil, dass zum einen zwei verschiedene Entfaltungsmechanismen für die Netze getestet werden könnten und zum anderen, dass im Falle eines Defekts eines Roboters oder des Entfaltungsmechanismuses eines Netzes, zumindest Daten von dem anderen Roboter gewonnen werden können.

2.2.3 Netz

Das Netz und vor allem der Entfaltungsmechanismus des Netzes spielen eine zentrale Rolle bei dem Furoshiki2 Projekt. Es ist geplant ein rundes Netz zu verwenden, dessen Rand zum Beispiel durch Kunststoff oder mit Hilfe eines anderen Materials gespannt wird. Der Roboter wird nun in die Mitte des Netzes gesetzt, so dass sich das Netz zwischen den beiden Teilen des Roboters befindet. Anschließend wird das vorgespannte Netz unter dem Roboter zusammengefalten und fixiert. Dieses Packet wird in der Rakete verstaut, wo es durch den Release Mechanismus gehalten wird, bis die Höhe erreicht wird, in der das Packet freigegeben werden soll. Wenige Sekunden später löst der Roboter die Fixierung und das Netz spannt sich nun aufgrund der Vorspannung außerhalb der Rakete von alleine wieder auf.

Da sich das gesamt Paket nach dem Verlassen der Rakete um den eigenen Schwerpunkt drehen wird, besteht auch die prinzipielle Möglichkeit, diese Rotation zur Entfaltung des Netzes

auszunutzen. Eine Möglichkeit wäre etwa ein eckiges Netz zu verwenden und an den Enden kleine Massen anzubringen. Sobald die Fixierung des Netzes gelöst wird, spannen die Massen das Netz aufgrund der Zentrifugalkraft das Netz auf, worauf im Gegenzug die Geschwindigkeit, mit der sich der Roboter und das Netz drehen, abnimmt.

Wie das Netz am besten gefalten wird, soll im Vorfeld durch eine Versuchsreihe festgestellt werden, ebenso welches Material sich besonders gut zum Spannen des Netzes eignet. Das Netz selbst wird mit einer reflektierenden, am besten fluoreszierenden, Farbe eingefärbt, um möglichst gut in der Videoübertragung erkennbar zu sein.

2.2.4 Video und Datenkommunikation

Es ist geplant für die Kommunikation die Rakete als Relais zu verwenden, da aufgrund der geringen Größe des Roboters es nicht möglich ist, einen ausreichend starken Sender auf diesem unterzubringen. Ein weiterer Grund ist, dass es nicht geplant ist, den Roboter relativ zur Erde zu stabilisieren. Durch Verwendung zweier Antennen ist es möglich fast gleichmäßig in alle Richtungen zu strahlen, wodurch auch keine Stabilisation im Bezug zur Rakete nötig ist.

Der Roboter wird zwei verschiedene Kommunikationssysteme verwenden. Zum einen wird er digitale Daten, die Telemetriedaten, senden, zum anderen ein analoges Videosignal. Die Telemetriedaten werden, wie beim letzten Experiment von der oberen Hälfte des Roboters zur Rakete gesendet, wo sie, zusammen mit den Messdaten der anderen Instrumente, über das Sendemodul (max. 1.25 MBit/s) der Rakete zur Erde gefunkt werden. Auch die Videodaten sollen mit Hilfe der Rakete zur Erde gesendet werden. Das Übertragungssystem von der Rakete zur Erde wird von den Herstellern der Rakete geliefert, das restliche Videosystem wird nun ab Kapitel 3 genauer behandelt.

2.3 Versuchsablauf

Ursprünglich war der folgende Ablauf vorgesehen: Ab dem 13. Jänner 2007 wird täglich der Countdown gestartet und abgebrochen, falls die gewünschten Startbedingungen nicht gegeben sind. Einige Minuten vor dem Start wird die externe Stromversorgung von der Rakete getrennt. Zu diesem Zeitpunkt werden die zwei Roboter und ihre Beleuchtung aktiviert und sie beginnen bereits Videodaten zum Relais zu übertragen, welches allerdings die Bilder noch nicht weiter leiten kann.

Sind die Startbedingungen gegeben, startet die Rakete. Sobald eine Höhe von 70 Kilometern erreicht wird und die erste Stufe ausgebrannt ist, wird die Ummantelung der Nase gelöst, so dass nun die Instrumente mit den Messungen beginnen können. Ab diesem Zeitpunkt steht auch die Videoverbindung zu den Robotern. Bis zu einer Höhe von 130 km soll nun nichts weiter geschehen, um nicht die Messungen zu beeinflussen. Sobald die 130 km erreicht sind, werden die Tore der French Sektion und des Educationrooms, in dem sich das Furushiki2 Experiment befindet, entfernt. Anschließend wird der Release Mechanismus der Roboter aktiviert und sie verlassen samt den vorgespannten Netzen die Rakete, von der sie sich nun mit einer konstanten Geschwindigkeit von 3.4 m/s entfernen. Zirka 9 Sekunden später sind die Roboter so weit entfernt, dass sie mit dem Französischen Experiment nicht mehr interferieren können.

Falls es nun zu Problemen mit dem Furoshiki2 Experiment kommen sollte, hat dies keinen weiteren Einfluss mehr auf die anderen Experimente. Nachdem einige Zeit die Roboter keine Beschleunigung mehr messen konnten, werden die Entfaltungsmechanismen der Netze aktiviert, welche sich daraufhin selbständig entfalten. Dieser Vorgang kann mittels des Videosystems genau beobachtet werden.

Ungefähr zur selben Zeit werden auf der Rakete die Arme des Französischen Experiments aktiviert, da dieses beginnen soll, sobald die Höhe von 300 km erreicht ist. Das hat allerdings keinen weiteren Einfluss auf das Furoshiki2 Experiment.

Wenige Sekunden nachdem die Mechanismen zum Entfalten der Netze aktiviert wurden, setzen sich die Roboter in Bewegung und fahren eine zuvor programmierte Route am Netz ab. Die Bilder der nächsten Minuten werden Aufschluss über das Verhalten der Roboter auf den Netzen liefern bis schließlich der Videokontakt zu den Robotern abreisst. Danach wird man nur mehr die Telemetriedaten der Roboter empfangen bis schließlich die Rakete und auch die Roboter wieder in die Atmosphäre eintreten. Damit ist das Furoshiki2 Experiment beendet und die gewonnen Daten können in der darauf folgenden Zeit ausgewertet werden.

3 Videosystem

In diesem Kapitel wird der Aufbau des Videosystems genau behandelt. Beginnend mit einer genauen Anforderungsanalyse in Kapitel 3.1 werden anschließend verschiedene Panorama und omnidirektionale Systeme vorgestellt und ihre mögliche Verwendung im Furoshiki2 Projekt diskutiert. Im Kapitel 3.3 wird schließlich das System welches implementiert wurde vom Aufbau her beschrieben. Abschließend wird noch eine mögliche Konfiguration des gesamten Systems vorgestellt.

3.1 Anforderungen

Wie bereits Kapitel 2 zu entnehmen war, soll beim Furoshiki2 Projekt das Verhalten des Roboters auf einem Netz beobachtet werden. Im Gegensatz zum vorherigen Projekt sollen die Bilder vom Roboter selbst geliefert werden. Da sich der Roboter anfangs im Zentrum des Netzes befindet, und möglichst das komplette Netz beobachtet werden soll, ist es notwendig ein Videosystem zu entwickeln, welches einen Sichtbereich von 360° um den Roboter herum abdeckt.

Weiters soll das Videosystem kompakt sein und nicht aus dem Roboter herausragen. Der Grund dafür ist, dass durch das Entfalten des Netzes oder durch eine Bewegung des Roboters es dazu kommen kann, dass sich kurzzeitig Teile des Netzes um den Roboter wickeln. Steht nun ein Teil aus dem Roboter hervor besteht die Gefahr, dass sich der Roboter im Netz verheddert, was das Ende dieses Experimentes bedeuten würde.

Aufgrund der geringen Größe der Roboter und vor allem dadurch, dass sie nicht im Bezug auf die Erde stabilisiert sein werden, kann das Videosignal nicht direkt zur Erde gesendet werden. Allerdings ist die Rakete im Bezug zur Erde stabilisiert und kann auch das benötigte Equipment aufnehmen um als Relais zu fungieren. Daher ist es sinnvoll, das Signal über die Rakete zu leiten wobei nun auch dafür eine entsprechende Komponente entwickelt werden muss. Dabei ist darauf Rücksicht zu nehmen, dass eventuell zwei Roboter verwendet werden. Jeder dieser Roboter liefert ein Videosignal, allerdings kann nur ein Videosignal von der Rakete zur Erde gesendet werden. Also muss die Relaiskomponente in der Lage sein, zumindest zwei Signale zusammenfassen zu können.

Eine letzte Beschränkung besteht darin, dass keine beweglichen Teile verwendet werden sollen. Würde beispielsweise eine Kamera verwendet werden die an einem Servo angebracht ist, und dieser würde diese Kamera nun nach links drehen, dann würde sich der Roboter im Gegenzug nach rechts drehen. Diese Bewegung des Roboters würde sich nun auf das Netz über-

tragen und es ebenfalls in Bewegung versetzen. Da allerdings das Verhalten des Netzes auf die Fortbewegung des Roboters untersucht werden soll, ist es sinnvoll andere Bewegungen des Roboters zu vermeiden.

Interessant ist, dass bis auf die Einschränkung, dass nur ein Videosignal von der Rakete zur Erde gesendet werden kann, die Anforderungen vor allem durch das Experiment selbst, nicht aber durch die Rakete geben sind. Es ist daher anzunehmen, dass auch bei einer Durchführung des Experiments auf einer anderen Rakete die Ansprüche an das Videosystem recht ähnlich wären.

3.2 Auswahl des Kamerasystems

Um das komplette Netz beobachten zu können, bietet sich die Verwendung von Panorama beziehungsweise von omnidirektionalen Systemen an. Ausgehend von den verschiedenen Techniken zur Herstellung von Panoramabildern werden auch von diesen Techniken abgeleitete Systeme diskutiert. Dabei werden zuerst Systeme mit einem einzelnen Sichtpunkt betrachtet und anschließend Systeme mit mehreren. Eine gute Übersicht über die verschiedenen Panorama-Systeme bietet [Ben01].

3.2.1 Systeme mit einem optischen Zentrum

Systeme mit einem optischen Zentrum beziehungsweise einem Projektionszentrum haben den Vorteil zu Systemen mit mehreren optischen Zentren, dass mit ihrer Hilfe Bilder mit einer geometrisch korrekten Perspektive erzeugt werden können [Nay97]. Da die genaue Verzerrung durch das Aufnahmesystem bekannt ist, kann von jedem Bildpunkt des omnidirektionalen Bildes die Position von diesem Punkt auf einer ebenen Fläche berechnet werden. Die Fläche kann sich, wie in Abbildung 3-1 gezeigt, in beliebiger Entfernung zum optischen Zentrum befinden.

Systeme mit einem optischen Zentrum verwenden zwangsweise nur eine Kamera, da sich zwei Kameras nicht exakt am selben Ort befinden können. Das hat den Vorteil, dass auf einem Roboter nur ein einziges Videosignal erzeugt wird, wodurch im Falle, dass nur ein Roboter verwendet wird, das gesamte Videosystem recht einfach gehalten werden kann.

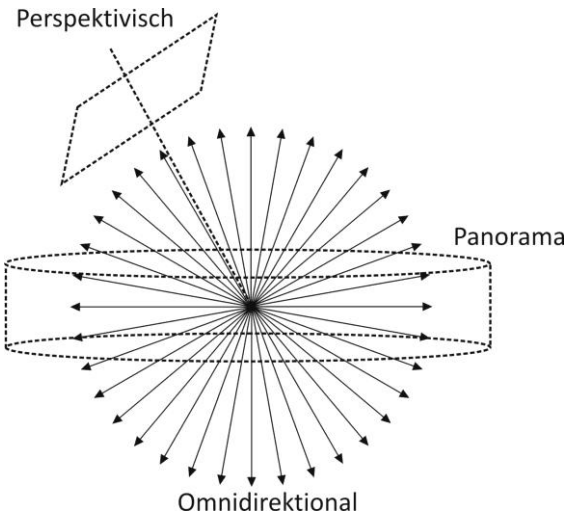

Abbildung 3-1 - Vergleich von Omnidirektional, Perspektivisch und Panorama

3.2.1.1 Rotierende Systeme

Eine Möglichkeit ein Panorama Bild zu erzeugen ist die, dass man eine Kamera verwendet die um das eigene optische Zentrum, welches in diesem Zusammenhang oft auch als Nodal oder Knotenpunkt bezeichnet wird, gedreht wird. Dabei wird Stück für Stück die Umgebung abfotografiert. Da alle Bilder von dem gleichen optischen Zentrum aufgenommen werden, können die Bilder problemlos zusammengesetzt werden. Die digitale Fotokamera ePan [Sea00] setzt etwa diese Technik ein, wobei hier nicht die Kamera selbst gedreht werden muss, sondern Teile der Kamera sich selbstständig um das optische Zentrum bewegen.

Einer der größten Nachteile dieser Technik ist der, dass die einzelnen Bilder nicht gleichzeitig gemacht werden. Dadurch, dass die Kamera für jedes Bild ein Stück weiter gedreht werden muss und danach alle Bilder erst zusammen gesetzt werden, dauert es recht lange bis schließlich ein komplettes Panoramabild zur Verfügung steht. Daher kann diese Technik nur verwendet werden, um statische Szenen aufzunehmen und kann auch nicht in Echtzeitanwendungen verwendet werden [Nay97]. Da das Videosystem zur Beobachtung des Netzes verwendet werden soll und es sich daher nicht um eine statische Szene handelt, kann mit dieser Technik kein Panorama erzeugt werden.

Natürlich ist es für die Beobachtung des Netzes nicht unbedingt nötig die Einzelbilder zu einem Panoramabild zusammenzusetzen. Man könnte die rotierende beziehungsweise schwenkende Kamera verwenden um das Netz direkt zu beobachten. Je nach Bewegungsgeschwindigkeit

bleiben zwar Teile des Netzes für einige Zeit unbeobachtet, allerdings könnte die Bewegungsgeschwindigkeit so weit erhöhen werden, dass dies kein wesentliches Problem mehr darstellt.

So ein System ist allerdings insofern problematisch, als dass es sich dabei um ein bewegliches System handelt, und es daher den geforderten Ansprüchen nicht gerecht wird. Des Weiteren wäre auch die Positionierung auf dem Roboter kritisch. Da die Wände des Roboters nicht transparent sind, müsste dieses System auf den Roboter aufgebaut werden, was ein weiterer Verstoß gegen die Anforderungen wäre.

3.2.1.2 Katadioptrische Systeme

Katadioptrische oder Spiegellinsen Systeme sind Systeme die Spiegel verwenden um das Blickfeld zu vergrößern. Die bekannte Zeichnung „Hand with Reflecting Sphere" des Künstlers MC Eschers aus dem Jahr 1935 verdeutlicht das Prinzip.

In den 1970ern wurde ein Spiegelsystem für Spiegelreflex Kameras zur Erzeugung von Panoramen entwickelt. Eine zu diesem System passende Technik erlaubte es in der Dunkelkammer die aufgrund des Spiegels verzerrten Panoramabilder zu entzerren [Cha97]. Damit wurde dieses Prinzip zur Erzeugung von Panoramaaufnahmen in der Fotografie eingeführt. Mit der Zeit entwickelten sich verschiedene Spiegelsysteme, die je nach Bauform verschiedene Sichtbereiche besitzen. Abbildung 3-2 zeigt exemplarisch den Aufbau und Sichtbereich eines Systems mit konischem Spiegel (a) sowie mit einem sphärischen Spiegel (b). Eine gute Übersicht zu den verschiedenen Spiegelformen bieten [Bru00] und [Ish98].

Katadioptrische Systeme haben einige Vorteile im Vergleich zu rotierenden Systemen. Das Bild entsteht aus einer einzigen Aufnahme und wird nicht wie bei den rotierenden Systemen aus mehreren, zeitlich versetzten Aufnahmen zusammen gesetzt. Daher eignen sich diese Systeme zur Beobachtung von bewegten Szenen. Anwendungsgebiete für solche Systeme sind zum Beispiel Videokonferenzsysteme oder Überwachungssysteme [Nay98] aber auch in der mobilen Robotik finden diese Systeme Einsatz, wo sie zum Beispiel zur Erkennung von Hindernissen verwendet werden [Ish98].

Da diese Systeme inzwischen in recht kompakter Bauform erhältlich sind, würde von der Größe her nichts gegen den Einsatz im Roboter sprechen. Auch die Tatsache, dass man gleichzeitig das gesamte Netz im Blickfeld hätte, spricht für das System. Allerdings stellt sich wiederum die Frage, wie man dieses System in den Roboter einbauen kann ohne die Form wesentlich zu verändern. Die Kamera selbst könnte natürlich im Roboter versenkt werden, allerdings würde

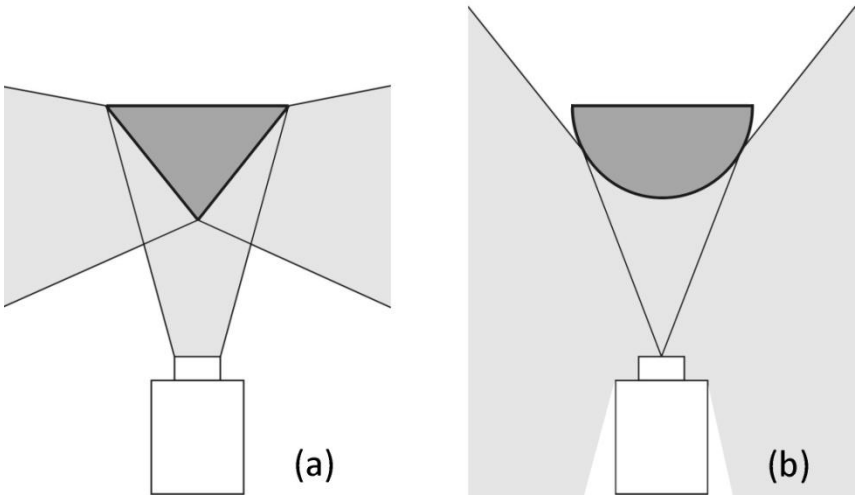

Abbildung 3-2 - konische und sphärische Spiegelsysteme

der Teil des Systems, der den Spiegel beherbergt, zwangsweise aus dem Roboter herausragen. Trotz der oben angeführten Vorteile von diesem System, wurde daher beschlossen, dieses System nicht zu verwenden.

3.2.1.3 Fischaugen

Fischaugen [Kum00] sind Linsen mit einer sehr kurzen Brennweite und einem sehr großen Blickwinkel. Mit diesen Linsen ist es möglich Objektive zu bauen, mit denen sich alles, was sich innerhalb eines Bereiches einer Halbkugel befindet, abbilden lässt. Es existieren einige wenige Objektive, die einen noch größeren Sichtbereich besitzen.

Ein Nachteil ist allerdings, dass es sehr schwer ist Fischaugen zu erzeugen, bei denen sich alle eintreffenden Strahlen in einem Punkt treffen. Das hat zur Folge, dass aus den Bildern, die man durch die Fischaugen erhält, keine verzerrungsfreien Bilder erzeugt werden können [Nay97]. Ein weiterer Nachteil von Fischaugen ist, dass sie relativ groß und daher auch recht schwer sind [Kum00], wodurch sie nicht unbedingt für einen Anwendungsbereich geeignet sind, wo das Gewicht eine wichtige Rolle spielt.

Auch wenn das Gewicht nicht das Problem wäre, stellt sich die Frage wo man so ein System mit einem Fischauge am besten platziert. Eine Möglichkeit wäre, die Linse auf der Unterseite des Roboters zu montieren, und die Kamera im Roboter zu versenken, wie dies in Abbildung 3-3 (a) dargestellt ist. Geht man davon aus, dass das Fischauge einen Blickbereich von 180 Grad besitzt, kann man so zumindest die Teile des Netzes beobachten, die sich sozusagen

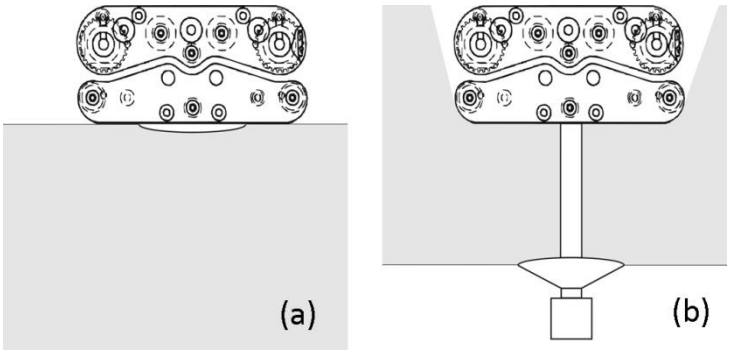

Abbildung 3-3 - Roboter mit Fischaugensystem. Der Sichtbereich ist hellgrau eingezeichnet

oberhalb des Roboters befinden. Wie in den Anforderungen beschrieben, soll möglichst nichts aus dem Roboter herausstehen. Bei dieser Konstruktion ist dahingehend anzumerken, dass wirklich nur die Oberfläche der Linse aus dem Roboter heraus steht, die keine Kanten und Ecken besitzt an der sich das Netz leicht verheddern könnte. Was allerdings neben dem Gewicht viel mehr gegen diese Konstruktion spricht ist, dass eigentlich nur ein Bruchteil des Blickfeldes zur Beobachtung genutzt wird und nur die Teile des Netzes zu sehen sind, die sich momentan unterhalb der Bewegungsebene des Roboters befinden.

Um das Netz um den Roboter herum zu beobachten, müsste sich die Kamera, wie in Abbildung 3-3 (b) dargestellt, auf einem kleinen Mast befinden und zum Roboter hinauf sehen. Neben einiger anderer Probleme die sich durch eine solche Konstruktion ergeben würden, wäre hier wiederum die Möglichkeit gegeben, dass sich das Netz an dem Mast verheddert.

Zusammengefasst bietet sich ein Panorama System mit einem Fischauge sowohl wegen des Gewichts und Größe der Linsen, als auch wegen der Schwierigkeit, so ein System vernünftig zu positionieren, nicht an.

3.2.2 Systeme mit mehreren optischen Zentren

Neben der Möglichkeit, nur eine Kamera zu verwenden und durch Zuhilfenahme verschiedener Techniken das Sichtfeld zu vergrößern, gibt es auch noch die Möglichkeit, mehrere Kameras zu verwenden. Da die Kameras allerdings verschiedene optische Zentren besitzen, ist es nicht möglich, aus den Einzelbildern ein Bild mit einer geometrisch korrekten Perspektive zu erzeugen [Nay97].

Verwendet man die in Kapitel 3.2.1.1 beschriebene Technik zum Erzeugen eines Panoramas, dreht aber die Kamera nicht im optischen Zentrum sondern nimmt die Fotos von verschiedenen Punkten aus auf, kommt es zu einem Parallaxefehler. Unter der Parallaxe [Jac04] versteht

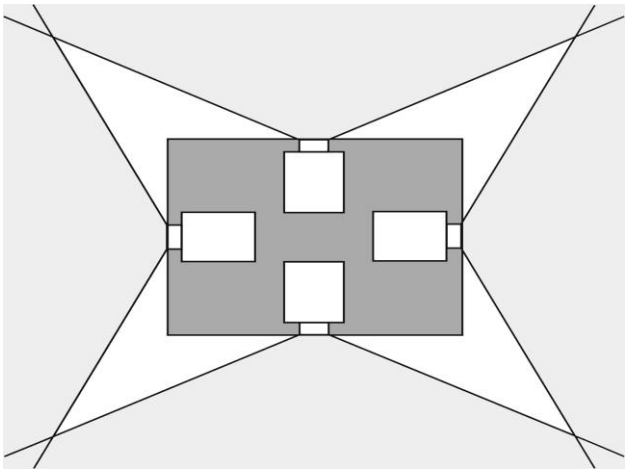

Abbildung 3-4 - Roboter (dunkelgrau) mit 4 Kameras.

man den Winkel, den zwei Gerade bilden, die von unterschiedlichen Standorten auf einen Punkt gerichtet sind. Anders formuliert, es wird damit der Effekt beschrieben, dass sich Objekte scheinbar für einen Beobachter relativ zueinander verschieben, wenn dieser sich bewegt. Dieser Effekt ist umso stärker, je näher sich der Beobachter bei einem der Objekte befindet.

Wurden nun die Bilder, die zu einem Panoramabild zusammengefügt werden sollen, von verschiedenen Punkten aufgenommen, kann es, je nach Nähe der aufgenommenen Motive und auch abhängig vom Abstand der Punkte, von wo die Fotos gemacht wurden, zu sichtbaren Fehlern kommen. Umgekehrt heißt das, dass wenn der Abstand zum Motiv groß genug ist, und die Punkte nahe genug beisammen sind, der Fehler vernachlässigbar ist. Weiters ist es durch die digitale Bildverarbeitung auch möglich, diese Fehler bis zu einem gewissen Grad zu retuschieren. Praktisch eingesetzt wird diese Technik bei Amateur-Digitalkameras, wo so relativ einfach, durch sequentielles Abfotografieren der Landschaft, Panoramafotos erzeugt werden können. Allerdings sind sie, wie das System mit der rotierenden Kamera, nicht dafür geeignet um bewegte Szenen zu erfassen.

Dieses Problem kann man allerdings lösen, indem man mehrere Kameras verwendet. Es gibt einige Projekte die diesen Ansatz verwenden. Microsoft Research entwickelte zum Beispiel die Ringcam [Cut02]. Diese 360° Panorama Kamera besteht aus sechs 60° Kameras, deren Bilder zu einem Panoramabild zusammengefügt werden. Ein ähnliches System findet man auch in [Foo00] und [Kim01].

Die Verwendung eines solchen Systems hätte einen Vorteil im Vergleich zu den bisher vorgestellten Systemen, und zwar, dass es recht einfach vollständig in den Roboter integriert werden könnte. Durch den Einsatz von vier Kameras, mit einem Blickbereich von jeweils 90° oder mehr, die jeweils an einer Seite des Roboters angebracht sind (siehe Abbildung 3-4), ist es

möglich, beinahe das komplette Netz zu beobachten. Nur sehr kleine Bereiche des Netzes, die sich unmittelbar in der Nähe des Roboters befinden, lassen sich so nicht beobachten, sofern der Sichtbereich der Kameras weniger als 180° beträgt. Da sich das Netz recht nahe an den Kameras befinden würde, wäre es aufgrund des Parallaxefehlers kaum möglich im Nachhinein ein Panorama ohne Fehler an den Übergangsstellen zwischen den Bildern der einzelnen Kameras zu erzeugen. Ein weiterer Punkt, der bei diesem System zu beachten wäre, ist, dass vier Videosignale im Roboter erzeugt werden würden. Folglich wäre der Ausfall einer Kamera nicht so katastrophal wie bei den Systemen mit einem optischen Zentrum, wo der Ausfall der einzigen Kamera einem Totalausfall gleich kommt. Allerdings, muss natürlich bei so einem System auch bedacht werden, dass entweder, im Gegensatz zu Systemen mit nur einer Kamera, mehrere Signale vom Roboter zur Rakete gesendet werden müssen oder dass schon am Roboter die Signale zu einem zusammen gefasst werden müssen. Auf jeden Fall wird der Vorteil der Redundanz durch zusätzliche Komplexität des Gesamtsystems erkauft.

Trotz der angeführten Nachteile wurde entschieden, dieses System zu verwenden. Der Grund für diese Entscheidung ist vor allem, dass es sich als einziges System relativ einfach in den Roboter integrieren lässt. Weiters stellt der Punkt, dass sich wahrscheinlich kein Panorama aus den einzelnen Bildern zusammen stellen lässt, kein sonderlich großes Problem da, denn um das Netz zu beobachtet ist es nicht unbedingt nötig, dass das komplette Netz auf einem Bild abgebildet ist.

Wie bereits erwähnt, soll das Gesamtsystem für zwei Roboter ausgelegt sein. Da allerdings nur ein Signal von der Rakete zur Erde gesendet werden kann, muss zwangsweise eine Komponente verwendet werden, die mehrere Signale zu einem vereint. Es stellt sich nun die Frage, wo die Signale zusammengefasst werden sollen. Zum einen besteht die Möglichkeit, dass jeder Roboter alle vier Signale zur Rakete sendet und dort alle acht Signale zu einem zusammengefasst werden. Allerdings müsste jeder Roboter mit vier Sendern ausgestattet werden und die Rakete mit acht Empfänger. Zum anderen könnte eine ähnliche Komponente im Roboter eingesetzt werden, um bereits dort die vier Signale zu einem einzigen zusammenzufassen. Diese Möglichkeit wurde gewählt, da wesentlich weniger Sende und Empfangsmodule benötigt werden.

3.3 System zum Zusammenfassen der Videosignale

Unabhängig davon ob ein oder zwei Roboter eingesetzt werden, ist es nötig, die Videosignale der einzelnen Kameras zusammen zu fassen. Eine ähnliche Aufgabenstellung gibt es im Überwachungsbereich. Dort ist es oftmals nötig, dass die Bilder von mehreren Kameras auf einem Bildschirm dargestellt werden. Um dies zu erreichen, werden häufig Bildteiler sowie automatische Umschalter verwendet oder Kombinationen aus diesen beiden Techniken.

Bildteiler sind Geräte, die die Bilder von mehreren Quellen zu einem vereinen. Quadrantenteiler erzeugen zum Beispiel ein Videosignal, wo die Bildfläche in vier gleiche Bereiche aufgeteilt ist und in jedem dieser Bereiche ein Video gezeigt wird. So ein System wurde beim Furushiki1 Projekt verwendet. Aber es gibt auch eine Reihe von anderen Möglichkeiten, wie Bildteiler mehrere Signale auf einmal darstellen. Gemeinsam ist, dass im Grunde die Auflösung herabgesetzt wird. Bei einem Quadrantenteiler werden die vier Bilder jeweils mit nur einem Viertel ihrer ursprünglichen Auflösung angezeigt.

Kommerziell erhältlich ist eine Vielzahl von verschiedenen Bildteilern. Allerdings war es trotz intensiver Suche nicht möglich einen Quadrantenteiler zu finden, der klein genug war, um ihn im Roboter selbst zu verwenden. Der kleinste Quadrantenteiler der gefunden wurde ist der von der Firma Ovation Systems angebotene FourSight [Ova], der allerdings auf Grund seiner Maße von 160 x 90 x 25 mm und einem Gewicht von 440 Gramm nur in der Rakete verwendet werden könnte.

Systeme mit automatischem Umschalten arbeiten nach dem Time-Division-Multiplexing Prinzip. Es wird das Videosignal einer Kamera ausgegeben und nach einer gewissen Zeit zu der nächsten Kamera weitergeschalten. Dieses System ist so nicht direkt für das Furoschiki2 Projekt verwendbar, da immer ein Teil des Netzes für längere Zeit unbeobachtet wäre. Schaltet man allerdings mit einer höheren Frequenz um, lässt sich das Videosignal zwar nicht mehr direkt betrachten, allerdings bleibt auch kein Teil des Netzes für längere Zeit unbeobachtet. Angenommen ein Videosignal hat eine Bildübertragungsrate von 25 Bildern in der Sekunde. Von vier Kameras werden nun vier solcher Signale geliefert. Da nur eine Übertragungsleitung zur Verfügung steht, wird nun 25 Mal in der Sekunde zwischen diesen Videosignalen hin- und hergeschalten. Trennt man nun, nachdem das so entstandene Signal übertragen wurde, die einzelnen Signalteile wieder heraus, erhält man von jeder Kamera 6.25 Einzelbilder in der Sekunde. Bei 8 Kameras sinkt die Bildanzahl entsprechend auf 3.125 Einzelbilder pro Kamera pro Sekunde. Wiederum von Ovation Systems gibt es ein Bildübertragungssystem welches genau

nach diesem Prinzip arbeitet. Allerdings ist das System mit einer Größe von 160 x 90 x 25 mm und 450 Gramm nicht direkt am Roboter einsetzbar.

Es ist also nötig eine Komponente, die im Weiteren als VideoController bezeichnet wird, zu entwickeln, die mehr Videos zusammen fasst. Bei der Wahl, ob ein System mit einer hohen Bildfrequenz und einer geringen Auflösung, wie das bei Bildteilern der Fall ist, oder ein System mit geringer Bildfrequenz und hoher Auflösung entwickelt werden soll, fiel die Entscheidung auf Zweiteres, also auf die Entwicklung eines Systems, welches auf dem Prinzip des Time-Division-Multiplexings beruht. Die Gründe dafür sind die, dass die Bildfrequenz zum Beobachten des Netzes ausreicht und, aufgrund der höheren Auflösung, eventuell mehr Details erkennbar sind.

Nun wären einige Systeme vorstellbar, mit denen sich die Signale multiplexen ließen. Die einfachste Möglichkeit wäre, regelmäßig zwischen den einzelnen Signalen umzuschalten. Geht man davon aus, dass die einzelnen Signale der Kameras ein Bild nach dem anderen übertragen, wie das vereinfacht gesagt beim Fernsehsignal (vgl. Kapitel 4) der Fall ist, müsste man zumindest solange warten, wie zwei Bilder für die Übertragung brauchen. Der Grund dafür ist der, dass im Allgemeinen die Kameras nicht synchron laufen und es daher sein kann, dass gerade dann auf ein Signal geschalten wird, nachdem mit diesem der Anfang eines Bildes übertragen wurde. Damit ein vollständiges Bild übertragen wird muss nun solange gewartet werden, bis das restliche Bild übertragen wurde sowie das nächste vollständige. Das Problem das sich nun ergibt ist, neben der Tatsache, dass so das Signal regelmäßig sozusagen schadhaft ist, dass nur ungefähr 50% der möglichen Übertragungskapazität ausgeschöpft wird. Eine leichte Verbesserung wäre nun der Einsatz von sogenannten „sync strippers“ mit deren Hilfe man den Anfang und das Ende eines Bildes in einem Videosignal herauslesen kann. Trotzdem wäre die Ausnützung der Übertragungskapazität nicht optimal. Nun gibt es zwar Kameras, die sich synchronisieren lassen, wodurch ein Multiplexen zwischen den einzelnen Kameras durch einfaches Umschalten möglich wäre und gleichzeitig der Übertragungskanal optimal ausgenutzt wird, allerdings wäre dadurch die Auswahl an Kameras stark eingeschränkt, sowie eine Verwendung von einem solchen System auf der Rakete nur schwer möglich.

Ein anderer Ansatz zum Multiplexen der Signale wäre der Einsatz eines digitalen Systems, wo die einzelnen Bilder der Signale gepuffert werden können, und aus ihnen das neue Ausgangssignal generiert werden kann. Dieser Ansatz hat mehrere Vorteile. Man ist etwa bei der Auswahl der Kameras nicht so eingeschränkt, wie wenn man nur synchronisierbare Kameras verwenden würde. Durch den Einsatz von AD-Wandlern für Videosignale wie sie zum Beispiel von

den Firma Analog Devices Inc. angeboten werden, ist so ein System nicht auf die Verwendung von digitalen Kameras beschränkt. Des Weiteren lässt sich so ein System mit nur geringen Modifizierungen auch für das Multiplexen der Videosignale auf der Rakete verwenden. Und schließlich können durch geringe Modifikationen am System auch Kameras verwendet werden, die mit geringeren Frameraten arbeiten als dies normalerweise der Fall ist. Ein Beispiel für so eine Kamera wäre die extrem lichtempfindliche 21K13X der Firma Videology Incorporation. Aufgrund all dieser Vorteile wurde beschlossen, diesen Ansatz weiter zu verfolgen.

Der Vollständigkeit halber sei noch erwähnt, dass es prinzipiell auch die Möglichkeit gebe, für den Roboter eine spezielle Kamera mit vier Bildsensoren zu entwickeln. Also dass mit Hilfe eines FPGAs, an dem zum Beispiel vier CCD-Bildsensoren oder CMOS-Module angeschlossen sind, ein Fernsehsignal erzeugt wird, wobei die Bilder von jenen 4 Sensoren stammen. Allerdings könnte dieses System nur in den Robotern verwendet werden und für die Rakete müsste ein eigenes System entwickelt werden wodurch sich im Endeffekt der Entwicklungsaufwand im Gesamten verdoppeln würde.

3.4 Aufbau des Gesamtsystems

Das gesamte System ist nun folgender Maßen aufgebaut: In jedem Roboter befinden sich vier Kameras sowie ein VideoController, der die Signale mittels Time-Division-Multiplexing zusammenfasst. Anschließend werden die Signale mit Hilfe eines Sendemoduls zu Rakete übertragen, wo wiederum die Signale von den verschiedenen Robotern zusammengefasst werden. Die Rakete sendet das Signal zur Erde, wo dieses empfangen wird. Das Signal wird nun sowohl aufgezeichnet als auch direkt in einen Computer eingespielt. Dort wird es aufgetrennt, so dass jedes der acht Videosignale betrachtet werden kann.

Als Kamera ist grundsätzlich jede Kamera geeignet, die ein Signal liefert, das der CCIR656 Norm entspricht, oder entsprechend transformiert werden kann. Zum Beispiel würde mit Hilfe des ADV7180 der Firma Analog Devices Inc. jede Kamera verwendet werden können, deren Ausgangssignal vom Typ FBAS, Y/C oder YPbPr ist. Aufgrund des beschränken Platzes wird es sich höchst wahrscheinlich um eine Platinenkamera handeln, da diese wegen ihrer Größe am besten für den Einbau in den Roboter geeignet sind.

Die Videosignale werden nun mit Hilfe des im Rahmen dieser Diplomarbeit entwickelten VideoControllers zusammengefasst. Dieser Controller wird mit Hilfe eines FPGA realisiert, der zur Pufferung der digitalen Videosignale noch mit einem SDR SDRAM verbunden ist. Eine mögliche

Konfiguration wäre zum Beispiel ein Virtex4 vom Typ XC4VSX25 der Firma Xilinx [Xil07a] gekoppelt mit einem K4S283233F-MN1L 4Mx32 SDRAM [Sam02].

Für die Übertragung des Videosignals von den Robotern zur Rakete bieten sich Sendemodule an, wie sie vor allem in der Videoüberwachung verwendet werden. Diese zeichnen sich vor allem durch ihre kompakte Bauform aus. Gegebenenfalls muss dazu das digitale Videosignal, welches vom FPGA ausgegeben wird, wieder in ein analoges gewandelt werden. Auch dafür gibt es eine Vielzahl verschiedener Komponenten am Markt.

Auf der Rakete selbst wird nun das Signal wiederum von einem kommerziell erhältlichen Empfangsmodul empfangen, gegebenenfalls digitalisiert und wiederum mittels eines FPGAs, der ebenfalls an einem SDRAM angeschlossen ist, verarbeitet. Schließlich wird das Signal an die vom Raketenhersteller zur Verfügung gestellten Sendeeinrichtung übergeben und zur Erde gesendet, wo es, wie bereits beschrieben, aufgezeichnet wird.

Um die Bilder der einzelnen Kameras nun separat betrachten zu können, muss nun der Videostream wieder aufgetrennt werden. Die günstigste Lösung ist hierbei, das Video mittels einer Framgrabberkarte in einen PC einzuspielen und dort mit Hilfe einer entsprechenden Software zu trennen. Eine entsprechende Software wurde entwickelt und wird in Kapitel 6 beschrieben. Mit dieser Software ist es nun auch möglich das Geschehnis live zu betrachten, sofern das von der Bodenstation empfangene Signal sofort zur Verfügung steht.

4 Video Grundlagen

In diesem Kapitel wird eine kurze Einführung in die für diese Diplomarbeit relevanten Bereiche der Videotechnik gegeben. Nach einer allgemeinen Einführung wird speziell auf die CCIR Norm 656 eingegangen.

4.1 FBAS

Ein Video besteht aus einer Reihe von Bildern, die in rascher Folge angezeigt werden und so beim Betrachter die Illusion einer fließenden Bewegung erzeugen. Beim analogen Farbfernsehen wird nun das Farb-, Bild-, Austast-, Synchronsignal, kurz FBAS-Signal, verwendet, welches schließlich noch zusammen mit einem Tonsignal für die Übertragung auf eine Trägerfrequenz modelliert wird. Neben der Bezeichnung FBAS sind auch die englischen Bezeichnung CVBS, für Colour, Video, Blanking und Sync, sowie Composite Video gebräuchlich [Schmi03].

Das FBAS Signal ist kompatibel mit dem „schwarzweiß Fernsehsignal“, dem BAS-Signal, wobei BAS wiederum für Bild, Austast und Synchron steht. Der Unterschied zwischen BAS und FBAS liegt vor allem darin, dass beim FBAS-Signal Farbinformationen hinein modelliert werden. Für beide Signale gilt, dass die einzelnen Bilder hintereinander übertragen werden. Je nach Fernsehnorm unterscheiden sich hierbei die Anzahl der Bilder die pro Sekunde übertragen werden sowie ihre Größe. So wird in Europa meist die PAL-50 Norm verwendet, die eine Bildfrequenz von 50Hz festlegt sowie eine Zeilenanzahl von 625 Zeilen pro Bild. Die in den USA gebräuchliche NTSC-Norm legt 525 Zeilen pro Bild fest wobei 29,97 Vollbildern pro Sekunde beim Farbfernsehen und 30 Vollbilder pro Sekunde beim Schwarzweißfernsehen übertragen werden. Dies entspricht wiederum einer Bildfrequenz von 59,94 beziehungsweise 60Hz pro Sekunde. Die Bildfrequenzen sind bewusst etwa gleich gewählt wie die Frequenzen im jeweiligen Stromnetz, um Interferenzen zu vermeiden. Wichtig in diesem Zusammenhang ist, dass die einzelnen Bilder (Frames) als jeweils zwei Halbbilder (Field oder auch Videofeld) übertragen werden, wodurch sich die doppelt so hohe Bildfrequenz im Bezug auf die Anzahl der Vollbilder erklären lässt. Der Grund für die Übertragung in Form von Halbbildern ist der, dass so das Bild weniger flimmert. Das erste Halbbild enthält alle ungeraden und das zweiter Halbbild alle graden Zeilen des Bildes.

Aufgrund der Fernsehtechnik in den 50er Jahren musste nach jedem Halbbild eine gewisse Zeit eingeplant werden, damit der Strahl des Röhrenfernsehers an den Bildanfang zurückkehren, er also vom unteren zum oberen Rand zurückspringen kann. Dafür wurde eine Vertikalaustastlücke von 1,6 ms reserviert. Bei 50 Halbbildern pro Sekunde beträgt die Dauer eines Halbbildes 20ms. Von diesen 20ms werden 1.6ms für den Rücksprung reserviert was 25 Zeilen entspricht.

Damit reduziert sich die nutzbare Zeilenzahl im Pal-50 System auf 575 Zeilen. Ebenso existiert nach jeder Bildzeile eine Horizontalaustastlücke von 12us, für die Positionierung des Strahles am Anfang der nächsten Zeile. Der Name Austastlücke kommt daher, da bei jedem dieser Sprünge der Strahl ausgeschalten sein muss, was auch als „ausgetastet“ bezeichnet wird.

In den Austastungslücken eingebettet befinden sich nun Synchronisationssignale. In jeder Horizontalaustastungslücke befindet sich ein H-Synchronimpuls mit einer Dauer von 4,7us. Ebenso gibt es einen V-Synchronimpuls für die vertikale Synchronisation, der aus fünf 27,3us langen Impulsen besteht. Weiters kommen beim V-Synchronimpuls noch fünf Vor- und Nachtrabanten mit einer Dauer von 2,35us hinzu, die für gewisse Synchronisationsschaltungen gebraucht werden. Da diese Impulse im BAS-Signal in einem anderen Spannungsbereich als das Bildsignal liegen, können sie recht einfach von diesem wieder getrennt werden.

In jeder der verwendbaren Bildzeilen wird nun eine Zeile des aufgenommen Bildes übertragen. Dabei werden die einzelnen Bildpunkte der Reihe nach abgetastet wobei ursprünglich beim Schwarzweißfernsehern von jedem Punkt die Helligkeit gemessen wird, die dann in das Bildsignal umgesetzt wird. Dabei liegt der Spannungspegel für Schwarz meist etwas über 0 Volt und der für Weiß meist bei 0,7 Volt. Beim Farbfernsehen werden drei Farbwertsignale gebildet, aus denen dann ein Chrominanz- sowie zwei Farbdifferenzsignale erzeugt werden. Aus den Farbdifferenzsignalen wird das Luminanzsignal berechnet, wobei sich die Berechnung je nach verwendetem System unterscheidet. Dieses Signal wird mit dem Chrominanzsignal zum Bildsignal zusammen geführt.

In Abbildung 4-1 ist nun eine Bildzeile als BAS-Signal dargestellt. Gut zu erkennen sind die Synchronisationsimpulse im negativen Spannungsbereich.

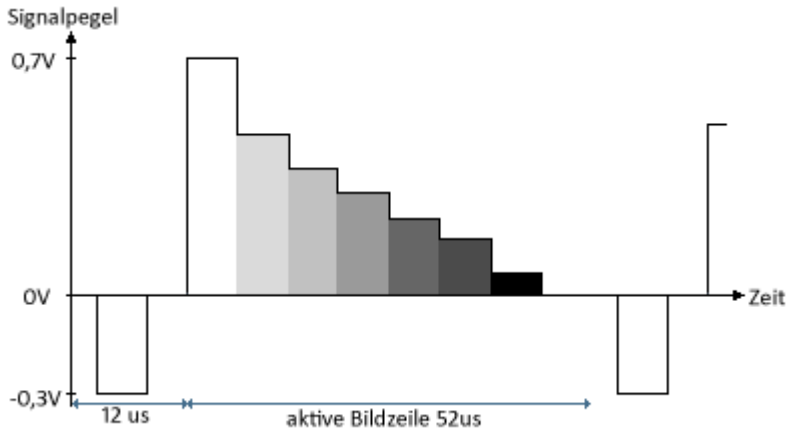

Abbildung 4-1 - BAS- Signal

4.2 CCIR 601 / 656

Die CCIR601 Norm [ITU95] beschreibt die digitale Codierung von interlaced-Video-Signalen wie etwa dem oben beschriebenen FBAS-Signal. Sie wurde ursprünglich vom Comité Consultatif International des Radiocommunication (CCIR) herausgegeben, die inzwischen in der ITU, der International Telecommunication Union, aufgegangen ist. Entsprechend besitzt die ehemalige CCIR 601 Norm inzwischen den Namen ITU-R BT.601. Unterstützt werden sowohl 525-Zeilensysteme als auch welche mit 625 Zeilen. Die Formate 4:3 sowie 16:9 werden mit einer Sampling-Rate von 13,5 MHz unterstützt, wobei es für die 16:9 Systeme auch eine Version mit 18 MHZ gibt. Wichtig in dem Zusammenhang ist, dass ein Digital Component Signal erzeugt wird. Das heißt, dass dieses Signal aus drei Komponenten besteht und zwar aus einer Helligkeitskomponente Y sowie zwei Farbdifferenzkomponenten Cr und Cb. In der ursprünglichen Version standen zur Codierung der Werte 8 Bit zur Verfügung, allerdings ist später auch eine 10 Bit Variante hinzugekommen. Im 8-Bit System wird bei der Quantisierung dem Helligkeits-Signal ein Wert zwischen 16 und 235 zugewiesen. Die zwei Farbdifferenzkomponenten können einen von 225 Werten annehmen, wobei der Wert 128 Null entspricht. Die Werte 0 sowie 255 sind bei allen drei Komponenten für Synchronisationsdaten reserviert, die beispielsweise in der CCIR656-Norm eingeführt werden.

Wie bereits erwähnt wird eine Sampling-Rate von 13,5 beziehungsweise 18MHz verwendet. Dies trifft allerdings nur für die Helligkeitskomponente zu. Bei der Abtastung der Farbdifferenzkomponenten muss man zwischen dem 4:4:4 System und dem 4:2:2 System unterscheiden. Beim 4:4:4-System werden die Farbdifferenzkomponenten ebenfalls mit derselben Frequenz wie die Helligkeitskomponente abgetastet. Beim 4:2:2-System, auf dem die CCIR656 Norm aufbaut, wird nur die halbe Taktrate verwendet. Die Ausgabe des 4:2:2 System ist ein Daten Strom mit 27MBit pro Sekunde, wenn 8 Bit zur Codierung verwendet werden sowie eine Abtastrate von 13,5MHz. Dabei werden die drei Komponenten gemultiplext, so dass das Signal wie folgt aufgebaut ist: Y Cr Y Cb Y Cr Y Cb

Auf dem 4:2:2-System aufbauend ist nun die CCIR656 Norm [CCIR86] definiert, die inzwischen die Bezeichnung ITU-R BT.656 trägt. Sie definiert ein Signalformat, das im Grunde dem 4:2:2 Format entspricht, welches allerdings mit einigen Codes, allen voran Timing-Codes, ergänzt wurde. Weiters wird ein bit-paralleles Interface sowie ein bit-serielles Interface spezifiziert.

Als Timing Codes wird nun jeweils am Ende des Austastungsintervalls, also kurz vor Anfang einer aktiven Zeile ein SAV Wort (Start of Active Video) eingefügt, sowie nach der aktiven Zeile ein EAV Wort (End of Active Video). Jedes dieser Wörter hat als Hexadezimal-Wert geschrieben

das Format FF 00 00 XY wobei FF 00 00 eine fixe Präambel ist. Im Byte XY sind nun drei Bit codiert:

- F: Gibt an ob es sich beim aktuellen Halbbild um das Erste oder um das Zweite handelt.
- V: Gibt an ob die aktuelle Zeile Teil einer Vertikalaustastlücke ist.
- H: ob es sich um ein SAV oder um ein EAV Wort handelt.

Vier weitere Bits des XY Bytes bilden eine Prüfsumme, so dass es möglich ist, Einzelbitfehler zu korrigieren. Ein Bit ist weiters fix auf 1 gesetzt. Durch Analyse der Codewörter lässt sich leicht erkennen, welches Halbbild gerade übertragen wird oder ob gerade eine Vertikalaustastlücke zwischen einem zweiten und einem ersten Halbbild übertragen wird. Eine solche Stelle ist deswegen interessant, da sich das Video an dieser schneiden lässt.

Die verschiedenen physikalischen Charakteristika für die Interfaces, die in dieser Norm definiert werden, spielen für diese Diplomarbeit keine Rolle. Was hingegen übernommen wurde war das Daten-Signal Format sowie die Takt-Charakteristika des bitparallelen Interfaces. Die Daten werden mittels eines 8-Bit breiten Busses übertragen und sind im NRZ-Format codiert. Der Takt des Videosignals wird durch eine separate Taktleitung übertragen.

5 Videocontroller

Wie bereits aus Kapitel 3 hervorgegangen ist, ist der Videocontroller, neben der Software, eine der im Rahmen dieser Diplomarbeit zu entwickelnden Komponenten. Der Videocontroller soll in der Lage sein, vier beziehungsweise zwei CCIR656-Signale zu einem zusammenfassen, wobei bei der Entwicklung natürlich auch das Einsatzgebiet nicht außer acht gelassen werden darf. In diesem Kapitel wird nun das Resultat dieser Entwicklungsarbeit genauer betrachtet.

Im Rahmen dieser Diplomarbeit wurden noch eine Reihe weiterer Module als die, die im Weiteren vorgestellt werden, entwickelt. Beispielweise wurden Module entwickelt um Kameras zu simulieren, oder auch komplette Designs um den Datenstrom einer Kamera auszulesen, auf den RAM zu speichern und dann per RS232 Schnittstelle auf den PC zu übertragen. Da diese allerdings nicht im Videosystem verwendet werden, wird auf sie nicht weiter eingegangen.

5.1 Plattform

Grundsätzlich ist einmal zu klären, auf Basis welcher Plattform das System entwickelt werden soll. Dabei gibt es die Möglichkeiten das System in Hardware oder in Software zu implementieren wobei auch ein Mischsystem denkbar wäre. Eine Umsetzung in Software würde bedeuten, dass ein Mikrocontroller verwendet wird, um die Signale zu multiplexen. Da allerdings von jeder Kamera 27 MegaByte Daten pro Sekunde geliefert werden, also 864 MegaBit pro Sekunde von dem Videocontroller verarbeitet werden müssen, wäre eine Umsetzung auf Basis eines Mikrocontrollers, wo es nicht wie bei einem Hardwaredesign möglich ist, verschiedene Funktionen zu parallelisieren, recht kompliziert. Daher wurde beschlossen, den Videocontroller in Hardware zu realisieren.

Als Sprache wurde die Very High Speed Integrated Circuit Hardware Description Language, kurz VHDL, gewählt. Diese an Ada angelegte Entwurfssprache [Schil03] ist besonders im europäischen Raum verbreitet.

Um die Lesbarkeit des Codes zu erhöhen wurde außerdem beschlossen, den in [Gas02] vorgeschlagenen Design-Methoden weitgehend zu folgen. So bestehen die meisten Entities nur aus zwei Prozessen, einem kombinatorischen und einem sequentiellen. Ebenso werden häufig record types verwendet sowie Zustandsmaschinen. Auch sonst wurde bei der Entwicklung vor allem auf die Lesbarkeit geachtet und daher versucht, komplexe Funktionen möglichst einfach zu gestellten, auch wenn dadurch manchmal der Platz auf dem FPGA nicht optimal ausgenutzt wurde.

5.2 Entwicklungsumgebung

Für die Implementierung des Designs wurde das ISE-Webpack [Xil07b] von Xilinx in der Version 9.1i verwendet. Dieses kostenlose Tool vereint Synthesewerkzeug, Werkzeuge für den Schaltungsentwurf sowie Projektverwaltung und unterstützt eine Reihe von FPGAs der Firma Xilinx. Für die Simulation wurde die ebenfalls kostenlose Xilinx Version des ModelSim XE III/Starter 6.1e [Xil07c] verwendet.

Weiters wurde für verschiedene Tests die HydraXC-50 (Abbildung 5-1) der Firma Eubus GmbH [Eub06] verwendet. Diese nur 55 mm x 24 mm große Hardwareplattform besitzt neben einem Virtex-4 XC4VFX12-SF363 als Kernstück unter anderem noch folgende Komponenten:

- 4Mx32 SDRAM
- 32Mbit serial Flash
- 10/100 Base-T Ethernet interface
- USB 2.0 HS/OTG Interface
- Mini-SD Card socket
- Real Time Clock
- 31 Pin TFT/User Connector
- Touch screen digitizer
- RS232 interface
- System Management Processor

Durch die geringe Größe, gekoppelt mit einer hervorragenden Leistung, wäre es auch denkbar, diese Plattform direkt im Roboter zu verwenden und so wiederum Entwicklungszeit einzusparen. Da diese Platine bereits am Institut vorhanden ist und auch der RAM ausreichend Speicherkapazität für die zu entwickelnde Anwendung besitzt, wurde sie als Entwicklungsplattform gewählt.

Die HydraXC besitzt eine kleine Besonderheit, und zwar ist ein 12 MHz Oszillator an den FPGA angeschlossen, allerdings an einem Pin, der nicht als Eingangspin für das Clock-Signal gedacht ist. Die ISE lässt kein Design zu, an dem der Clock nicht an einem Clock-Pin angeschlossen ist. Es wird daher eine Fehlermeldung ausgegeben und das Design lässt sich in Folge nicht kompilieren. Durch Setzen der Umgebungsvariablen XIL_PLACE_ALLOW_LOCAL_BUFG_ROUTING auf 1 lässt sich dieser Fehler auf eine Warnung zurückstufen, wodurch dieses Problem umgangen werden kann.

Abbildung 5-1 - HydraXC

Da in vielen Fällen andere Frequenzen als die Eingangsfrequenz verlangt werden, besitzt der Virtex 4 vier DCMs [Xil07e]. Diese DCMs können mit Hilfe eines digitalen Frequenzsynthesizers (DFS) verschiedene Takte generieren und durch eine Delay-Locked Loop die Phasenlage ändern. Die Delay-Locked Loop ist erst ab einer Frequenz von 32MHz verfügbar [Xil07d]. Daher können die DCMs, die mit 12MHZ angesteuert werden, nur im reinen DFS-Modus arbeiten. Allerdings kommt es beim Virtex 4 zu Problemen, wenn nur der DFS-Modus verwendet wird und zwar wird das „Lock"-Signal, welches normalerweise ausgegeben werden sollte wenn sich der DCM stabilisiert hat, nicht ausgegeben. Je nach Model gibt es nun dafür verschiedene Lösungsansätze die in [Xil07f] beschrieben werden. Daher ist es nun nötig das Design auch innerhalb der Virtex 4 Serie an das jeweilige Modell anzupassen.

Weiters wurde eine Kamera verwendet, die ein CCIR 656 konformes Signal liefern konnte. Konkret handelte es sich um eine CMUCam [RI07] die ebenfalls, wie die HydraXC, bereits vorhanden war. Dieses Modul besteht aus einem SX28 Mikrocontroller an dem eine OV6620 Omnivision CMOS Kamera angeschlossen ist. Zur Kommunikation besitzt die CMUCam unter anderem eine RS-232 Schnittstelle, wodurch es möglich ist, die Kamera über ein Terminal zu konfigurieren. Es wurde eine Weiche zwischen dem Omniview-Modul und dem Mikrocontrollerboard eingebaut, um das Videosignal abzunehmen. (Siehe Abbildung 5-2) In der CMUCam wird das Omniview Modul mit 5 Volt betrieben, wodurch es noch nötig war, das Signal mittels eines Pegelwandlers (74LVX3245WM) auf 3.3 Volt zu reduzieren, da der FPGA mit dieser Spannung arbeitet. Der Schaltplan ist im Anhang zu finden.

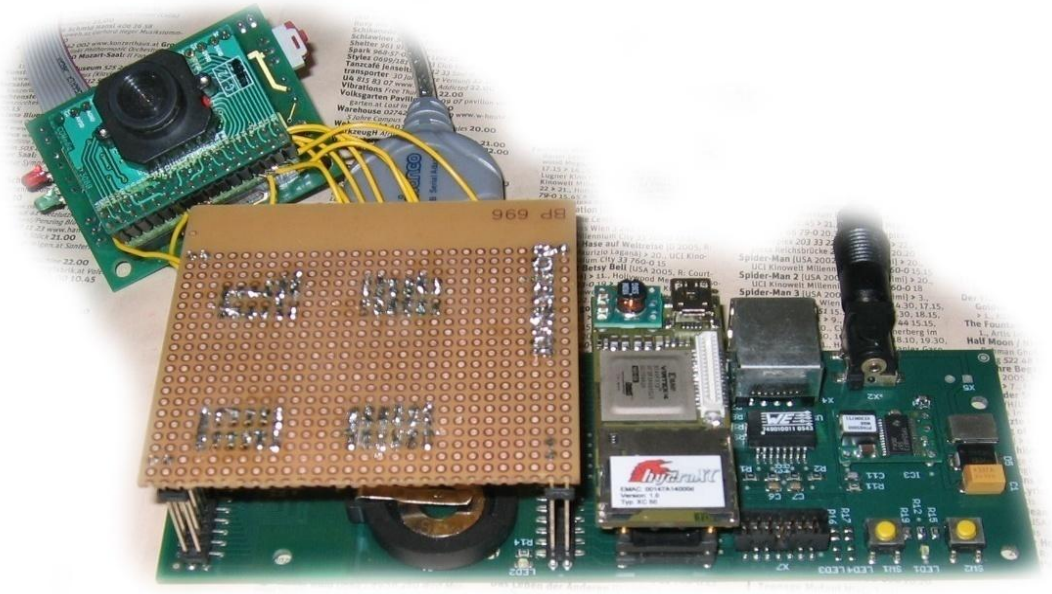

Abbildung 5-2 - HydraXC auf Entwicklungsboard mit Pegelwandler und CMUCam.

5.3 Controller des Roboters

Im nachfolgenden Teil wird nun der Aufbau des Videocontrollers des Roboters genauer beschrieben. Er soll vier Kamerasignale zu einem multiplexen. Es werden die einzelnen Module aus denen der Controller besteht vorgestellt, ihre Funktionsweise erklärt, sowie ihre Schnittstellen beschrieben.

5.3.1 Allgemeine Funktionsweise

Die Funktionsweise des gesamten Controllers ist relativ simpel. Der Speicher im RAM wird in vier Teile geteilt und jeder Teil einer Kamera zugewiesen. In jedem dieser Speicherbereiche wird von der jeweiligen Kamera ein komplettes Bild abgelegt. Anschließend beginnt eine Funktion ein Bild aus einem dieser Speicherbereiche auszulesen und gibt es als CCIR656 konformes Bildsignal aus. Sobald das Bild zu Ende ist, wird das nächste Bild aus einem anderen Speicherbereich ausgelesen und gleichzeitig beginnt eine andere Funktion, den leeren Speicher mit einem neuen Bild zu füllen. Dafür hat diese Funktion solange Zeit, bis die Bilder aus den drei anderen Speicherbereichen ausgelesen wurden.

Wie bereits in Kapitel 4 beschrieben, enthält das digitale Videosignal der Kamera Synchronisationssignale, wodurch unter anderem erkennbar ist, wo sich die Vertikalaustastlücke des ersten Videofeldes befindet, also wo der Bereich ist, in dem keine Daten von Halbbildern übertragen werden. In der Vertikalaustastlücke zwischen einem zweiten und ersten Halbbild zweier

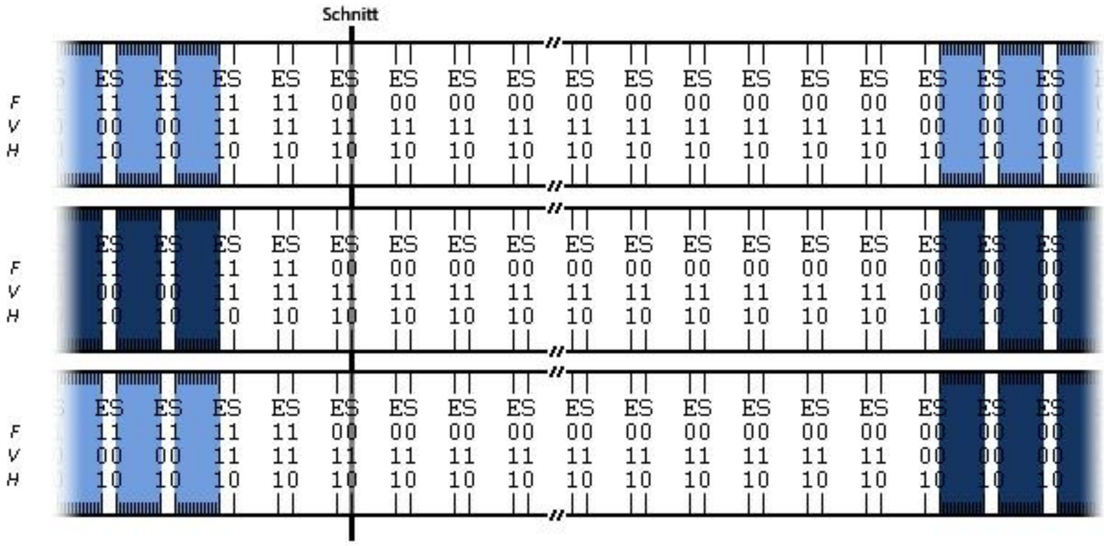

Abbildung 5-3 - Schnitt zwischen zwei Videostreams sowie generierter Streams.

verschiedener Vollbilder haben die SAV-Wörter den F-Wert 0 und den V-Wert 1. Um nun die einzelnen Bilder sauber zu trennen, bietet es sich an, beim ersten Auftreten des SAV Wortes „010“ mit dem Einlesen eines Bildes zu beginnen, beziehungsweise mit dem Lesen zu enden. Falls aus irgendwelchen Gründen das entsprechende SAV Signal nicht erkannt wird, weil es etwa zu einem Fehler in der Datenübertragung gekommen ist, ist es recht wahrscheinlich, dass zumindest eines der nächsten SAV-Wörter erkannt wird. In diesem Fehlerfall würde sich nur die Vertikalaustastlücke kurzzeitig etwas verlängern. Allerdings kommt es nicht dazu, dass gleich zwei Vollbilder eingelesen werden. Die Ausgabefunktion muss nur noch die einzelnen Bilder aneinander Reihen.

In Abbildung 5-3 sind drei digitale Videosignale dargestellt. In den farbig markierten Bereichen werden Daten transportiert, die weißen Bereiche zwischen den farbigen stellen Austastungslücken dar. Gut zu erkennen ist jeweils die Vertikalaustastlücke - der große, weiße Bereich in der Mitte. Die ersten zwei Videosignale stellen Signale von zwei verschiedenen Kameras da, das dritte Signal wäre das, was vom Videocontroller generiert werden soll. Signal Nummer 3 ist anfangs identisch mit Nummer 1. Sobald das SAV Wort „010“ erreicht wird, ist das Signal 3 nun mit dem Signal 2 identisch und entspricht weiterhin der CCIR 656 Norm. Würde nicht in diesem Intervall umgeschalten werden, würden die Einzelbilder fehlerhaft werden. Durch die Zwischenspeicherung der Bilder ist es möglich, die verschiedenen Streams zu synchronisieren.

5.3.2 Allgemeiner Aufbau

Der Videocontroller ist aus einer Reihe von Modulen aufgebaut, die miteinander über einen 256Bit breiten Bus sowie mit einigen weiteren Steuerleitungen miteinander verbunden sind. Aufgrund der Breite des Busses ist es möglich, die Kommunikation zwischen den Modulen relativ einfach zu gestallten wodurch das gesamte System einfacher zu verstehen ist. Dadurch sollte es auch einfacher sein, das Design zu erweitern sowie eventuelle Fehler zu finden.

Zeit ist im Design natürlich auch ein wichtiger Faktor. Da der Controller keine Kontrolle über den Datenstrom der von den Kameras stammt hat, ist es wichtig, dass sichergestellt ist, dass der Datenstrom innerhalb einer vorgegebenen Zeit bearbeitet wird. Ebenso muss auch für den ausgehenden Datenstrom sichergestellt sein, dass alle von der Norm angegebenen Zeiten eingehalten werden. Es handelt sich um ein Echtzeitsystem, also um ein System dessen korrektes Verhalten nicht nur durch die Richtigkeit der Ausgabe bestimmt wird, sondern auch durch die Rechtzeitigkeit [Kop97]. Auch in diesem Zusammenhang ist die Breite des Busses von Vorteil, da so in kurzer Zeit große Datenmengen zwischen den Modulen ausgetauscht werden können und gleichzeitig die Intervalle zwischen zwei Übergaben größer werden. So steht für die Verarbeitung eines Paketes, auch wenn es nun größer ist, mehr Zeit zur Verfügung. Dadurch ist es nun wiederum möglich, das System übersichtlicher zu gestalten.

Da der Videocontroller unter anderem ein Signal mit einer Taktrate von 27MHz ausgeben muss, ist es nötig, das Ausgabemodul mit 27MHz oder einer Frequenz, die durch eine gerade Zahl geteilt 27MHZ ergibt, zu betreiben. Auch von anderen Modulen sind zeitliche Vorgaben gegeben. So sollte das Schedule-Modul mit zirka 80MHz oder schneller betrieben werden (vergleiche Kapitel 5.3.5), und der RAM darf mit maximal 100MHZ angesteuert werden. Da es nun zwischen 80MHz und 100MHz keine Frequenz gibt, die durch eine gerade Zahl geteilt 27MHz ergibt, ist es nötig, dass in dem Gesamtsystem zwei verschiedene Taktdomains verwendet werden. Konkret werden die Taktraten 80MHz und 108 MHz verwendet.

Zur Generierung der Takte werden zwei DCM-Untis des Virtex verwendet um aus dem 12MHz schnellen Takt des Quarzes das entsprechende Signal zu generieren. Wie bereits erwähnt müssen hier, abhängig davon, welcher Virtex FPGA zum Einsatz kommt, Anpassungen vorgenommen werden. Auch bei der Verwendung anderer FPGAs müssen die DCMs durch entsprechende Module ersetzt werden. Wird an dem FPGA ein RAM angeschlossen, der zum Beispiel mit 120MHz angesteuert werden kann, wäre auch eine Lösung mit nur einem 108MHz Takt denkbar.

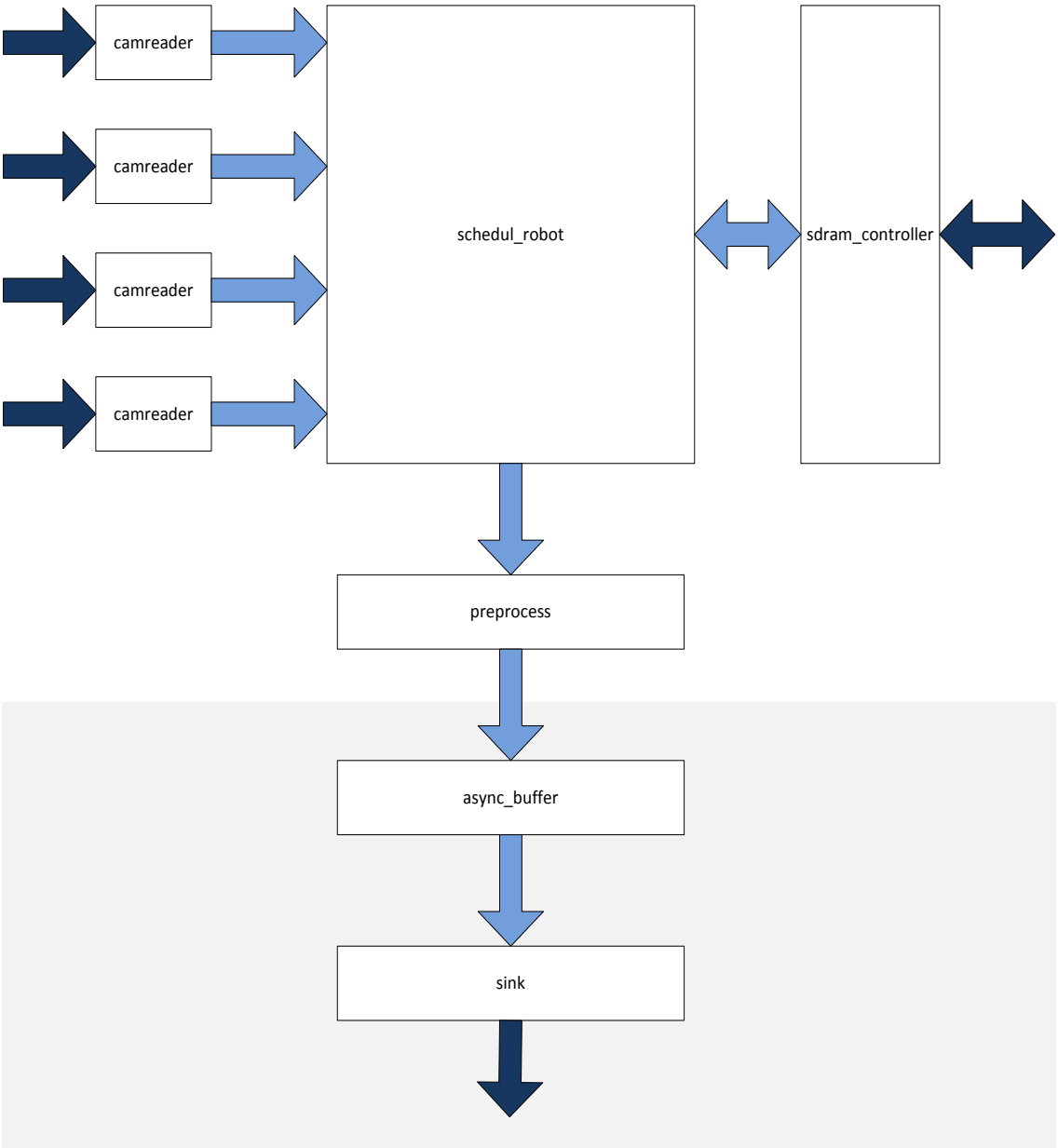

Abbildung 5-4 - Blockdiagramm des Videocontrollers des Roboters

In Abbildung 5-4 sind nun in Form eines Blockdiagramms die Zusammenhänge der einzelnen Module, mit Ausnahme des Top-Moduls, grob dargestellt. Weiters sind auch die beiden Taktdomains eingezeichnet. Die Module, die sich im weißen Bereich befinden, sind mit 80MHz getaktet, die im hellgrauen mit 108MHz. Die Pfeile zeigen die wichtigsten Datenpfade an, wobei die dunkleren Pfeile die Verbindungen zur Umgebung darstellen und die hellen die internen Pfade.

Anhand des Blockdiagramms kann man auch den Lauf der Daten durch den Videocontroller gut veranschaulichen. Die Camreader lesen die Datenströme der einzelnen Kameras ein und übergeben sie je nach Bedarf an den Schedule. Dieser übergibt sie zur Zwischenspeicherung an den

SDRAM-Controller von dem er auch Daten erhält. Dann werden die Daten an den Preprozessor übergeben, worauf sie anschließend bei der Übergabe in das „async_buffer“-Modul in die andere Taktdomäne transferiert werden. Schließlich wird der Datenstrom über die Sink wieder ausgegeben.

5.3.3 top.vhd

Das Modul „top“ repräsentiert den kompletten Videocontroller. In diesem Modul werden alle Module des Videocontrollers eingebunden und miteinander vernetzt. Welche Leitungen der einzelnen Module miteinander verbunden sind, wird bei der Schnittstellenbeschreibung der einzelnen Module genauer erklärt.

Das Modul „top“ besitzt eine Reihe von Ports mit deren Hilfe Signale aus einigen Modulen nach außengeleitet, beziehungsweise Signale von der Umgebung empfangen werden können. In Tabelle 5-1 werden die einzelnen Ports aufgelistet und beschrieben.

Tabelle 5-1 - Beschreibung der Ports des Moduls top.vhd

Name	Richtung	Breite	Beschreibung
port_clk	In	1 Bit	An diesem Port wird das Taktsignal angelegt, welches intern zu den DCM-Modulen geleitet wird.
port_in	In	1 Bit	Dieser Eingang ist im momentanen Design low-aktiv. Durch Aktivierung wird die Reset-Funktion des „top“ -Moduls ausgelöst.
port_out	Out	1 Bit	Dieser Port ist mit dem Schedule-Modul verbunden und wird auf high gesetzt wenn der Videocontroller aktiv ist. Da es bei der HydraXC Wackelkontakte gibt, ist so mit Hilfe einer LED leicht zu erkennen, ob das Design neu auf den FPGA gespielt werden musste. Dieser Port kann gegebenenfalls eingespart werden.
port_Cs_n	Out	1 Bit	Dieser Port für das „Chip-Select“-Signal muss mit dem CS Eingang des RAMs verbunden werden. Bei der HydraXC-50 wäre das der Pin J19. Intern ist er mit dem Ram_Cs_n Port des

			„sdram_controller“ –Moduls verbunden.
port_Ras_n	Out	1 Bit	Dieser Port muss mit dem RAS Eingang des RAMs verbunden werden. Bei der HydraXC-50 wäre das der Pin J18. Intern wird er mit dem Ram_Ras_n Port des „sdram_controller“ –Moduls verbunden.
port_Cas_n	Out	1 Bit	Dieser Port muss mit dem CAS Eingang des RAMs verbunden werden. Bei der HydraXC-50 wäre das der Pin K18. Intern ist er mit dem Ram_Cas_n Port des „sdram_controller“ –Moduls verbunden.
port_We_n	Out	1 Bit	Dieser Port muss mit dem WE Eingang des RAMs verbunden werden. Bei der HydraXC-50 wäre das der Pin K20. Intern ist er mit dem Ram_We_n Port des „sdram_controller“ –Moduls verbunden.
port_Cke	Out	1 Bit	Dieser Port muss mit dem CKE Eingang des RAMs verbunden werden. Bei der HydraXC-50 wäre das der Pin J16. Intern wird er mit dem Ram_Cke Port des „sdram_controller“ –Moduls verbunden.
port_Addr	Out	12 Bbit	Dieser Bus dient zur Auswahl einer Speicheradresse des RAMs und muss entsprechen an den Adressbus des RAMs angeschlossen werden. Bei der HydraXC-50 sind das mit port_Addr_0 beginnend: F19, F18, E20, E18, E16, F16, F17, G16, G17, J17, G20, H18 und H17. Intern ist der Adressbus mit dem Ram_addr Port des „sdram_controller“ –Moduls verbunden.
port_Ba	Out	2 Bit	Bus zum Auswählen der Banken des Rams und ist entsprechend am BA Port des Rams anzuschließen. Ist intern wiederrum mit dem „sdram_controller“ –Modul verbunden. Die Pins H19 und G19 sind bei der HydraXC-50 mit

			dem Ram entsprechend verbunden
port_Dqm	Out	4 Bit	Ein weiterer Steuerbus für den RAM, wiederum intern mit dem „sdram_controller“ verbunden. Er muss mit dem Dqm-Eingang des Rams verbunden werden, was auf der HydraXC-50 den Pins K19, K16, E19 und D17 entspricht.
port_Dq	Inout	32 Bit	Dieser Bus ist der einzige bidirektionale Bus des Designs. Er dient zum Datentransfer vom und zum RAM. Entsprechend muss er mit dem Dq Port des Rams verbunden werden. Bei der HydraXC-50 sind das folgende Pins: P20, N19, P19, M20, N18, M19, L20, L19, L16, M18, L17, N16, M16, P16, N17, P17, D18, D19, C19, B17, C18, A18, B19, B18, A16, B16, C20, B15, A15, C16, C17. Intern ist der Bus wiederrum mit dem „sdram_controller” verbunden.
port_ram_clk	Out	1 Bit	Das letzte Signal, das mit dem RAM verbunden werden muss, ist das Taktsignal des RAMs; Anschluss H16 auf der HydraXC-50. Intern ist es wie alle anderen RAM Signale mit dem „sdram_controller“ verbunden.
port_PxlClk	Out	1 Bit	Über port_PxlClk wird das Taktsignal des gemultiplexten Videosignals ausgegeben. Das Signal wird vom Modul „sink“ erzeugt.
port_Data	Out	8 Bit	An port_Data liegt das durch den Videocontroller generierten Videosignal im 8-bit parallelen CCIR656 Format an. Erzeugt wird es, wie das Taktsignal des Ausgabesignals, durch das „sink“ - Modul.
port_DC0_cam_clk	In	1 Bit	Eingang des Taktsignals der Kamera 0.
port_ DC0_cam_data	In	8 Bit	Datenschnittstelle zur Kamera 0. Dieses Signal wird zusammen mit dem dazugehörigen Taktsignal im „camreader“ 0 verarbeitet.
port_DC1_cam_clk	In	1 Bit	Wie port_DC0_cam_clk, nur für Kamera 1.

port_ DC1_cam_data	In	8 Bit	Wie port_DC0_cam_data, nur für Kamera 1. Verarbeitung in „camreader“ 1.
port_DC2_cam_clk	In	1 Bit	Wie port_DC0_cam_clk, nur für Kamera 2.
port_ DC2_cam_data	In	8 Bit	Wie port_DC0_cam_data, nur für Kamera 2. Verarbeitung in „camreader“ 2.
port_DC3_cam_clk	In	1 Bit	Wie port_DC0_cam_clk, nur für Kamera 3.
port_ DC3_cam_data	In	8 Bit	Wie port_DC0_cam_data, nur für Kamera 3. Verarbeitung in „camreader“ 3.

Das „top“ –Modul hat allerdings nicht nur die Aufgabe die einzelnen Module und Ports miteinander zu verbinden, sondern auch mit Hilfe von DCMs die einzelnen Module mit der benötigten Taktfrequenz zu versorgen. Außerdem achtet es darauf, dass die einzelnen Module nach einem Reset erst dann wieder starten, wenn das Taktsignal stabil ist. Wird nun von außen ein Reset ausgelöst, werden alle Module sowie die DCMs zurückgesetzt. Der Reset der einzelnen Module wird nun so lange gehalten, bis sowohl das von außen anliegende Resetsignal wieder inaktiv ist als auch sich die DCMs stabilisiert haben, also Ihre Lock-Signale aktiv sind. Je nachdem ob das Reset-Signal von außen high- oder low-aktiv ist, muss die top.vhd angepasst werden.

5.3.4 camreader.vhd

Der Camreader verarbeitet das Signal, welches er von einer Kamera erhält. Dieses Modul wird entsprechend vier Mal im gesamten System verwendet. Tabelle 5-2 beschreibt die Ports des Moduls.

Tabelle 5-2 - Beschreibung der Ports des Moduls camreader.vhd

Name	Richtung	Breite	Beschreibung
cam_data	In	8 Bit	Eingang eines Kamerasignals. Dieser Port wird über das Top-Modul nach außen geleitet.
cam_clk	In	1 Bit	Hier liegt das Taktsignal der entsprechenden Kamera an.
Clk	In	1 Bit	Taktsignale (80Mhz) für das Modul, wird vom Top-Modul geliefert.
Reset	In	1 Bit	High aktiver Reset

valid	Out	1 Bit	Wird auf high gesetzt, wenn das Modul mit der Kamera synchronisiert ist. Wird mit dem Schedule-Modul verbunden
block_ready	Out	1 Bit	Wird auf high gesetzt, wenn ein Datenblock übergeben werden kann. Ist wie alle folgenden Signale mit dem Schedule-Modul verbunden.
next_block	In	1 Bit	Signal vom Schedule-Modul, welches high gesetzt wird, wenn ein Datenblock übernommen wurde.
frametoogle	Out	1 Bit	Gibt eine steigende Flanke aus, sobald im Videostream ein Bild zu Ende ist, was gleichbedeutend mit dem Beginn des nächsten Bildes ist.
Data	Out	256 Bit	Bus zur Datenübergabe an den Scheduler

Dieses Modul bildet aus Sicht des Schedule-Moduls einen Wrapper um den Videoeingang. Anfangs benötigt es eine gewisse Zeit um sich mit der Kamera zu synchronisieren. Sobald das erfolgt ist, wird das Valid-Signal auf high gesetzt, und der Schedule kann beginnen mit dem Camreader zu kommunizieren. Wenn der Camreader 256 Bit von der Kamera empfangen hat, legt er diese Daten an den Data-Bus und setzt das block_ready-Signal auf high. Der Schedule hat von diesem Zeitpunkt an nun 1185,184ns Zeit, die Daten zu lesen. Auf diese Zeitspanne kommt man wie folgt: Mit einem Takt von 27MHz werden jeweils 8 Bit von der Kamera geliefert, oder anders gesagt, alle 37,037ns liegen 8 Bit an. Daher dauert es 1185,184ns bis 256 Bit von der Kamera empfangen wurden. Wird das Datenpaket nicht innerhalb dieser Zeit abgeholt, wird es einfach verworfen. Daher ist es unbedingt nötig, dass der Schedule immer rechtzeitig die Daten abholt, damit es zu keinem Datenverlust kommt. Sobald der Schedule die Daten gelesen hat, meldet er das an den Camreader zurück, indem er das next_block-Signal auf High zieht. Als Antwort wird das block_ready Signal wieder auf Low gesetzt worauf auch das next_block-Signal wieder zurückgenommen wird. Solche Handshakes werden auch in den anderen Modulen öfters eingesetzt, werden aber im Weitern nicht mehr so genau beschrieben. Enthält ein Datenwort das SAV-Signal mit dem Wert 010, wird das Signal Frametoogle auf High gesetzt. Dort bleibt es so lange, bis das SAV-Signal mit dem Wert 110 in einem Datenpaket enthalten ist. Abbildung 5-5 zeigt eine Reihe von solchen Datenübergaben, sowie einen Flankenwechsel von Frametoogle. Es ist gut zu sehen, dass hier als letztes ein Paket gelesen wird, welches das SAV Wort enthält, danach aber keine Daten mehr von diesem Modul abgeholt

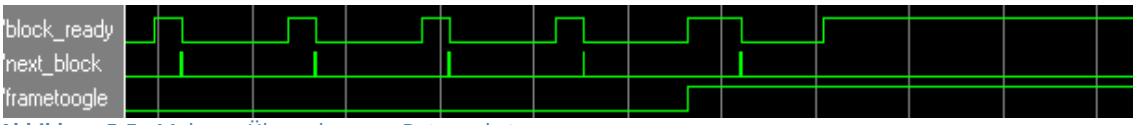

Abbildung 5-5 - Mehrere Übergaben von Datenpaketen.

werden. Der Grund dafür ist, dass nur ein Bild von der Kamera gebraucht wird, und erst wieder ein Bild von der Kamera gelesen werden muss, nachdem das gerade gelesene Bild komplett aus dem Speicher ausgelesen wurde. Mehr dazu bei der Beschreibung des Schedule-Moduls.

Zentraler Teil des inneren Aufbaus ist die in Abbildung 5-6 dargestellte Statemachine. Die Hauptaufgabe dieser Statemachine ist es, dass Frametoogle-Signal zu generieren. Sobald eine positive Flanke am Clocksignal auftritt, wird das dazugehörige Datenwort in einen internen, insgesamt 256bit großen Puffer geschrieben und mittels der Statemachine untersucht. Ist der Puffer voll, werden die Daten in den Ausgangspuffer transferiert und es wird, wie bereits oben beschrieben, das block_ready-Signal gesetzt. An dieser Stelle ist zu erwähnen, dass sowohl das Datenwort als auch das Taktsignal der Camera durch zwei Puffer geschleust werden, bevor sie analysiert werden, um das Auftreten von metastabilen Zuständen möglichst zu vermeiden. Insgesamt werden die Signale mit 80 MHz abgetastet wodurch das Shannon-Theorem erfüllt ist und daher garantiert werden kann, dass jede Flanke im Taktsignal der Kamera erkannt wird. Erkennt die Statemachine in vier hintereinander folgenden Datenwörtern die Sequenz FF0000EC wird das frametoogle-Signal auf low gesetzt. Wie bereits erwähnt, handelt es sich bei FF0000 um eine Präambel der ein Synchronisationswort folgt. Bei EC handelt es sich um das SAV Wort mit den Wert „110“ welches in der Austastlücke zwischen dem ersten und zweiten Halbbild vorkommt. Sobald also die Vertikalaustastlücke zwischen dem ersten und zweiten Halbbild gefunden wurde, wird begonnen nach der Sequenz FF0000AB zu suchen, was dem SAV Wort „010“ entspricht, wodurch das Ende eines Vollbildes oder anders gesagt die Austastlücke zwischen einem zweiten und ersten Halbbild erkannt wird. Sobald dieses gefunden wird, wird das Frametoogle-Signal auf High gesetzt um das Ende, beziehungsweise den Anfang des Vollbildes an das Schedule-Modul zu melden und die Statmachine beginnt von vorne zu arbeiten.

Da die Camreader die Schnittstelle zwischen den einzelnen Kameras und dem restlichen Design bilden, sind sie ideal geeignet um Mechanismen für die Fehlertoleranz einzubauen. Besonders wichtig ist in diesem Zusammenhang, dass ein Fehler, wie zum Beispiel der Ausfall einer Kamera nicht den Ausfall des gesamten Controllers zur Folge hat. Wenn dies der Fall wäre, wäre die Ausfallswahrscheinlichkeit des gesamten Systems um ein vielfaches höher, als wenn nur eine Kamera verwendet werden würde.

Es stellt sich die Frage, welche Fehler überhaupt Auswirkungen auf das Gesamtsystem haben können. Wie bereits in Abschnitt 5.3.1 erwähnt, wird der RAM in vier Teile geteilt und in jedem Teil wird ein Bild abgelegt. Die Bilder werden nun der Reihe nach ausgelesen und sobald ein Bild gelesen wurde, wird ein neues Bild in den Speicher übertragen. Dafür ist solange Zeit, bis die anderen drei Bilder gelesen wurden. Weiters wurde erwähnt, das die Bilder dort zusammen gefügt werden, wo zum ersten Mal das SAV Wort „010“ auftritt. Es muss also garantiert werden, dass innerhalb der vorgegebenen Zeitspanne ein komplettes Bild eingelesen wird und dass dieses Bild mit dem SAV Wort „010“ beginnt und endet.

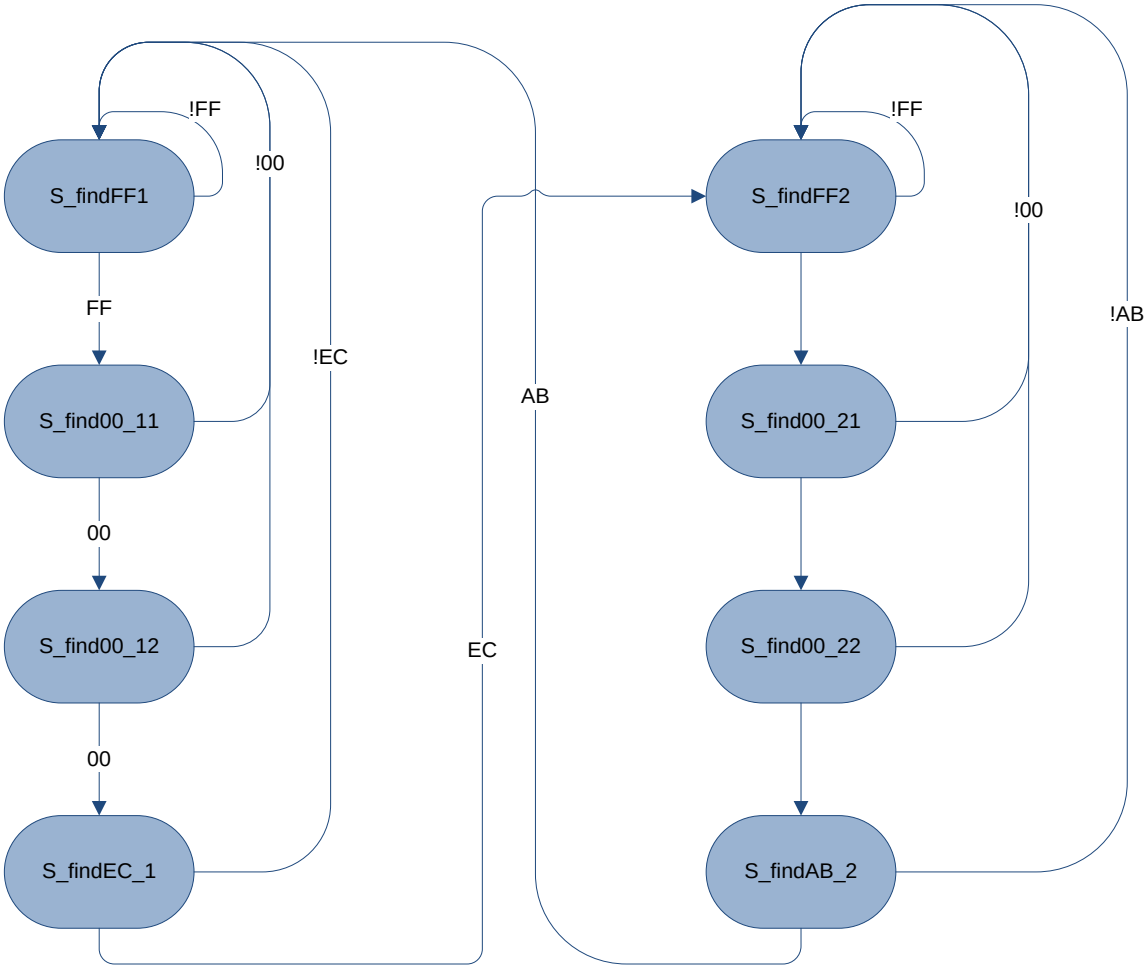

Abbildung 5-6 - Statemachine des Camreaders.

Es wurde anfangs von folgenden Fehlerszenarien ausgegangen: Bitfehler, vorübergehender Ausfall einer oder mehrerer Leitungen, vorübergehender Ausfall aller Leitungen, Defekt der Kamera oder der Leitung, sowie Fehlkonfiguration der Kamera. Da aber nicht erkannt werden kann, ob es sich um einen vorübergehenden Ausfall der Leitungen oder der Kamera handelt, oder ob es wirklich zu einem Defekt gekommen ist, wird davon ausgegangen, dass nach einiger Zeit wieder normal Daten von der Kamera empfangen werden.

Um auf die verschiedenen Fehler entsprechend reagieren zu können, sind im Camreader-Modul eine Reihe von Mechanismen eingebaut, die während der Startphase entsprechend initialisiert werden. Nachdem das Modul zurückgesetzt wurde, wird die Zeit zwischen zwei positiven Flanken des Clocksignals der Kamera gemessen. Sobald diese Variable bekannt ist, kann ein vorübergehender Ausfall, beziehungsweise ein Bitfehler in der Taktleitung erkannt werden und entsprechende Maßnahmen ergriffen werden. Weiters werden die Zeiten eruiert, die zwischen dem jeweiligen zweimaligen Auftreten von der Sequenz FF0000AB und FF0000EC vergehen. Tritt während der Messung ein Fehler in der Taktleitung auf, wird die Messung wiederholt. Diese Messergebnisse werden mit vorher definierten Werten verglichen, um eine Fehlkonfiguration erkennen zu können. Sind die Messungen abgeschlossen, wird das Signal Valid auf High gesetzt und das Modul beginnt zu arbeiten.

Es wäre beispielsweise auch möglich, dass eine Kamera oder die Leitung bereits beim Start des Moduls ausgefallen ist, und so nie die Messungen beendet werden können. Daher läuft im Hintergrund eine Art Watchdog mit, der, falls die Messungen nach einer längeren Zeit noch immer nicht erfolgreich waren, die Erwartungswerte durch Defaultwerte ersetzt, das Valid-Signal auf High setzt und das Modul in den Arbeitszustand versetzt.

Auch im laufenden Betrieb werden weiterhin die verschiedenen Werte gemessen und die Erwartungswerte fortlaufend angepasst. Dabei wird darauf geachtet, dass gewisse Grenzwerte nicht überschritten werden. Falls es zu so einer Überschreitung kommen sollte, werden die Grenzwerte verwendet. Kommt es nun zum Ausfall der Taktleitung, wird das erkannt und das Modul beginnt selbstständig mit der zuletzt bekannten Taktgeschwindigkeit die Daten von der Datenleitung zu lesen. Natürlich werden nun auch vorübergehend die Justierungen an den Erwartungswerten eingestellt. Sobald das Taktsignal erkannt wurde, wird dieses wiederum verwendet und das System passt die Erwartungswerte dem Takt wieder an.

Ähnlich verhält es sich mit dem Auftreten der Sequenzen FF0000AB und FF0000EC. Treten diese nach einer gewissen Zeit nicht auf, werden sie in den Datenstrom eingefügt und das frametoogle-Signal entsprechend gesetzt. Damit ist sichergestellt, dass auch im Falle, dass

etwa die Kamera ausfällt, vorrübergehend das Schedule-Modul mit Dummy-Daten versorgt wird. Werden die entsprechenden Sequenzen wieder erkannt, synchronisiert sich das Camreader Modul wieder mit dem Datenstrom.

Bitfehler sind für das System nur bei den oben beschriebenen Sequenzen von Bedeutung, in denen die entsprechenden Synchronisationswörter kodiert sind. Diese Synchronisationswörter sind wie bereits erwähnt so kodiert, dass Einzelfehler korrigiert werden können. Daher reagiert die Statemachine nicht nur auf AB und EC, sondern auch auf alle Wörter, die nach der CCIR 656 Norm diese Wörter kodieren. Ein weiterer Punkt ist, dass diese Sequenz mehrfach in Folge auftritt, so dass im Falle, dass das erste Auftreten nicht erkannt wurde, wahrscheinlich eines der nächsten erkannt wird. Damit erhöht sich zwar kurzzeitig die Dauer der Vertikalaustastung, allerdings muss kein künstliches Signal wie oben beschrieben eingefügt werden. Dies geschieht erst, wenn alle entsprechenden Signalwörter nicht erkannt wurden.

Durch diese Mechanismen ist sichergestellt, dass die nachfolgenden Module zumindest mit Dummy-Daten versorgt werden. Da diese Dummy-Daten regelmäßig die Wörter AB und EC enthalten, ist außerdem sichergestellt, dass es zu keinen Problemen bei der Verarbeitung der Daten kommt.

Dieses Design lässt auch die Verwendung von Kameras zu, deren Datenstrom eine geringere Taktrate als die in der CCIR 656 Norm genannte besitzt. Hierfür müssen allerdings die Grenzwerte in dem Camreader-Modul angepasst werden.

5.3.5 schedule_robot.vhd

Die Aufgabe des Schedule-Modules ist es, den Zugriff auf den RAM, der mit Hilfe des sdram_controller-Moduls erfolgt, zu koordinieren. Weiters achtet es darauf, dass wenn ein neues Bild in den RAM geladen werden kann, die Daten von dem entsprechenden Camreader bezogen werden. Ebenso stellt es sicher, dass der Präprozess, der im Abschnitt 5.3.7 genauer behandelt wird, rechtzeitig mit den Daten versorgt wird. Da dieses Modul mit insgesamt sechs anderen Modulen kommuniziert (vier Camreader, der Sdram-Controller sowie der Präprozess) ist die hohe Anzahl an Signalen, die in Tabelle 5-3 beschrieben werden, nicht verwunderlich.

Tabelle 5-3 - Beschreibung der Ports des Moduls schedule_robot.vhd

Name	Richtung	Breite	Beschreibung
Clk	In	1 Bit	Taktsignal (80Mhz) für das Modul, wird vom Top-Modul geliefert.
Reset	In	1 Bit	High-aktiver Reset .
Cam0_valid	In	1 Bit	Mit camreader der Kamera 0 verbunden (siehe Port „valid“ in Kapitel 5.3.4)
Cam0_block_ready	In	1 Bit	Mit camreader der Kamera 0 verbunden (siehe Port „block_ready“ in Kapitel 5.3.4)
Cam0_frametoogle	In	1 Bit	Mit camreader der Kamera 0 verbunden (siehe Port „frametoogle“ in Kapitel 5.3.4)
Cam0_data	In	256 Bit	Mit camreader der Kamera 0 verbunden (siehe Port „Data“ in Kapitel 5.3.4)
Cam0_next_block	Out	1 Bit	Mit camreader der Kamera 0 verbunden (siehe Port „next_block“ in Kapitel 5.3.4)
Cam1_valid	In	1 Bit	Mit camreader der Kamera 1 verbunden (siehe Port „valid“ in Kapitel 5.3.4)
Cam1_block_ready	In	1 Bit	Mit camreader der Kamera 1 verbunden (siehe Port „block_ready“ in Kapitel 5.3.4)
Cam1_frametoogle	In	1 Bit	Mit camreader der Kamera 1 verbunden (siehe Port „frametoogle“ in Kapitel 5.3.4)
Cam1_data	In	256 Bit	Mit camreader der Kamera 1 verbunden (siehe Port „Data“ in Kapitel 5.3.4)
Cam1_next_block	Out	1 Bit	Mit camreader der Kamera 1 verbunden (siehe Port „next_block“ in Kapitel 5.3.4)
Cam2_valid	In	1 Bit	Mit camreader der Kamera 2 verbunden (siehe Port „valid“ in Kapitel 5.3.4)
Cam2_block_ready	In	1 Bit	Mit camreader der Kamera 2 verbunden (siehe Port „block_ready“ in Kapitel 5.3.4)
Cam2_frametoogle	In	1 Bit	Mit camreader der Kamera 2 verbunden (siehe Port „frametoogle“ in Kapitel 5.3.4)
Cam2_data	In	256 Bit	Mit camreader der Kamera 2 verbunden (siehe Port „Data“ in Kapitel 5.3.4)

Cam2_next_block	Out	1 Bit	Mit camreader der Kamera 2 verbunden (siehe Port „next_block“ in Kapitel 5.3.4)
Cam3_valid	In	1 Bit	Mit camreader der Kamera 3 verbunden (siehe Port „valid“ in Kapitel 5.3.4)
Cam3_block_ready	In	1 Bit	Mit camreader der Kamera 3 verbunden (siehe Port „block_ready“ in Kapitel 5.3.4)
Cam3_frametoogle	In	1 Bit	Mit camreader der Kamera 3 verbunden (siehe Port „frametoogle“ in Kapitel 5.3.4)
Cam3_data	In	2 Bit	Mit camreader der Kamera 3 verbunden (siehe Port „Data“ in Kapitel 5.3.4)
Cam3_next_block	Out	1 Bit	Mit camreader der Kamera 3 verbunden (siehe Port „next_block“ in Kapitel 5.3.4)
Preproz_Ready	In	1 Bit	Wird mit dem Präprozessor verbunden. Sobald der Präprozessor neue Daten benötigt, wird das dem Schedule-Modul durch ein High-Signal an dieser Leitung mitgeteilt.
Preproz_Data	Out	256 Bit	Bus zur Datenübertragung zwischen dem Präprozessor und dem Schedule -Modul
Preproz_Write	Out	1 Bit	Signalisiert dem Präprozessor, dass neue Daten anliegen.
Ram_Data_from	In	256 Bit	Bus um Daten vom SDRAM-Controller zu empfangen
Ram_Data_to	Out	256 Bit	Bus um Daten an den SDRAM-Controller zu übertragen.
Ram_Ready	In	1 Bit	Der SDRAM-Controller setzt dieses Signal auf High wenn die letzte Operation abgeschlossen ist und er bereit für den nächsten Befehl ist.
Ram_EOF	In	1 Bit	Wird vom SDRAM-Controller auf High gesetzt, wenn die letzten Daten eines Bildes übertragen werden.
Ram_We	Out	1 Bit	Aktiviert die Schreibfunktion im SDRAM-Controller
Ram_Re	Out	1 Bit	Aktiviert die Lesefunktion im SDRAM-Controller

Cam	Out	2 Bit	Wenn Daten zum SDRAM-Controller übertragen werden, wird damit angegeben, von welcher Kamera sie stammen. Wenn Daten vom SDRAM-Controller empfangen werden sollen wird über diesem Bus ausgewählt, von welcher Kamera die Daten stammen sollen
out_led	Out	1 Bit	Wird auf High gesetzt sobald der Reset nicht mehr aktiv ist.

Der Funktionsweise des Moduls lässt sich am besten mit Hilfe von Prozessen, wie sie aus der Software bekannt sind, beschreiben. Es gibt Write Prozesse, die die Daten von einem jeweils zugewiesenen Camreader lesen und an den SDRAM-Controller übergeben. Weiters gibt es einen Read Prozess, der Daten vom SDRAM-Controller anfordert und an den Präprozessor übergibt. Außerdem gibt es noch eine Art Leerlauf Prozess, auf den später noch eingegangen wird. Bis auf den Leerlauf Prozess benötigen alle Prozesse fast immer Zugriff auf eine gemeinsame Ressource, den RAM oder genauer gesagt den SDRAM-Controller. Daher gibt es eine Art Scheduler, der jeweils einen Prozess aktiviert, so dass nicht zwei Prozesse versuchen gleichzeitig auf den SDRAM-Controller zuzugreifen. Dieser Prozess läuft nun eine Zeit lang und gibt dann die Kontrolle wieder an den Scheduler ab, der darauf den nächsten Prozess aktiviert. Dabei ist die Zeit, wie lang ein Prozess maximal aktiv ist, bekannt. Das ist deswegen besonders wichtig, da, wie bereits in vorherigen Kapitel erwähnt wurde, bei den Camreadern sichergestellt werden muss, dass die vom Camreader bereitgestellten Daten innerhalb einer gewissen Zeitspanne gelesen werden müssen, da sie sonst verworfen werden. Wie später noch genauer beschrieben wird, verhält es sich beim Präprozessor ähnlich, allerdings müssen bei diesem Modul innerhalb einer gewissen Zeitspanne Daten geliefert werden, damit es nicht zu einem Fehler kommt. Bevor nun der eigentliche Scheduler genauer betrachtet wird, werden zuerst die verschiedenen Prozesse genauer erörtert.

Der Read Prozess hat die Aufgabe, Daten für den Präprozess vom SDRAM-Controller abzurufen. Der RAM ist in vier Teile geteilt und in jedem dieser Teile befindet sich ein Bild von jeweils einer Kamera. Der Read Prozess lässt sich nun von dem SDRAM-Controller Stück für Stück ein Bild aus einem der Speicherbereiche liefern. Sobald ein Bild komplett ausgelesen wurde, werden die Daten vom nächsten Speicherbereich angefordert, sowie der Write Prozess gestartet,

der mit der Kamera assoziiert ist, dessen Speicherbereich nun leer ist. Der Prozess ist in Abbildung 5-7 dargestellten, in Zuständen geschieht folgendes:

- **sleep:** Während sich der Prozess im Zustand „sleep“ befindet, ist er gerade nicht aktiv. Der Prozess kann sich nicht selbst wieder aktivieren, sondern muss vom Schedule aktiviert werden, worauf er sich nun wieder im Zustand „startup“ oder im Zustand „waiting“ befindet.
- **startup:** Nach dem Reset des Moduls befindet sich der Read Prozess im Zustand „startup“. Hier wird geprüft, ob sich von jeder Kamera bereits ein Bild im RAM befindet. Dies erkennt der Prozess daran, dass außer ihm nur noch Leerlauf Prozesse vorhanden sind, was gleichbedeutend damit ist, dass alle Write Prozesse beendet wurden. Später wird noch auf den Start des Moduls genauer eingegangen. Sind noch Write Prozesse aktiv, wechselt er in den Zustand „sleep“, worauf nun wieder ein Write Prozess abgearbeitet wird. Andernfalls wechselt er in den Zustand „waiting“.
- **waiting:** In diesem Zustand wird überprüft, ob der Präprozessor Daten anfordert. Falls er keine benötigt, wechselt der Prozess in den „sleep“ Zustand und überprüft, wenn er wieder aktiviert wird erneut, ob Daten gebraucht werden. Falls jedoch welche benötigt werden, wird dem SDRAM-Kontroller signalisiert, dass dieser weitere Daten aus dem RAM von dem Bild liefern soll, welches zur Zeit gerade bearbeitet wird. Anschließend wechselt er in den Zustand „getdata“.
- **getdata:** Hier wird überprüft, ob der SDRAM-Controller bereits mit dem Auslesen der Daten aus dem RAM begonnen hat, was immer zutreffen sollte. Sobald dies der Fall ist, wird die Signalisierung, dass mit dem Auslesen begonnen werden soll, beendet, und es wird in den Zustand „transfer“ gewechselt.
- **transfer:** In diesem Zustand wird solange gewartet, bis der SDRAM-Controller signalisiert, dass die Daten erfolgreich aus dem RAM gelesen wurden und am Ram_Data_from-Port anliegen. Sowie dies der Fall ist, werden die Daten an den Datenport des Präprozessors angelegt und diesem signalisiert, dass er die Daten lesen kann. Falls es sich um die letzten Daten eines Bildes handelte, wird der Read Prozess gestartet, der mit der Kamera assoziiert ist, von der das Bild stammte. Wie später noch beschrieben wird, wird der Prozess auf eine Art Warteliste gesetzt, und sobald wie möglich in den Schedule aufgenommen. Weiters wird intern gespeichert, dass beim nächsten Durchlauf die Daten vom nächsten Speicherbereich, also das nächste Bild, angefordert werden müssen. Abschließend wird in den Zustand „transfer_end“ gewechselt.

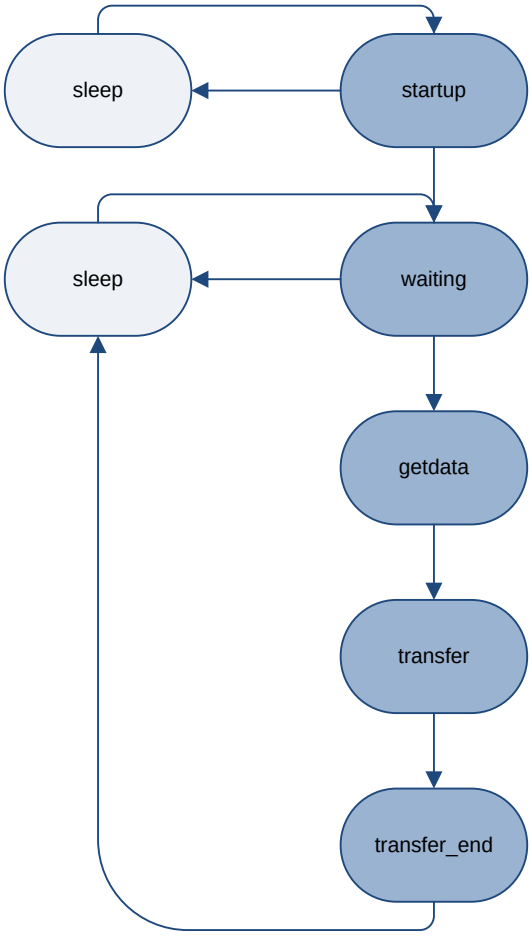

Abbildung 5-7 - Darstellung des Read Prozesses

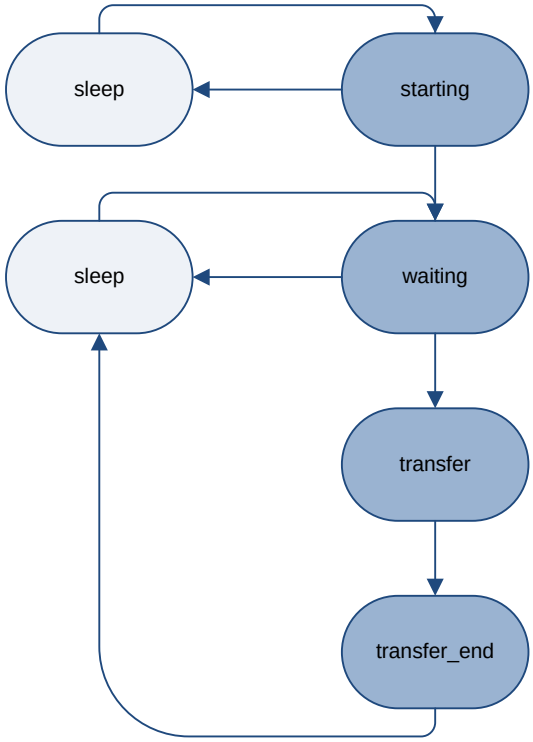

Abbildung 5-8 - Darstellung des Write Prozesses

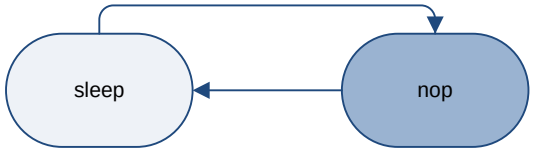

Abbildung 5-9 - Darstellung des Leerlauf Prozesses

- **transfer_end:** Es wird in dem Zustand solange gewartet, bis der Präprozessor die Daten übernommen hat. Sobald dies der Fall ist, wird die Signalisierung, dass Daten bereit liegen zurückgenommen und in den Sleepmode gewechselt. Wenn der Prozess wieder aktiviert wird, startet der Durchlauf im Zustand „wait“ von Neuem.

Da der Prozess jedesmal gleich lange auf die Reaktion der anderen Module warten muss, ist es nicht schwer die maximale Ausführungszeit zu bestimmen, wobei unter Ausführungszeit in dem Zusammenhang die Anzahl der Takte gemeint ist, die zwischen der Aktivierung des Prozesses und des Wechselns in den Zustand „sleep“ vergehen. Im Falle des Read-Prozesses sind das maximal 27 Takte.

Jeder Write-Prozess ist mit einer Kamera assoziiert, entsprechend gibt es vier Write-Prozesse. Die Aufgabe dieser Prozesse ist es, mittels des SDRAM-Controllers ein Bild von einer Kamera in den entsprechenden Speicherbereich im RAM zu übertragen. Sobald dies geschehen ist, beendet sich der Prozess von selbst und wird erst bei Bedarf vom Read-Prozess neu gestartet. Abbildung 5-8 stellt diesen Prozess dar. Die einzelnen Zustände lassen sich folgendermaßen beschreiben:

- **sleep:** Stellt wiederum die Phase der Innaktivität des Prozesses dar.
- **starting:** Der Prozess soll ein komplettes Bild in den RAM transferieren. Dafür ist es notwendig, den Anfang eines Bildes zu erkennen, was durch eine positive Flanke auf dem frame-toogle Port zu erkennen ist. Solange also auf dem Port noch ein High ausgegeben wird, muss nichts weiter gemacht werden und der Prozess kann wiederum in den Zustand „sleep“ wechseln. Ist das Signal auf Low, wechselt der Prozess in den „wait“-Zustand
- **wait:** In diesem Zustand wird geprüft, ob neue Daten vom Camreader anliegen. Falls nein wird in den Zustand „sleep“ gewechselt. Falls ja, und bisher noch keine positive Flanke auf der frame-toogle Leitung aufgetreten ist, wird ebenfalls in den Sleep-Zustand gewechselt. Tritt nun zum ersten Mal die Flanke auf oder ist sie bereits aufgetreten, werden die Daten zum SDRAM-Controller transferiert und diesem signalisiert, dass sie in dem zu der Kamera gehörenden Speicherplatz gespeichert werden. Tritt zum zweiten Mal eine Flanke auf der frame-toogle Leitung auf, was gleichbedeutend mit dem Ende des Bildes ist, wird dies vermerkt, damit der Prozess am Ende des Durchlaufes beendet wird. Abschließend wird dem Camreader signalisiert, dass die Daten gelesen wurden und es wird in den Zustand „transfer“ gewechselt.
- **transfer:** Hier wird nun überprüft, ob der SDRAM-Controller mit der Übertragung der Daten zum RAM bereits begonnen hat. Sobald dies der Fall ist, wird in den Zustand „transfer_end“ gewechselt.
- **transfer_end:** Es wird solange gewartet, bis der SDRAM-Controller signalisiert, das er die Schreib-Operation beendet hat. Ist dies der Fall, wird überprüft, ob ein Vermerk vorliegt, dass das Bild zu Ende ist. Ist dies ebenfalls der Fall, beendet sich der Prozess und versucht als letzte Handlung, seinen Slot im Schedule einem Prozess zuzuweisen der auf der Warteliste steht. Steht kein solcher Prozess zu Verfügung, wird stattdessen ein Leerlaufprozess in den Schedule eingefügt. Falls das

Bild nicht zu Ende war, wechselt der Prozess in den „sleep“-Zustand und setzt bei erneuter Aktivierung beim Zustand „waiting“ fort.

Die maximale Ausführungszeit beträgt 22 Takte.

Der Leerlauf-Prozess (Abbildung 5-9) besteht genau genommen nur aus einem echten Zustand. Sobald dieser Prozess aktiviert ist, kontrolliert er, ob sich ein Prozess auf der Warteliste befindet. Ist dies der Fall, übergibt er seinen Slot an diesen. Ansonsten wechselt er wieder in den „sleep“-Zustand.

Beim Schedule handelt es sich im Grunde um eine Liste von Prozessen wobei immer nur einer dieser Prozesse, zum Beispiel der an der ersten Stelle, gerade aktiv ist. Sobald dieser in den Zustand „sleep“ wechselt, wird er, wie in Abbildung 5-10 zu sehen ist, hinten angereiht und der nächste Prozess wird aktiviert.

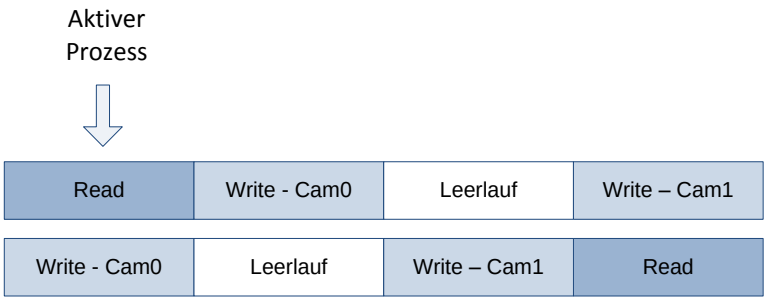

Abbildung 5-10 - Der Schedule

Die Zeit die benötigt wird, damit alle Prozesse einmal aktiviert wurden, wird im folgenden Zyklus genannt. Da mit einem 256 Bit Bus gearbeitet wird und ungefähr alle 37 Nanosekunden 8 Bit pro Kamera empfangen werden beziehungsweise ausgegeben werden müssen, liegen nur alle 1185ns neue Daten an den Schnittstellen der Camreader zum Schedule-Modul an. In dieser Zeit muss also jede Kamera, die Daten liefert, die benötigt werden, abgefragt werden, sowie der Readprozess ausgeführt werden. Wichtig in dem Zusammenhang ist, dass es ausreicht, wenn der Schedule drei Write Prozesse vorsieht. Benötigt ein Bild im Videosignal t Sekunden bis es übertragen wurde, dann benötigt der Camreader im schlimmsten Fall beinahe 2t Sekunden, bis er ein komplettes Bild in den RAM übertragen hat. Der Grund dafür ist der, dass es sein kann, das der Camreader kurz nach Beginn eines Bildes aktiviert wurde, wodurch er dieses Bild abwarten muss, damit er das folgende Bild lesen kann. So gesehen, müssten nur zwei Camreader im Schedule vorhergesehen sein da so sicher gestellt ist, das Camreader0 nie gleichzeitig mit Camreader3 aktiv ist, ebenso wie Camreader2 und Camreader4. Da es aller-

dings, wie bereits beschrieben, zu Fehlern bei der Erkennung der Bildanfänge kommen kann, muss damit gerechnet werden, dass im schlimmsten Fall sogar drei Reader aktiv sind. Die Fehlermechanismen in den Camreader - Modulen garantieren, dass in jedem Fall das Auslesen eines Bildes unter 3t bleibt. Daher muss im schlimmsten Fall in einem Zyklus der Read-Prozess sowie drei Write-Prozesse ausgeführt werden. In Summe nimmt also ein Zyklus 93 Takte in Anspruch. Wird das System mit 80Mhz getaktet, dauert also ein Zyklus 93*12,5ns = 1162,5ns womit die Zeitvorgaben erfüllt sind.

Im Modul selbst sind die einzelnen Prozesse als Zustände einer Statemachine realisiert. Der Schedule ist ein Array, der die zu den einzelnen Prozessen gehörenden Zustände enthält. Der Zustand, der sich an der Stelle 0 befindet, wird ausgeführt. Um in den „sleep-Mode“ zu wechseln, rotiert ein Prozess die Liste einfach um eine Stelle weiter, wodurch er selbst hinten angereiht wird und ein Nachfolgeprozess aktiviert wird. Der Prozess beendet sich, indem er sich aus dem Array löscht und durch den Leerlaufprozess oder einen anderen Prozess ersetzt wird. Auch bei der Warteliste handelt es sich um einen Array, wo ein Prozess, der vom Reader-Prozess gestartet wird, einfach in den Array aufgenommen wird.

Abschließend sei noch erwähnt, dass nachdem das Modul resetet wurde, es zuerst einmal solange wartet bis von allen viel Camreadern das Valid-Signal empfangen wurde. Anschließend wird der Schedule mit dem Read-Prozeß, sowie mit den Write-Prozessen für die Camreader 0 bis 2 initialisiert. Camreader3 wird auf die Warteliste gesetzt, so dass er in den Schedule aufgenommen wird, sobald einer der anderen Camreaders ein komplettes Bild gespeichert hat. Wurde von allen vier Camreadern jeweils ein Bild gespeichert, befinden sich neben dem Read-Prozess nur Leerlaufprozesse im Schedule, wodurch der Read-Prozeß beginnt, die Bilder wieder auszulesen. In Abbildung 5-11 sind die Aktivitätsphasen der vier Write-Prozesse zu sehen.

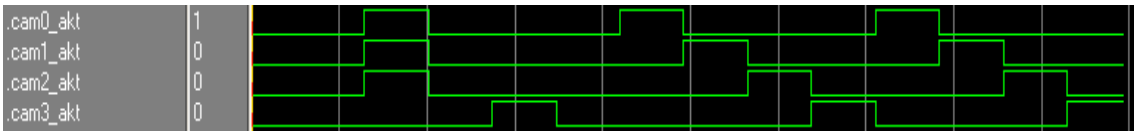

Abbildung 5-11 - Aktivitäten der Write-Prozesse

5.3.6 sdram_controller.vhd

Der SDRAM-Controller vereinfacht den Zugriff auf den RAM. Er stellt den gesamten RAM in der Form von vier FIFO-Speichern dar, die jeweils 4MB an Daten fassen. Dieses Modul besitzt die in Tabelle 5-4 dargestellten Schnittstellen nach außen:

Tabelle 5-4 - Beschreibung der Ports des Moduls sdram_controller.vhd

Name	Richtung	Breite	Beschreibung
Ram_Addr	Out	12 Bit	Dient zur Auswahl einer Speicheradresse beim Lesen und Schreiben vom beziehungsweise am RAM. Dieses, sowie alle nachfolgenden Port die mit dem Präfix „Ram_“ beginnen, werden über das Top-Modul mit den gleichnamigen Ports des RAMs verbunden.
Ram_Ba	Out	2 Bit	Dieser Bus dient zur Wahl einer der vier Banken des RAMS
Ram_Cke	Out	1 Bit	Dient zur Maskierung des Clocksignals
Ram_Cs_n	Out	1 Bit	Chip select
Ram_Ras_n	Out	1 Bit	Der Row address strobe ist eine der drei Ports für die Steuerbefehle.
Ram_Cas_n	Out	1 Bit	Colum address strobe, eine weitere Steuerleitung.
Ram_We_n	Out	1 Bit	Write enable, die dritte Steuerleitung.
Ram_Dqm	Out	4 Bit	Datenmaske zur Auswahl einzelner Bytes beim Lesen und Schreiben.
Ram_Dq	Inout	32 Bit	Bidirektionaler Datenbus um die eigentlichen Daten zu transportieren.
Ss_Clk	In	1 Bit	Clocksignal, 80 MHz, für das Modul, wird vom Top-Modul geliefert.
Ss_reset	In	1 Bit	Resetsignal, wird ebenfalls vom Topmodul geliefert.
Ss_Cam	In	2 Bit	Zur Auswahl eines der vier FIFOs was äquivalent mit der Auswahl der Kamera ist, von der ein Bild stammt. Dieser Bus ist wie alle nachfolgenden Ports mit dem Schedule-Modul verbunden
Ss_Data_in	In	256 Bit	Bus, an dem das Modul die Daten erhält, welche auf den RAM geschrieben werden sollen.
Ss_Data_out	Out	256 Bit	Bus über den die vom RAM gelesenen Daten dem Schedule zur Verfügung gestellt werden.

Ss_We	In	1 Bit	Steuersignal um die Schreibefunktion zu aktivieren
Ss_Re	In	1 Bit	Steuersignal um die Lesefunktion zu aktivieren
Ss_Ready	Out	1 Bit	Wird auf High gesetzt, sobald die letzte Aktion beendet wurde und das Modul bereit ist den nächsten Befehl zu verarbeiten.
Ss_EOF	Out	1 Bit	Wird auf High gesetzt sobald die letzten Daten aus einem der vier FIFOs gelesen wurden.

Der RAM selbst ist intern in Form von vier Banken aufgebaut, die mittels des Ba-Buses einzeln angesteuert werden können. Jede dieser Banken kann 1048576 Worte zu 32 Bit speichern, also 4MB. Eine Bank besteht aus 4096 Reihen mit jeweils 256 Spalten. Aufgrund der Tatsache, dass der RAM vier Banken besitzt und vier Kameras verwendet werden, ist es naheliegend, jeder Kamera eine eigene Bank zuzuweisen. Weil die Daten, die von den Kameras geliefert werden, genau in derselben Reihenfolge auch wieder ausgelesen werden, ist es ebenso naheliegend, nach außen hin die Adressierung des RAMs zu verbergen und einfach das Verhalten eines FIFOs zu zeigen. Daher muss vom Schedule-Modul nur angegeben werden, von welcher Kamera die Daten stammen beziehungsweise gelesen werden sollen, was der Auswahl einer Bank gleich kommt.

Die Bedienung nach außen hin erfolgt folgendermaßen: Sobald Ss_Ready auf High ist, kann entweder auf einen der vier FIFOs geschrieben oder von einem gelesen werden. Sollen Daten geschrieben werden, muss über die Ss_Cam Schnittstelle der gewünschte FIFO ausgewählt werden. Weiters müssen am Ss_Data_in Bus die Daten anliegen, die in den FIFO geschrieben werden sollen. Durch setzen von Ss_We wird das Schreiben aktiviert, was daran zu erkennen ist, dass das Ss_Ready Signal ab der nächsten Flanke auf Low geht, da das Modul arbeitet. Dies ist der passende Zeitpunkt, um das Ss_We Signal zurückzusetzen. Sobald der Schreibvorgang beendet wurde, wird Ss_Ready wieder auf High gesetzt, und der nächste Befehl kann ausgeführt werden.

Um von einem FIFO zu lesen muss wiederum der entsprechende FIFO mittels Ss_Cam ausgewählt und das Ss_Re auf High gesetzt werden. Das Modul reagiert wiederum damit, dass Ss_Ready auf Low gesetzt wird. Sobald der Lesevorgang beendet ist, wird das Ss_Ready Signal auf High gesetzt und die gewünschten Daten liegen an Ss_Data_out an. Sie liegen an diesem Bus so lange an, bis der nächste Lesebefehl ausgeführt wird. Wurden die letzten Daten aus

einem FIFO ausgelesen, wird das durch Ss_EOF angezeigt. In Tabelle 5-5 sind nochmals die Befehle für das Modul zusammen gefasst.

Tabelle 5-5 - Befehle des sdram_controller.vhd

Ss_We	Ss_Re	Befehl
1	0	Schreibe Daten
1	1	Schreibe Daten
0	0	Kein Befehl
0	1	Lese Daten

Im Modul selbst wurden zuerst die benötigten Befehle für den RAM mit Hilfe von Funktionen modelliert. In Tabelle 5-6 wird beschrieben, auf welche Werte die Leitungen zum RAM von den verschiedenen Funktion gesetzt werden, wobei „X“ für Werte steht, die nicht von der Funktion festgesetzt werden beziehungsweise keinen Einfluss auf den Befehl haben. Es wurden nicht alle Befehle, die es für den RAM gibt, benötigt. Die gesamten Befehlssatz sowie Informationen zum Timing und zur Ansteuerung des RAMs sind im JEDEC Standard JESD21-C festgelegt und werden im Fall des K4S283233F-M auch im Datenblatt [Sam02] beschrieben. Im Weiteren wird nur auf die benötigten Befehle und Funktionen eingegangen.

Tabelle 5-6 - Befehle des RAMs

Befehl:	Cke	Cs_n	Ras_n	Cas_n	We_n	Dqm(3-0)	Ba(1-0)	Addr(11-0)
Nop	1	0	1	1	1	0000	XX	XXXXXXXXXXXX
Auto_refresh	1	0	0	0	1	0000	XX	XXXXXXXXXXXX
Active	1	0	0	1	1	0000	XX	XXXXXXXXXXXX
Read	1	0	1	0	1	0000	XX	0100XXXXX000
Write	1	0	1	0	0	0000	XX	0100XXXXX000
Precharge_all	1	0	0	1	0	0000	00	010000000000
Load_mode_reg	1	0	0	0	0	0000	00	XXXXXXXXXXXX

Die Dqm-Leitungen haben im Grunde keinen Einfluss auf die Befehle. Sie dienen nur zum Maskieren von Daten auf dem Datenbus. Im RAM selbst sind die Daten in 32 Bit Blöcken angeordnet und entsprechend ist der Datenbus auch 32 Bit breit. Will man nun nur 1 Byte in einen Block überschreiben, kann man dies tun, indem man mit Hilfe des Dqm-Busses die Bytes, die

nicht gespeichert werden sollen, maskiert. Da in diesem Design jedesmal alle 32 Bit verwendet werden, wird der DQM immer auf 0000 gesetzt, damit nichts maskiert wird. Zu den einzelnen Befehlen:

- **Nop:** Steht für „No Operation Command“ und liegt an dem RAM an, wenn sonst kein Befehl ausgeführt werden soll. Sowohl die aktuell anliegende Banknummer als auch Adresse haben keinen Einfluss.
- **Autor_refresh:** Da es sich bei dem SDRAM um einen flüchtigen Speicher handelt, muss der Inhalt des Speichers regelmäßig aufgefrischt werden, da er sonst verloren geht. Mit Hilfe des Autorefreshs lässt sich jeweils eine Reihe auffrischen. Laut Datenblatt muss der gesamte Speicher alle 64 Millisekunden aufgefrischt werden, was gleichbedeutend damit ist, dass innerhalb dieser Zeitspanne 4096-mal dieser Befehl ausgeführt werden muss.
- **Active:** Dient zum Aktivieren einer Reihe. Um auf einen gewissen Speicherbereich zuzugreifen, muss bevor der eigentliche Schreibe oder Lesebefehl ausgeführt wird, die Reihe, in der sich der Speicher befindet, aktiviert werden. Auf dem Adressbus liegt bei der Aktivierung des Befehls die Adresse der gewünschten Reihe an.
- **Read:** Dient zum Auslesen eines Speicherbereiches. Interessant ist hierbei die Adresse, die anliegt, wenn dieser Befehl ausgeführt wird. Durch Setzen der Adressleitung 10 auf High, wird der Auto-Precharge aktiviert. Unter Precharge versteht man das Deaktivieren einer Zeile. Wird der Auto-Precharge aktiviert, wird der Precharge für die aktuelle Reihe zum frühest möglichen Zeitpunkt nach dem Auslesen der Daten aktiviert. Mit Hilfe der Leitungen 2 bis 0 kann, wenn ein Burst verwendet wird, angegeben werden, in welcher Reihenfolge die Daten geliefert werden. In diesem Design wird ein Burst von 8 verwendet. Ein Burst von 8 bedeutet, dass nicht nur die 32 Bit Daten, die sich an der durch die übrigen Adressleitungen adressierten Stelle befinden, ausgegeben werden, sondern auch nach einander die Daten, die sich in den darauffolgenden sieben Stellen befinden. Es können also recht rasch in Summe 256 Bit ausgelesen werden, ohne dass für jedes Wort ein eigener Lesebefehl ausgeführt werden muss. In Abbildung 5-16 ist so ein Lesevorgang mit einem Burst von 8 zu sehen.
- **Write:** Befehl zum Auslesen von Daten. Die Bits bei der Adresse haben dieselbe Bedeutung wie beim Read-Befehl. Wieder wird der Auto-Precharge aktiviert sowie ein Burst von 8 verwendet.
- **Precharge_all:** Dieser Befehl dient dazu, alle Reihen, die zur Zeit aktiv sind, zu deaktivieren. Im Grunde handelt es sich um einen normalen Precharge Befehl, mit dem sich

normalerweise nur die aktive Reihe in einer Bank deaktivieren lässt. Durch Setzen des 10ten Bits in der Adresse, werden nun mit diesem Befehl, die aktiven Reihen von allen Banken deaktiviert und die eventuell anliegende Bankadresse ignoriert.

- **Load_mode_reg:** Mit Hilfe dieses Befehles ist es möglich einige Einstellungen am RAM vorzunehmen. Wie später noch gezeigt wird, wird dieser Befehl im aktuellen Design dazu verwendet um die CAS-Latency auf 3 Taktzyklen zu setzen, sowie den Burstmode zu aktivieren. Die CAS-Latency gibt die Anzahl der Takte an, die vergehen, bis nach dem Anlegen einer Adresse Daten geliefert werden.

Das Modul besteht im Grunde aus der in Abbildung 5-12 dargestellten Statemachine, wobei sich hinter allen Zuständen, mit Ausnahme des Idle Zustandes, eine weitere Statemachine verbirgt.

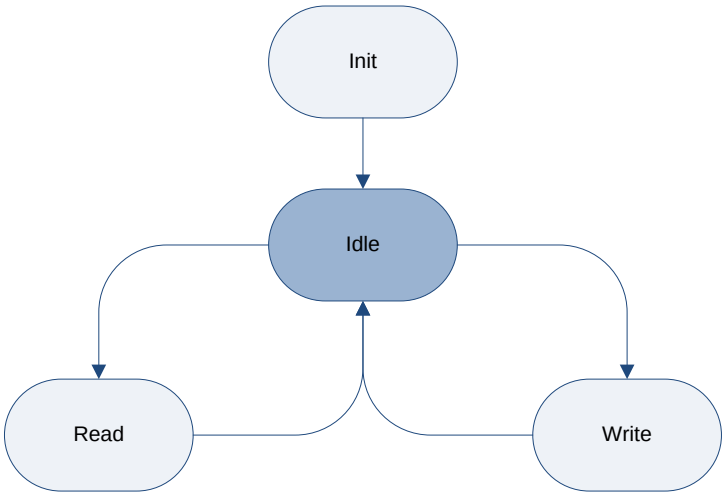

Abbildung 5-12 - Statemachine des SDRAM-Kontrolles

Nach dem anfänglichen Reset wird zuerst der RAM Initialisiert. Ist dieser Vorgang abgeschlossen, wechselt das Modul in den Idle Zustand, wo es auf Befehle wartet. Je nach Befehl (siehe Tabelle 6.5) wird eine Schreib oder Leseoperation durchgeführt und anschließend wieder in den Idle Zustand gewechselt.

In Abbildung 5-13 sind nun die Abläufe in den Zuständen Init, Read sowie Write genauer dargestellt. Die nop -Befehle werden immer dann verwendet, wenn aufgrund von verschiedenen Timingbedingungen, die wieder dem Datenblatt [Sam02] zu entnehmen sind, gewartet werden muss, bis ein neuer Befehl gegeben werden kann.

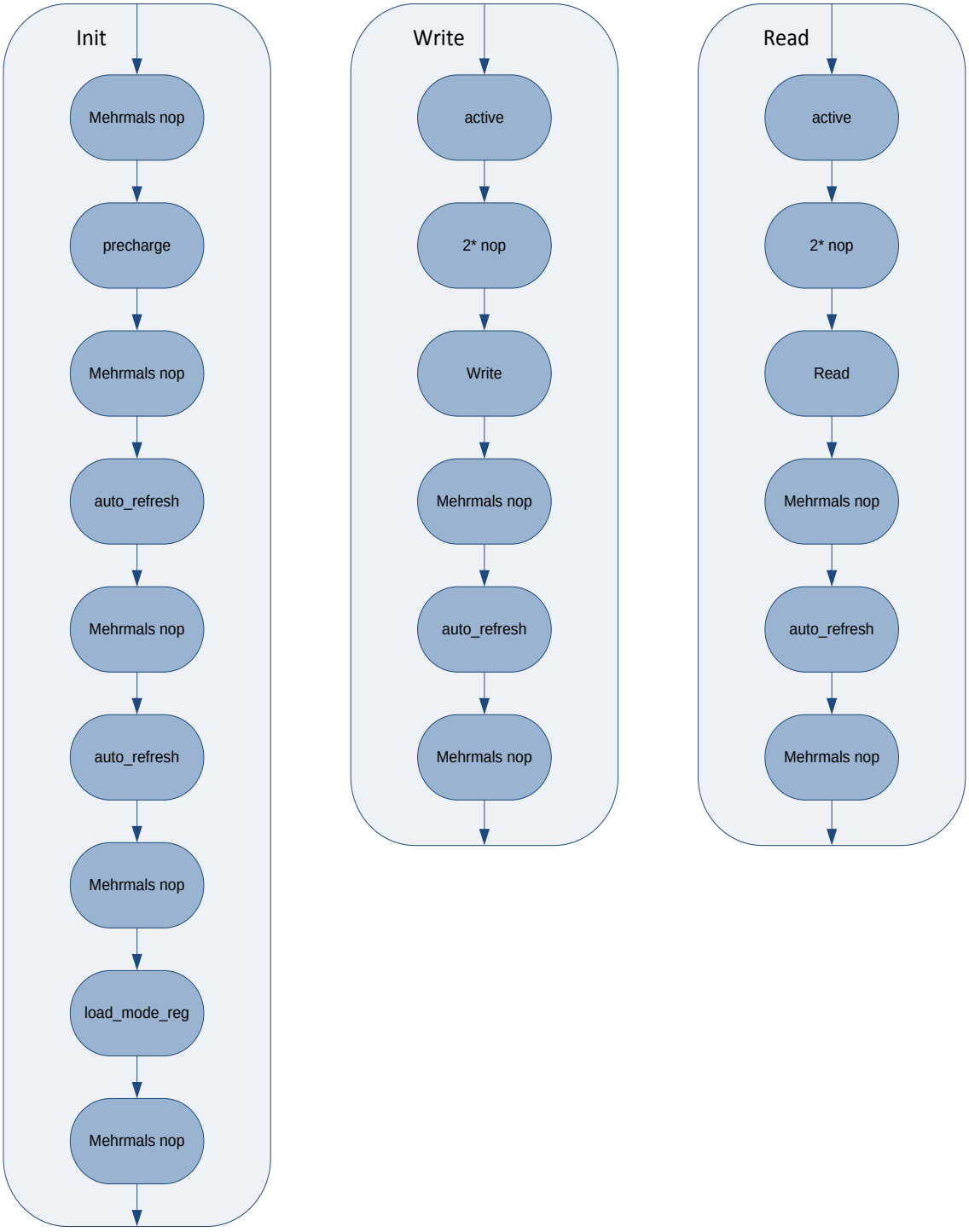

Abbildung 5-13 - Ablauf der Initialisierung sowie einer Lese und Schreibe Operation

Bei der Initialisierung des RAMs wird zuerst einige Zeit gewartet bevor der precharge_all_bank Befehl gegeben wird, damit alle Reihen deaktiviert sind. Nach einiger Zeit wird nun zwei Mal der autorefresh aktiviert und schließlich mit dem load_mode_reg Befehl der RAM, wie bereits

beschrieben, konfiguriert. Diese Sequenz wurde vom Hersteller angegeben. Abbildung 5-14 zeigt diese Initialisierungs- Sequenz.

Um auf den RAM zu schreiben, wird zuerst die Reihe aktiviert, in der sich der gewünschte Speicherbereich befindet. Nachdem zwei Takte vergangen sind, kann der eigentliche Write Befehl gegeben werden. Im selben sowie in den darauffolgenden Takten werden jeweils 32 Bit auf den Speicher geschrieben. Sobald das Schreiben abgeschlossen ist, wird automatisch die aktuelle Reihe mittels Auto-precharge deaktiviert. Abschließend wird noch ein Autorefresh ausgeführt. Weil während des Betriebes laufend auf den RAM geschrieben und von diesem gelesen wird und jedesmal auch ein Autorefresh ausgeführt wird, ist sicher gestellt, dass es alle 64ms zum Auffrischen des gesamten Speichers kommt. Eine Schreiboperation ist in Abbildung 5-15 dargestellt.

Das Auslesen aus dem RAM läuft ähnlich wie das Schreiben ab. Zuerst wird eine Reihe aktiviert und wenig später wird der eigentliche Lesebefehl gegeben. In den darauffolgenden Takten werden die Daten nun vom RAM geliefert und anschließend die Reihe deaktiviert. Abschließend wird wie bei der Schreiboperation ein Autorefresh durchgeführt. Abbildung 5-16 stellt diesen Vorgang dar.

Für die Adressierung werden vom SDRAM-Controller jeweils zwei Zähler pro Bank verwaltet. Ein Zähler zählt die Anzahl an 256 Bit Paketen, die in der jeweiligen Bank gespeichert sind. Durch diesen Zähler kann daher immer die Adresse für das nächste Packet bestimmt werden. Der zweite Zähler zählt die Anzahl der gelesenen Pakete für die jeweilige Bank und dient zur Adressierung für kommende Leseoperationen. Weites kann mit Hilfe dieser Zähler festgestellt werden, ob das letzte Datenpaket aus dem Speicher gelesen wurde. Ist dies der Fall, wird bei der Ausgabe des Paketes dieser Umstand durch Setzen von Ss_EOF signalisiert. Weiters werden in diesem Fall die beiden Zähler wieder zurückgesetzt.

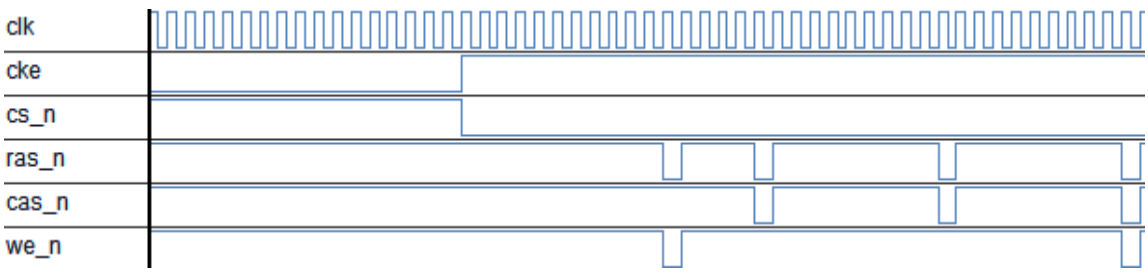

Abbildung 5-14 - Initialisierung des SDRAMs

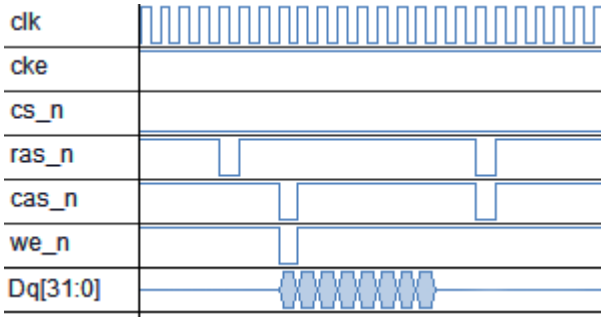

Abbildung 5-15 - Schreiboperation am SDRAM

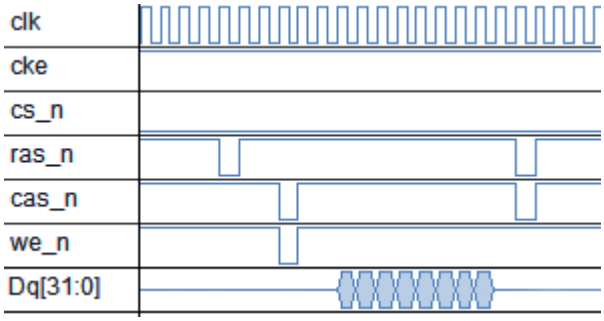

Abbildung 5-16 - Leseoperation am SDRAM

5.3.7 preprocess.vhd

Der Präprozess hat vor allem zwei Aufgaben: Zum einen bereitet er die Daten auf, die später von der Sink ausgegeben werden und zum anderen transferiert er zusammen mit dem async-buffer Modul die Daten in eine andere Taktdomain. In Tabelle 5-7 werden die Ports des Moduls beschrieben.

Tabelle 5-7 - Ports des Präprozessors

Name	Richtung	Breite	Beschreibung
Clk	In	1 Bit	Taktsignal (80MHz) für das Modul, wird vom Top-Modul bereitgestellt
Reset	In	1 Bit	Resetport des Moduls
schdl_Data	In	256 Bit	Dient zum Erhalt von Daten vom Schedule-Modul
schdl_Write	In	1 Bit	Wird vom Schedule-Modul auf High gesetzt, sobald neue Daten am Data-Port anliegen
schdl_Ready	Out	1 Bit	Über diesen Port wird ein neues Datenpaket vom

			Schedule beantragt.
sink_Data	Out	256 Bit	Dient zur Datenübergabe an das Puffer-Modul (bzw. an die Sink, wenn sich diese in der selben Taktdomain befindet)
sink_Pointer	Out	5 Bit	Dient zur Übergabe des Pointers an das Puffer-Modul
sink_Write	Out	1 Bit	Wird auf High gesetzt um die Lesefunktion des Puffers zu aktivieren.
sink_Ready	In	1 Bit	Wird vom Puffer auf High gesetzt, sobald dieser bereit ist Daten aufzunehmen.

Wie bereits erwähnt werden bei der Ausgabe des neuen Videostreams die einzelnen Bilder an der Stelle aneinander gehängt, wo das SAV- Wort mit dem Wert AB das erste Mal auftritt. Die Camreader zeigen bereits an, in welchem 256 Bit Block dieses SAV Wort vorkommt, so dass jeweils ein komplettes Bild im RAM gespeichert wird. Der erste und der letzte Datenblock eines jeden Bildes enthält nun das SAV Wort sowie vor beziehungsweise nach dem Wort Daten, die nicht mehr zu dem eigentlichen Bild gehören.

Die Aufgabe des Präprozessor-Moduls ist es, den Datenblock so zu verändern, dass zuerst die zum aktuellen Bild gehörigen Daten stehen und danach erst die Daten, die noch zu vorherigen beziehungsweise zum nachfolgenden Bild gehören. An das nachfolgende Modul wird nun nicht nur das neue Datenpaket übertragen, sondern auch ein Pointer, der angibt, wie viele Stellen des Datenpakets nicht zum aktuellen Bild gehören, also verworfen werden sollen.

In Abbildung 5-17 ist nun dieser Vorgang für verschiedene Fälle dargestellt. Das erste Datenpaket eines Bildes wird so verändert, dass das SAV Wort am Anfang steht. Bei allen anderen Datenpaketen muss das Paket selbst nicht verändert werden. Bei Datenpaketen, die nur Daten vom aktuellen Bild enthalten, ist der Pointer immer Null, da ja keine Daten verworfen werden sollen. Beim letzten Datenpaket zeigt der Pointer auf das SAV Wort, da dieses und alle nachfolgenden Daten verworfen werden müssen, wenn gleich anschließend das erste Datenpaket des nächsten Bildes ausgegeben werden soll.

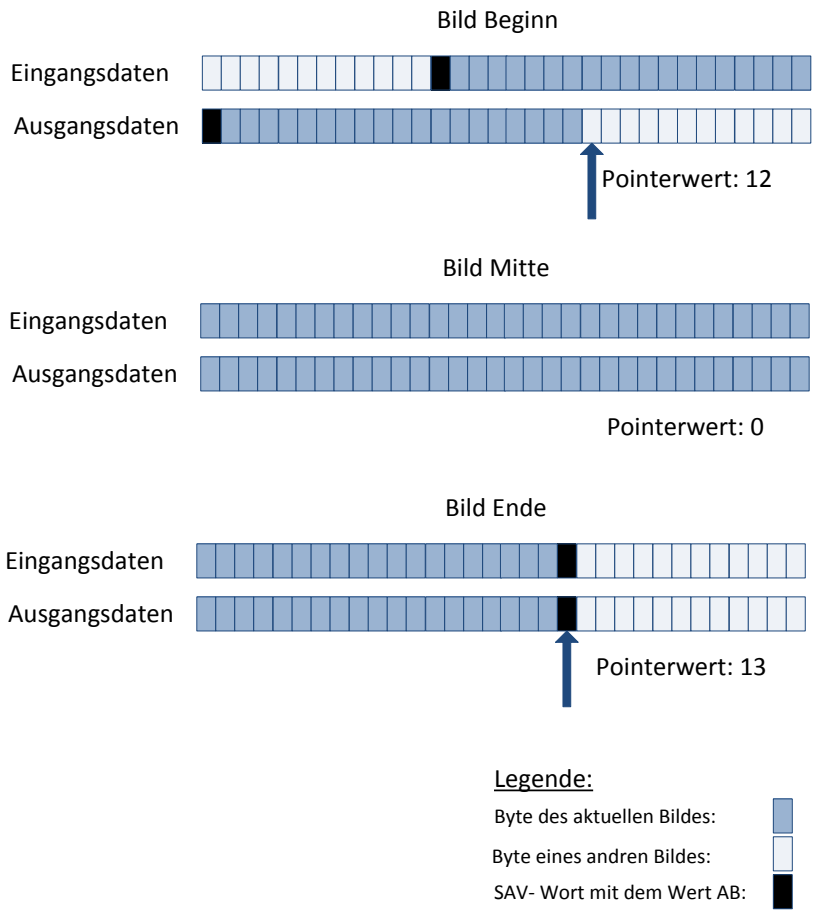

Abbildung 5-17 - Verarbeitung im Präprozess

Intern ist das Modul als Statemachine aufgebaut. Diese wird in Abbildung 5-18 dargestellt. Nach einem Reset startet das Modul im Zustand gedata. In diesem Zustand wird der Port schdl_Ready auf High gesetzt um Daten anzufordern. Sobald der Schedule das Modul informiert, dass Daten anliegen, was durch ein High-Signal auf dem Port schdl_Write geschieht, wird schdl_Ready wieder auf Low gesetzt und die Statemachine wechselt in den Zustand work. In diesem Zustand wird durch eine Statemachine, die der in Abbildung 5-6 dargestellten recht ähnlich ist, die Position des SAV Wortes mit dem Wert AB bestimmt und gegebenenfalls die Daten im Datenpaket, wie oben beschrieben, verschoben. Sobald diese Operation abgeschlossen ist, wechselt die Statemachine in den Zustand write_start. Hier wartet das Modul, bis der nachfolgende Puffer bereit ist das Datenpaket anzunehmen. Ist dies der Fall, wird über den Write-Port der Lesevorgang des Puffers aktiviert und die Statemachine wechselt in den Zustand write_end. Dort wird wiederum solange gewartet, bis der Puffer die Daten gelesen hat. Sobald dies geschehen ist, wird in den Zustand getdata gewechselt und der Zyklus beginnt von neuem.

Eine Besonderheit ist, dass das sink_Ready-Signal, welches anzeigt, dass das Puffer-Modul bereit ist Daten aufzunehmen, durch zwei Flipflops geroutet wird. Der Grund dafür ist, dass das nachfolgende Modul in einer anderen Taktdomain liegt worauf in dem Kapitel 5.3.8 noch genauer eingegangen wird.

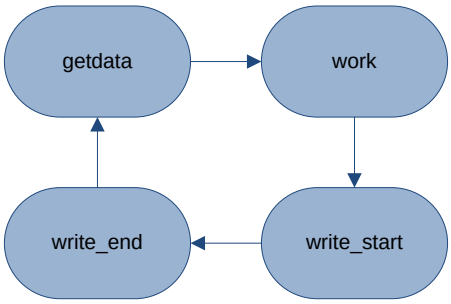

Abbildung 5-18 - Statemachine des Präprozesses

5.3.8 async_buffer.vhd

Die Aufgabe dieses Moduls ist es, zusammen mit dem Präprozessor die Übergabe der Daten in die andere Taktdomain zu bewerkstelligen. Wird nur eine Taktdomain verwendet, kann dieses Modul aus dem Design entfernt werden. Aufgrund der Tatsache, dass dieses Modul auch wie ein Puffer wirkt, wird die Verzögerung, die durch die Datenübergabe zwischen den Taktdomains entsteht, vor dem Sink-Modul verborgen. In Tabelle 5-8 werden die Ports des Moduls beschrieben.

Tabelle 5-8 - Ports des async_buffer- Moduls

Name	Richtung	Breite	Beschreibung
Clk	In	1 Bit	Taktsignal (108MHz) für das Modul, wird vom Top-Modul bereitgestellt
Reset	In	1 Bit	Resetport des Moduls
preproz_Data	In	256 Bit	Datenanbindung an den Präprozess
preproz_Pointer	In	5 Bit	Dient zum Erhalt des Pointers vom Präprozess
preproz_Write	In	1 Bit	Wird vom Präprozess auf High gesetzt, sobald neue Daten anliegen
preproz_Ready	Out	1 Bit	Wird auf High gesetzt, sobald neue Daten gelesen werden können

sink_Data	Out	256 Bit	Dient zur Datenübergabe an die Sink
sink_Pointer	Out	5 Bit	Dient zur Übergabe des Pointers an das Sink Modul
sink_Write	Out	1 Bit	Wird auf High gesetzt um die Lesefunktion der Sink zu aktivieren.
sink_Ready	In	1 Bit	Wird von der Sink auf High gesetzt, sobald er bereit ist Daten aufzunehmen.

Da die Übergabe der Daten vom Präprozess zu diesem Modul zwischen zwei Taktdomains erfolgt, ist es nötig, eine geeignete Synchronizer-Schaltung [Dal98] zu entwerfen. Abbildung 5-19 zeigt diese Schaltung, wobei wie der Synchronizer des Präprozesses bereits im Präprozessor-Modul enthalten ist.

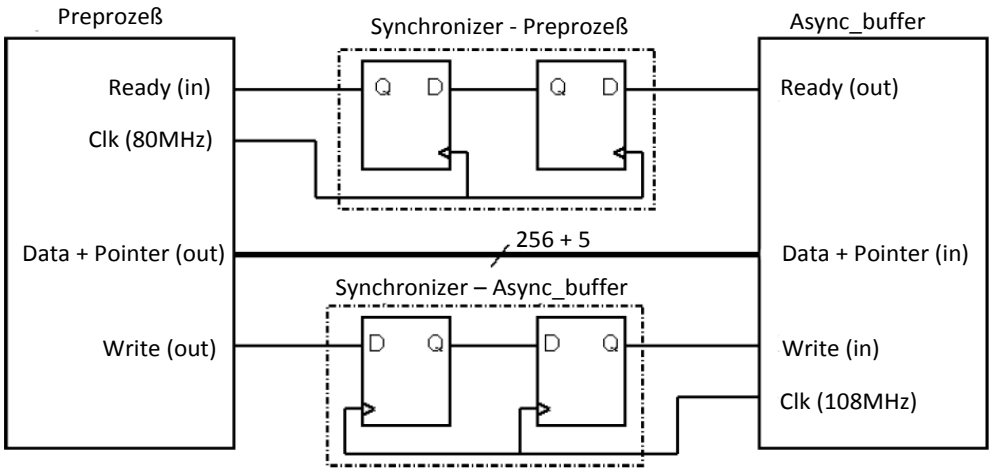

Abbildung 5-19 - Synchronizer

Für die Datenübergabe wird nun wiederum ein Handshake verwendet. Kann das Puffermodul Daten aufnehmen, setzt es Ready auf High. Der Präprozess empfängt das Signal mit einiger Verzögerung. Sobald er nun Daten für das Puffer-Modul besitzt, legt er diese an den Daten- beziehungsweise an den Pointer-Bus an, und setzt Write auf High. Die Signale bleiben nun solange angelegt, bis das Ready-Signal des Puffer-Moduls auf Low geht. Das Ready-Signal wird vom Puffer-Modul auf Low gesetzt, sobald es das Write-Signal empfangen und die Daten in den internen Puffer kopiert hat. Diese Schaltung wird unter anderem in [Gin03] noch genauer beschrieben.

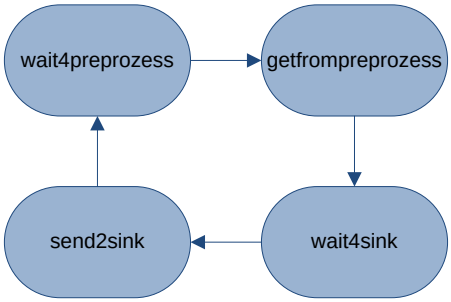

Abbildung 5-20 - Statemachine von async_buffer.vhd

Das Modul ist mittels einer recht einfachen Statemachine, die in Abbildung 5-20 zu sehen ist, aufgebaut. Im State wait4preprozess wird das preproz_Ready-Signal auf High gesetzt und gewartet, bis das Write-Signal eintrifft. Ist dies der Fall, werden die anliegenden Daten in den internen Puffer kopiert und in den State getfrompreproz gewechselt. Hier wird wieder das Ready-Signal zurückgenommen und gewartet, bis dies beim Write-Signal ebenfalls der Fall ist. Nun erfolgt der Wechsel in den Zustand wait4sink, wo solange gewartet wird, bis das Sink-Modul bereit ist, Daten aufzunehmen. Sobald dies der Fall ist, wird das Write-Signal für die Sink aktiviert und in den Zustand send2sink gewechselt. Dort wird solange gewartet, bis die Sink die Daten übernommen hat, worauf wieder in den Anfangszustand gewechselt wird.

5.3.9 sink.vhd

Die Aufgabe des Sink- Moduls ist es, das Ausgabesignal aus den Datenpaketen zu generieren. Dies geschieht dadurch, dass die Daten, die in den Datenpaketen enthalten sind, byteweise bis zum Pointer ausgegeben werden. In Tabelle 5-9 werden die Ports des Moduls beschrieben.

Tabelle 5-9 - Ports des sink- Moduls

Name	Richtung	Breite	Beschreibung
Clk	In	1 Bit	Taktsignal (108MHz) für das Modul, wird vom Top-Modul bereitgestellt
Reset	In	1 Bit	Resetport des Moduls
sink_Data	In	256 Bit	Datenbus
sink_Pointer	In	5 Bit	Bus für den Pointer
sink_Write	In	1 Bit	Port für Write- Befehl vom Async_buffer- Modul
sink_Ready	Out	1 Bit	Wird High gesetzt, sobald das Modul bereit ist Daten aufzunehmen.
port_data	Out	8 Bit	Port für die Ausgabe des CCIR 656 Signals

port_PxlClk	Out	1Bit	Ausgabeport des Taktsignals (27Mhz) des CCIR 656 Signals

Das Modul besteht aus insgesamt vier Speichern, die jeweils ein Datenpaket sowie dem dazugehörigen Pointer aufnehmen können. Des Weiteren gibt es zwei Statemachines, die auf diese Speicher zugreifen und ihren Zugriff mit Hilfe von vier Speicherzellen, die als Semaphoren dienen, koordinieren. In Abbildung 5-21 ist der Aufbau dargestellt.

Die Statemachine Read hat die Aufgabe, die Kommunikation mit dem async_buffer-Modul zu übernehmen und die Daten in die Speicher zu transferieren. Dazu überprüft sie anfangs im Zustand rs_0, ob der Puffer 0 leer ist. Das erkennt sie an der Semaphore S0, die in diesem Fall Null ist. So wie sie erkennt, dass der Puffer frei ist, wird das Ready-Signal auf High gesetzt, um dem async_buffer-Modul zu signalisieren, dass Daten benötigt werden. Sobald die Daten bereit sind, wird das Write Signal vom async_buffer-Modul auf High gesetzt. Nun kann die Statemachine die Daten in den Puffer schieben, sowie die Semaphore auf Eins setzen. Weiters wird das Readysignal wieder auf Low gesetzt und in den Zustand rs_0w gewechselt. In diesem Zustand wird solange gewartet, bis das Writesignal zurückgenommen wird, und so der Handshake für die Datenübertragung beendet ist. Nun erfolgt das selbe Prozedere für alle andern Puffer. Sobald auch der Puffer 3 gefüllt ist, wird mit der Befüllung des Puffers 0 fortgesetzt, der zwischenzeitlich von der anderen Statemachine geleert wurde.

Die Statemachine Write wartet nach der Initialisierung so lange, bis alle Puffer gefüllt sind. Dann beginnt sie die Daten vom ersten Puffer auszugeben. Sobald das letzte Byte ausgegeben wurde, wird durch die Semaphore der Speicher als leer markiert, und die andere Statemachine kann ihn wieder auffüllen. Statemachine-Write fährt bei allen anderen Puffern entsprechend fort.

Wie bereits erwähnt, wird vom Schedule-Modul garantiert, dass alle 1162,5ns ein Datenpaket mit 256 Bits an den Präprozessor geliefert wird der die Daten über das async_buffer- Modul zur sink liefert. Weil jedes dieser Module die Daten sofort nach ihrem Erhalt bearbeitet und sie dann so lange behält, bis das nachfolgende Modul sie übernommen hat und ihre Bearbeitungszeit weniger als 1162,5ns beträgt, sind sie im Bezug auf das Timing transparent. Es

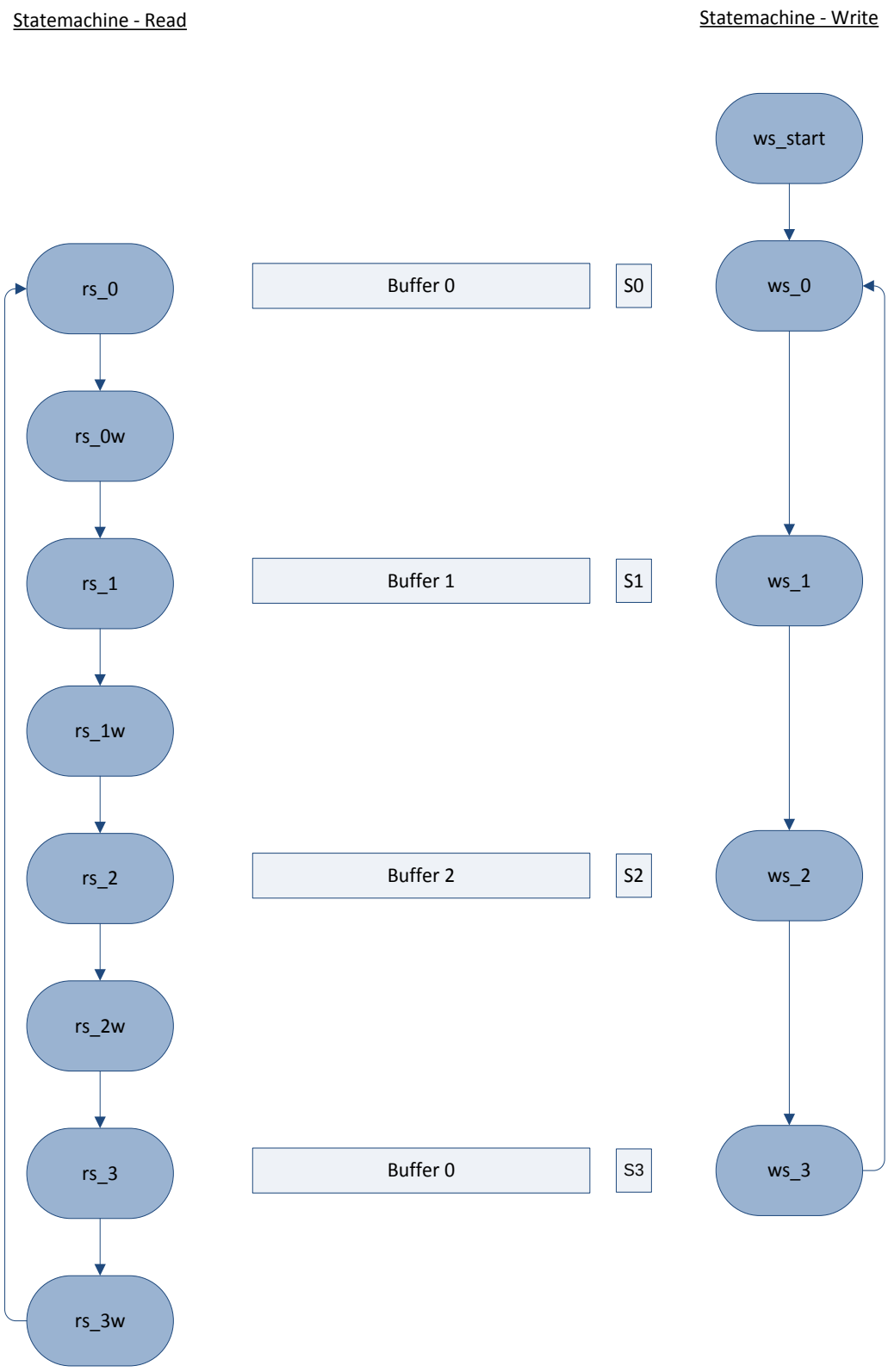

Abbildung 5-21 - Aufbau Sink – Modul

kann also davon ausgegangen werden, dass wenn von der Statemachine Read ein Paket angefordert wird, dieses nach spätestens 1162,5ns auch zur Verfügung steht. Die Ausgabe eines kompletten Datenpaketes dauert 1185ns, also etwas länger. Wenn immer das komplette Datenpaket ausgegeben werden würde, würde es reichen, wenn das Sink –Modul zwei Puffer besitzt. Einen von dem die Daten ausgegeben werden, und ein anderer, der in der Zwischenzeit befüllt wird.

Beim Übergang von zwei Bildern wäre es aber nun denkbar, dass aus zwei hintereinander folgenden Datenpaketen jeweils nur ein Byte verwendet werden kann. Dieses regelmäßige Event tritt allerdings nur, je nach Framerate, nach ein paar hundert vollständig nutzbaren Datenpaketen auf. Daher reicht die Verwendung von zwei weiteren Puffern aus, um diesen Datenengpass zu überbrücken.

5.4 Controller der Rakete

Der Controller, der in der Rakete die Signale der beiden Roboter verarbeitet, ist bis auf ein paar Veränderungen in zwei Modulen ident mit dem Controller der Roboter. Seine Aufgabe ist es nach jeweils vier Bildern zwischen den Datenströmen umzuschalten.

5.4.1 top.vhd

Da es in diesem Design nur zwei Videoquellen gibt, wurde die Anzahl der Camreader entsprechend auf zwei reduziert. Die Portliste entspricht bis auf die vier Ports, die von Camreader zwei und drei verwendet wurden, der in Kapitel 5.3.3. Ansonsten ist dieses Modul mit dem top-Modul der Roboter ident.

5.4.2 schedule_rocket.vhd

Auch dieses Modul ist großteils ident mit dem aus Kapitel 5.3.5. Die Portliste unterscheidet sich nur darin, dass die Ports, die zur Kommunikation mit den Camreadern zwei und drei gedient haben, entfernt wurden. Entsprechend gibt es auch nur mehr zwei Write Prozesse wodurch auch nur mehr maximal drei Prozesse im Schedule vorhanden sein können. Daher wurde der Schedule auch von vier auf drei Prozesse verkleinert.

Weiters wurde sowohl der Read-Prozess als auch die Write-Prozesse geringfügig verändert. Jeder Write-Prozess liest nun in Folge vier Bilder aus dem Datenstrom aus, und speichert jeweils ein Bild in einen der vier Speicherbereiche des RAMs. Danach trägt sich der Prozess wiederum aus dem Schedule aus. Der Read-Prozess wurde dahingehend verändert, dass er, nachdem er jeweils zwei Bilder vom RAM gelesen hat, einen der Write-Prozess startet. Weiters wird

das jeweils letzte Datenpaket eines Bildes, welches das SAV Wort mit dem Wert AB enthält, zweimal an den Präprozess weitergegeben, da die nachfolgenden Module jeweils zwei Bilder an der Stelle, wo das SAV auftritt, miteinander verbinden. Somit werden die vier Bilder jeweils künstlich aufgetrennt und anschließend wieder zusammengesetzt. Dies scheint zwar auf den ersten Blick etwas umständlich, bietet aber den Vorteil, dass dadurch die anderen Module nicht verändert werden müssen.

6 Software

Wie bereits erwähnt, muss der Videostream, nach dem er von der Rakete zur Erde gesendet wurde, wieder aufgetrennt werden, damit dieser betrachtet werden kann. Hierzu wurde ein Programm entwickelt, das dies ermöglicht.

6.1 JMF

Die Software ist in der Sprache Java geschrieben. Im Gegensatz zu vielen anderen Sprachen wird ein Java Programm nicht direkt auf einem Computer ausgeführt, sondern läuft in einer virtuellen Maschine, der Java virtual machine, die durch ein Programm auf dem Computer emuliert wird [Hor00]. Dadurch ist es möglich, ein Java Programm auf verschiedenen Plattformen auszuführen, ohne dass das Programm dafür neu kompiliert werden muss.

Durch die Verwendung der virtuellen Maschine ist es allerdings nicht ohne weiteres möglich, direkt auf Resourcen wie auf Videoquellen zuzugreifen. Diese Funktionalität wird nun durch das Java Media Frameworks, kurz JMF, bereitgestellt. Diese Bibliothek bietet eine Reihe von Funktionen an, um auf einfache Weise Audio und Videosignale von verschiedenen Quellen zu empfangen, sie zu verarbeiten, sowie auszugeben, beziehungsweise zu speichern. Neben einer reinen Java-Version existiert eine Reihe von Versionen, die speziell auf verschiedene Plattformen, wie etwa Windows, Linux und MacOS, zugeschnitten sind.

Das Konzept hinter JMF lässt sich vereinfacht folgendermaßen darstellen. Verschiedene Quellen wie zum Beispiel Streams von Videokarten, Kameras, Mikrofonen und Dateien, sowie Streams die über ein Netzwerk übertragen werden, werden als DataSource gekapselt [Eid04]. Diese dienen als Datenquelle für so genannte Player, die das Videobild anzeigen und den Ton ausgeben können. Eine spezielle Art von Player ist der Processor, der die Daten mit Hilfe von Filtern und Effekten verändern kann. Weiters kann der Processor auch wieder als Datenquelle dienen. Außerdem gibt es dann noch die sogenannten DataSinks, die einen Stream aufnehmen können. Beispiele dafür wären zum Beispiel Files in denen Videos gespeichert werden, oder auch ein Netzwerk über welches das Video gesendet wird [Sun99].

Durch Entwicklung entsprechender Effekte, sowie durch Anpassung der durch das JMF gegebenen Klassen, war es auf recht einfache Weise möglich, das gewünschte Programm zu erstellen. Im Weiteren werden die einzelnen erstellten Klassen grob umrissen.

6.2 Programme

Um den Videostream zu betrachten wurden zwei Programme geschrieben. Der Viewer erhält den Stream direkt von der Framegrabberkarte und zeigt alle acht in dem Stream vorhandenen Videos gleichzeitig an. Weiters bietet er eine Funktion an, um den Stream von der Karte lokal zu speichern. Das zweite Programm, der Editor, kann für die Extraktion einzelner Videos aus einem bereits gespeicherten Videostream verwendet werden. Damit ist es auch möglich, die auf die jeweilige Karte zugeschnittene Capture-Software zu verwenden, um eine möglichst gute Qualität zu erzielen. Die Programme können sowohl direkt gestartet werden, als auch über einen kleinen Startdialog.

6.2.1 RS_Start

Diese lauffähige Klasse stellt ein grafisches Interface, welches in Abbildung 6-1 zu sehen ist, zur Verfügung, über welches sowohl der Editor als auch der Viewer gestartet werden kann. Die Klasse erzeugt dazu einen JFrame auf dem mehrere Buttons platziert sind. Die Klasse implementiert auch einen ActionListener, so dass, sobald ein Button gedrückt wurde, eine Funktion der Klasse aufgerufen wird, die nun entsprechend den Viewer beziehungsweise den Editor startet.

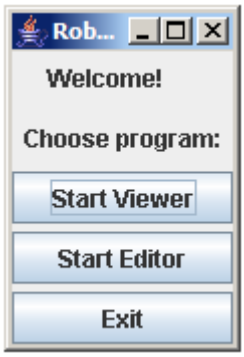

Abbildung 6-1 - Start Dialog

6.2.2 RS_Viewer

Wie bereits erwähnt dient der Viewer dazu, die Videostreams, die direkt von der Framegrabberkarte geliefert werden, zu betrachten. Dazu wird ein entsprechendes Interface zu Verfügung gestellt, welches direkt nach dem Start erzeugt wird. Anschließend wird eine Instanz der Klasse SelectSourceDialog erzeugt. Bei SelectSourceDialog handelt es sich um eine innere Klasse die eine Unterklasse von „Dialog“ ist. Entsprechend stellt sie nun einen Dialog zur Verfügung

mit dem sich, wie in Abbildung 6-2 zu sehen ist, die Videoquelle auswählen lässt, sowie das Format, in dem das Video eingespielt werden soll. Ebenso kann hier die Datei angegeben werden, in der der Videostream gespeichert werden soll. Für die Auswahl der Datei wird ein Filechooser verwendet, wobei die Auswahl der Files mit Hilfe eines Filefilters auf avi-Dateien beschränkt wurde. Der Filefilter wurde ebenfalls als innere Klasse realisiert.

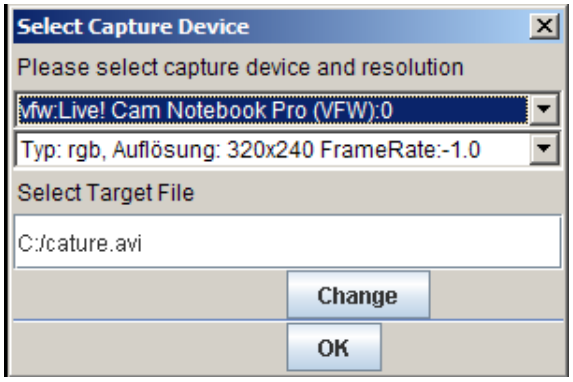

Abbildung 6-2 - Auswahldialog

Sobald nun die Videoquelle sowie die Datei ausgewählt wurden, wird aus der Videoquelle eine klonbare Datenquelle erzeugt. Durch Klonen ist es möglich, den Videostream sowohl anzuzeigen als auch lokal zu speichern. Ein Klon wird nun mit Hilfe einer Instanz von RS_Transformer auf das Format 320 zu 240 Pixel transformiert. Da jeweils vier Videos nebeneinander angezeigt werden, bieten höhere Auflösungen praktisch keine Vorteile. Anschließend wird die Ausgabe des Transformers von einer Instanz der Klasse RS_DataSourceWrapper übernommen, die in Kapitel 6.2.3 noch genauer beschrieben wird. Diese Klasse erzeugt aus jeweils acht hintereinander folgenden Bildern ein einzelnes Bild mit der Auflösung 1280 zu 480 Pixel. Der daraus resultierende Videostream hat entsprechend nur mehr ein Achtel der ursprünglichen Framerate. Dieses Ergebnis wird schließlich mit Hilfe einer Instanz der Klasse RS_prozessor am Bildschirm angezeigt, wie dies in Abbildung 6-3 zu sehen ist. Dieser Prozessor wird allerdings erst gestartet, wenn das komplette System erstellt wurde.

Aus dem im Dialog ausgewählten File wird eine DataSink erstellt. Ein weiterer Klon der ursprünglichen Quelle wird als Datenlieferant für einen weiteren Transformer verwendet, der den Videostream in ein MJPEG-Format transformiert und mit Hilfe der DataSink in dem File abspeichern kann.

Sind alle diese Objekte einmal geschaffen,miteinander verbunden, sowie bereit den Videostream zu verarbeiten, wird der Prozessor, der für die Ausgabe am Bildschirm zuständig ist, gestartet, worauf der Videostream angezeigt wird. Durch Klick auf den Button „Record“ wird

außerdem der Speichervorgang gestartet, der nun auch jederzeit unterbrochen und wieder fortgesetzt werden kann.

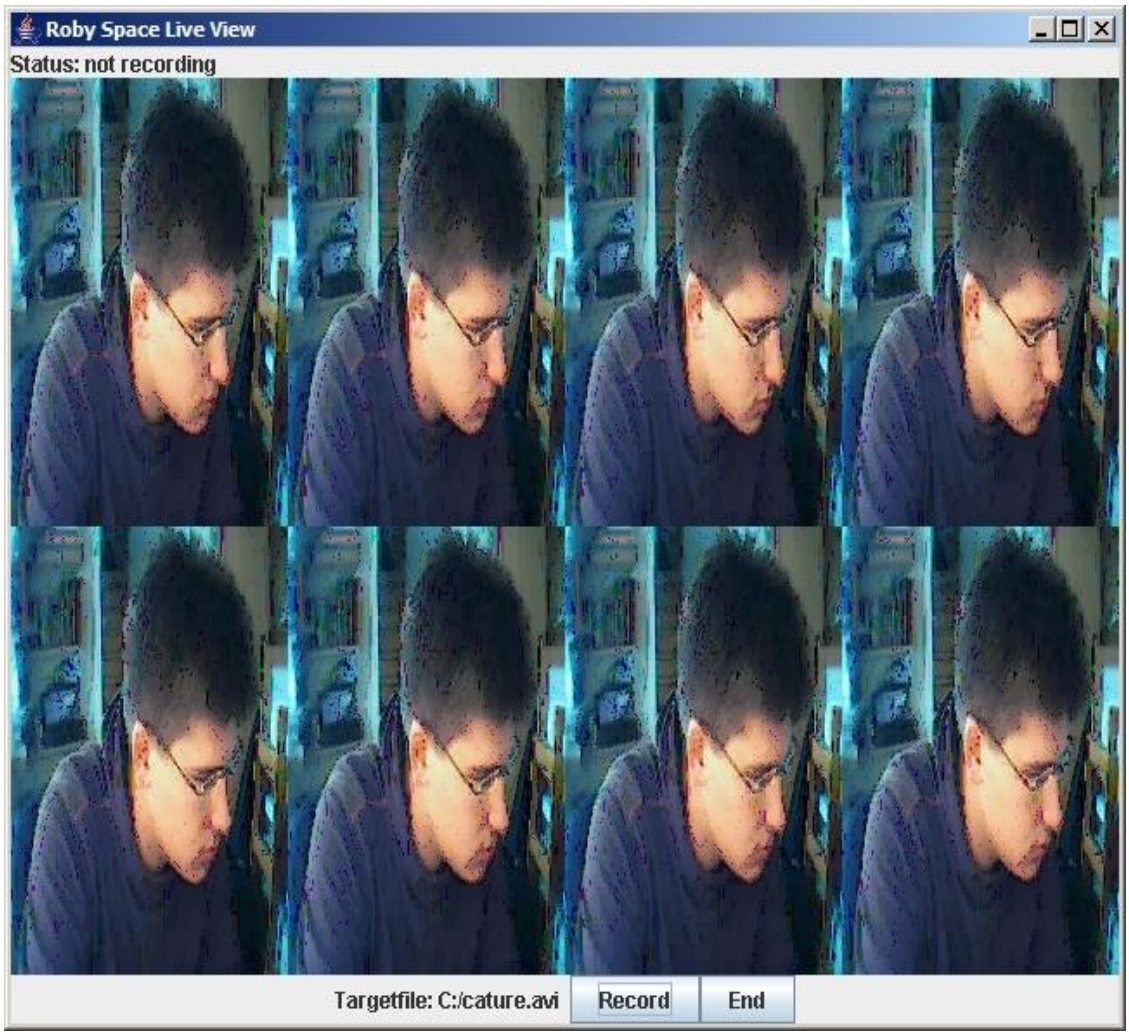

Abbildung 6-3 - Viewer

6.2.3 RS_Transformer

Bei der Instanziierung eines Objekts vom Typ RS_Transformer werden eine DataSource sowie einige Informationen zum gewünschten Videoformat übergeben. Diese DataSource dient als Quelle für einen Prozessor, den der Transformator erstellt. Im Prozessor wird der Eingangs-Stream in einen oder mehrere Tracks aufgeteilt, je nachdem aus wie vielen Tracks das Video besteht. Sobald der Prozessor sich im Zustand „Configured“ befindet, kann nun mit Hilfe der Track-Controller das Format der einzelnen Tracks angepasst werden. Der Transformer sucht sich dazu den Track-Controller des Video-Tracks heraus und setzt das Format neu. Das hat zur Folge, dass der Prozessor, während er sich im Zustand „Started“ befindet, den Track umforma-

tiert. Der Transformator gibt dem Prozessor den Befehl in den Zustand „Prefetched“ zu wechseln, worauf die Initialisierung abgeschlossen ist.

Der Transformer stellt weiters Funktionen wie „start“, „stop“ und „getDataOutput“ zur Verfügung, die den jeweiligen Befehl an den Prozessor weiterreichen, sowie gegebenenfalls auch das Ergebnis zurückreichen. Damit ist es nun möglich den Prozessor zu steuern. Außerdem ist eine Funktion implementiert, um einen Actionlistener an den Transformer zu übergeben, der im Falle, dass der Video-Stream zu Ende ist, aktiviert wird.

6.2.4 RS_DataSourceWrapper

Diese Klasse erweitert die Klasse PushBufferDataSource und implementiert das CaptureDevice Interface. Bei der Instanziierung der Klasse wird eine Datenquelle angegeben. Der DataSourceWrapper leitet den Stream dieser Datenquelle nun durch den RS_StreamWrapper, der als innere Klasse realisiert ist. Die Ausgabe des RS_StreamWrappers wird, wenn der Stream des RS_DataSourceWrappers abgefragt wird, zurückgegeben. Ansonsten werden die Funktionsaufrufe an die ursprüngliche Quelle weitergeleitet.

Im RS_StreamWrapper, der einen PushBufferStream sowie einen BufferTransferHandler implementiert, geschieht nun die eigentliche Bearbeitung. Bei der Initialisierung wird ein neuer Puffer mit der Dimension 1280 zu 480 erzeugt. Sobald die Quelle einen neuen Puffer, der jeweils ein Bild enthält, bereit stellt, wird die transferData Funktion ausgelöst. Mittels Zeichenfunktionen wird nun das Bild in einen Bereich des neuen Puffers kopiert. Sobald die transfer-Datafunktion acht Mal aufgerufen wurde, also das komplette Bild im Puffer neu gezeichnet wurde, wird der Transferhandler des Objekts, welches den RS_DataSourceWrapper als Quelle verwendet, aufgerufen. Dadurch wird die Framerate auf ein Achtel reduziert.

6.2.5 RS_processor

Der RS_processor erzeugt einen Prozessor und bindet in die Verarbeitungskette des Prozessors den RS_DivideCodec ein. Dazu wird, sobald der Prozessor den Zustand „Configured“ erreicht, mit Hilfe des TrackControls der erste VideoTrack herausgesucht und in dessen CodecChain der RS_Divide_Codec eingefügt.

Diese Klasse stellt außerdem eine Reihe von Funktionen zur Verfügung mit denen der Prozessor gesteuert werden kann. Neben den üblichen Funktionen wie „Start“, „Stop“, „Zurückspu-

len“ und Ähnlichem wird beispielsweise auch ein Statusbar zur Verfügung gestellt, der, falls die Dauer des Videos bekannt ist, anzeigt, wie weit das Video bereits abgespielt ist. Dieser Statusbar wird durch einen eigenen Thread im Hintergrund laufend aktualisiert. Ebenso wird ein Label erzeugt, auf dem Informationen, wie etwa die Art der Codierung, angezeigt werden.

6.2.6 RS_Divide_Codec

Der RS_Divide_Codec implementiert das Interface „Codec“. Der Codec hat die Aufgabe nur jedes n-te Bild wiederzugeben. Sowohl bei der Instanziierung als auch mittels einer Funktion kann ein entsprechendes Intervall angegeben werden. Bei einem Intervall von vier wird zum Beispiel das erste Bild im Video vier Mal ausgegeben, danach das fünfte Bild vier Mal und so weiter. Mittels eines zweiten Wertes wird nun noch ein Offset angegeben.

Sobald ein neues Bild des Videos eintrifft, wird die Funktion „process“ aufgerufen, wobei ein Eingangs und ein Ausgangs- Puffer übergeben werden. Anhand der erwähnten Parameter wird nun entschieden, ob der neue Eingangs-Puffer in einen internen Puffer kopiert wird oder nicht. In den Ausgangs-Puffer wird immer der innere Puffer kopiert. So lässt sich auf einfache Weise die gewünschte Funktionalität realisieren. Anzumerken ist noch, dass wenn ein Intervall von Eins gewählt wird, der Codec keine Auswirkungen auf die Ausgabe hat.

6.2.7 RS_Editor

Wie bereits erwähnt können mit Hilfe des Editors die Signale der einzelnen Kameras aus dem Videostream wieder extrahiert werden. In Abbildung 6-4 ist das Interface des Editors zu sehen, welches unmittelbar nach dem Start des Programmes erzeugt wird.

Über die Menüleiste ist es möglich, das Video-File zu öffnen, aus dem das Video einer Kamera extrahiert werden soll. Nachdem ein File ausgewählt wurde, wird mit Hilfe des RS-Transformers das Video in ein MJPEG Format [App07] konvertiert und temporär gespeichert. Der Grund dafür ist der, dass zwar der RS_Viewer ein aufgenommenes Video standardmäßig als MJPEG speichert, allerdings soll es auch möglich sein, Videos, die mit Hilfe einer anderen Software, die eventuell MJPEG nicht unterstützt, aufgenommen wurden, zu bearbeiten. Durch diese Konvertierung kann für den Rest des Programms davon ausgegangen werden, dass das Video immer in diesem Format vorliegt. Als MJPEG werden informell Videoformate bezeichnet, bei denen jedes einzelne Bild im JPEG Format codiert ist. Man kann sich das Video-Format ähnlich wie ein MPEG Format vorstellen, das allerdings nur aus I-Frames besteht. Dieses For-

mat eignet sich vor allem deswegen besonders gut zur Speicherung des Videos, aus dem die einzelnen Streams noch nicht extrahiert sind, da in diesem Video, im Gegensatz zu vielen anderen Videos, zwei benachbarte Frames Inhaltlich nicht voneinander abhängig sind, es also nicht möglich ist, aus einem Frame den nachfolgenden Frame bis zu einem gewissen Grad herzuleiten.

Abbildung 6-4 - Editor

Aus dem neu erzeugten File wird nun eine neue, klonbare Datenquelle erstellt. Weiters werden zwei RS_Prozessoren erstellt, die jeweils einen Klon als Quelle erhalten. Bei einem Prozessor wird das Intervall für den RS_Codec fix auf Eins gesetzt, wodurch das Video unverändert bleibt. Dieser Prozessor dient dazu, das Original Video anzuzeigen, welches im Interface auf der linken Seite zu sehen ist. Weiters wird von diesem Prozessor das Informations-Label angezeigt, welches im Interface über der Ausgabe des Videos zu sehen ist. Auch der Statusbalken stammt von diesem Prozessor.

Die Ausgabe des zweiten Prozessors wird auf der rechten Seite angezeigt. Die Parameter des Codecs von diesem Prozessor lassen sich durch die zwei Eingabefelder variieren. Dadurch hat man die Möglichkeit, sich den Videostream, der von einer Kamera geliefert wurde, anzusehen. Mit Hilfe des Menüs kann man den so extrahierten Videostream auch speichern. Nach Auswahl des Files, in dem das Video gespeichert werden soll, wird ein weiterer RS_Prozessor erzeugt, der, auf Basis der momentan im Interface angegebenen Parameter, einen einzelnen Videostream extrahiert und speichert. Sobald dieser Vorgang beendet ist, kann über das Interface

ein anderer Stream ausgewählt, oder das Programm beendet werden, wobei dabei auch das temporäre File gelöscht wird.

6.2.8 RS_Listener

Hierbei handelt es sich um ein Interface, welches die Funktion „RS_Action“ definiert. Klassen, die dieses Interface implementieren, können als Actionlistener an den Transformer übergeben werden, der gegebenenfalls die Funktion „RS_Action“ auslöst.

7 Fazit und Ausblick

Ziel der Arbeit war es, ein VideoSystem für die Roboter, die im Rahmen der Furushiki2 Projekts verwendet werden sollen, zu entwerfen. Dabei war vor allem auf die besonderen Bedingungen Rücksicht zu nehmen. Nach einem Vergleich verschiedener Panoramasysteme fiel die Wahl auf ein System mit vier Kameras, da dieses System so in den Roboter eingebaut werden kann, dass keine weiteren Teile herausragen.

Da die Signale der einzelnen Kameras nicht direkt von den Robotern zur Erde übertragen werden können, wurde beschlossen, die Rakete als Relais zu verwenden. Da diese allerdings nur ein analoges Videosignal senden kann und es auch nicht praktikabel scheint, jedes der vier Videosignale eines Roboters separat zur Rakete zu senden, wurde ein Videocontroller entwickelt, der die Videosignale zusammen fasst. Dieser arbeitet nach dem Time Division Multiplexing Prinzip und wurde in Hardware auf einem FPGA realisiert. Dieser Videokontroller zeichnet sich vor allem dadurch aus, dass er auch Videosignale die nicht synchronisiert sind, zu einem neuen Videosignal, ohne Fehler beim Umschalten zu verursachen, vermischen kann. In jeden Roboter wird einer dieser Controller eingebaut, so dass nur ein einzelnes Signal zur Rakete gesendet werden muss. Ein weiterer Controller auf der Rakete fasst, falls zwei Roboter verwendet werden, diese Signale nochmals zusammen.

Um den Videostrom wieder aufzuteilen wurde eine Software entwickelt, die das Videosignal direkt von einer Framegrabberkarte ausliest und die einzelnen Signale wieder separat anzeigt. Außerdem können auch gespeicherte Videos mit dieser Software aufgetrennt werden, so dass auch eine professionelle Capture Software verwendet werden kann.

Zum Abschluss noch ein paar Gedanken über mögliche Verbesserungen: Der entwickelte Controller nützt zwar, was die Übertragung des Videosignals angeht, den Übertragungskanal so gut wie möglich aus, trotzdem gibt es noch versteckte freie Kapazitäten. Wie in Kapitel 4.1 beschrieben liegt zwischen den Halbbildern jeweils eine Vertikalaustastlücke mit einer Größe von 25 Zeilen. Auch wenn in diesen Zeilen in einem Standard FBAS-Signal keine Bildinformationen übertragen werden können, können sie zur Übertragung anderer Informationen genutzt werden, wie dies auch in der Praxis oft der Fall ist. [Schmi03] So werden beim analogen Fernsehen einige Zeilen genutzt um den Teletext zu übertragen. Andere Zeilen werden als Prüfzeilen verwendet und dienen damit messtechnischen Zwecken. Ebenso werden die Video Programm System Informationen sowie der Vertical Interval Timecode in einigen Zeilen übertragen.

Interessant wäre es nun den Videocontroller so zu erweitern, dass er auch diese Kapazitäten nützt, um etwa Telemetriedaten zu übertragen, die im aktuellen Konzept ja auf einem anderen Kanal übertragen werden. Natürlich müssten Überlegungen angestellt werden, wie dies am Besten zu realisieren ist. Es gibt beispielsweise schon Digital-Analogwandler für Videosignale die automatisch Teletext einbetten können, wodurch der Entwicklungsaufwand entsprechend gering wäre. Es könnte aber auch der Videocontroller selbst die Daten in das Video einbetten, wobei natürlich nicht das Teletext Format verwendet werden muss. Allerdings ist es wahrscheinlich dann nicht möglich, die Daten mit Hilfe einer Standard-Framegrabberkarte wieder zu extrahieren.

Es wäre auch denkbar, nicht nur Telemetriedaten zu senden, sondern Steuerbefehle von den Robotern an den Videocontroller in der Rakete zu senden. Wenn zum Beispiel ein Roboter erkennt, dass er sich verheddert hat und er trotz einiger Versuche nicht mehr los kommt, könnte in diesem Fall ein Befehl an den Videocontroller in der Rakete gesendet werden, so dass dieser nur mehr den Videostream vom anderen, funktionierenden Roboter überträgt.

Auch wenn dieses System im Hinblick auf das Furoshiki2 Projekt entwickelt wurde, wäre es denkbar, dieses Videosystem, zumindest teilweise, auch für andere Projekte zu nutzen.

8 Literaturverzeichnis

[App07] Apple, Inc (2007): *QuickTime File Format Specification.* Online: http://developer.apple.com/documentation/QuickTime/QTFF/qtff.pdf. (Stand: 12.9. 2007).

[ARR04] Andoya Rocket Range (2004): *ALOMAR eARI.* Online: http://alomar.rocketrange.no/eARI/ALOMAR_eARI_17122003.html (Stand: 3.3.2007)

[ARR06] Andoya Rocket Range (2006): *Configuration Control HotPay2* (Stand: 2.6.2006)

[Ben01] Benosman, Ryad; Kang, Sing B. (2001): *Panoramic Vision*. New York: Springer-Verlag.

[Bru00] Bruckstein ,Alfred M.; Richardson, Thomas J. (2000): *Omniview Cameras with Curved SurfaceMirrors*. Proceedings of the IEEE Workshop on Omnidirectional Vision (OMNIVIS'00), pp.79-84.

[Cha97] Charles, Jeffrey R. (1997): *Converting Panoramas to Circular Images and Vice Versa - Without a Computer.* Online: http://www.eclipsechaser.com/eclink/astrotec/panconv.htm (Stand: 17.9.2007).

[CCIR86] CCIR (Hrsg.) (1986): *Recommendation 656. Interfaces for digital component video Signals in 525-line and 625-line Television systems.*

[Cut02] Cutler, Ross; Rui, Yong; u.a. (2002): *Distributed meetings: a meeting capture and broadcasting system.* Redmond: Microsoft Research.

[Dal98] Dally, William J.; Poulton, John W. (1998): *Digital Systems Engineering*. New York: Cambridge University Press.

[Eid04] Eidenberger, Horst; Divotkey, Roman (2004): *Medienverarbeitung in Java.* dpunkt Verlag.

[Eub06] eubus GmbH (2006): *hydraXC.* Online: http://www.hydraxc.com/ (Stand: 16.7.2007).

[Foo00] Foote, Jonathan; Kimber, Don (2000): *FlyCam: Practical Panoramic Video and Automatic Camera Control.* Proceedings of IEEE International Conference on Multimedia and Expo, Vol III, pp. 1419-1422.

[Gas02] Gasiler, Jiri (2002): *Fault Tolerant Microprocessors for Space applications.* International Conference on Dependable Systems and Networks, pp. 41-50.

[Gin03] Ginosar, Ran (2003): *Fourteen Ways to Fool Your Synchronizer.* IEEE Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC).

[Hor00] Horton, Ivor (2000): *Beginning Java 2.* Birmingham: Worx Press Ltd.

[Ish98] Ishiguro, Hiroshi (1998): *Development of Low-Cost Compact Omnidirectional Vision Sensors and their applications.* Proceedings of International Conference on Information Systems Information systems, analysis and synthesis, pp.433-439.

[ITU95] ITU (Hrsg.) (1995): *Recommendation ITU-R BT.601.5*.

[Jac04] Jacobs, Corinna (2004): *Digitale Panoramen - Tipps, Tricks und Techniken für die Panoramafotografie*. Berlin: Springer-Verlag.

[Kay04] Kaya, Nobuyuki, et al. (2004): *Crawling robots on large web in rocket experiment on furoshiki deployment.* Vancouver: 55th International Astronautical Congress 2004.

[Kim01] Kimber, Don; Foote, Jonathan; Lertsithichai, Surapong (2001): *FlyAbout: Spatially Indexed Panoramic Video.* Proceedings of ACM Multimedia, pp. 339-347.

[Kop97] Kopetz, Hermann (1997): *Real-Time Systems. Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers

[Kum00] Kumler, James; Bauer, Martin (2000): *Fish-Eye lens designs and their relative performance*. SPIE Proceedings-Current Developments in Lens Design and Optical Systems Engineering, pp.360-369.

[Nay97] Nayar, Shree K. (1997): *Catadioptric Omnidirectional Camera.* 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97).

[Nay98] Nayar, Shree K.; Boult, Terry (1998): *Omnidirectional Vision Systems: 1998 PI Report.* Proceedings of the DARPA Image Understanding Workshop (IUW), pp.93-99.

[Ova] Ovation Systems Ltd.: *FourSight: real-time advanced video quad screen splitter / combine.* Online: http://www.ovation.co.uk/Quad.html (Stand: 17.9.2007).

[RI07] The Robotics Institute at Carnegie Mellon University (2007): *The CMUcam Vision Sensors.* Online: http://www.cs.cmu.edu/~cmucam (Stand: 18.9.2007).

[Sam02] Samsung Electronics (Hrsg) (2002): *K4S283233F-M Datasheet.*

[Schil03] Schild, Gerhard H.; Redlein, Alexander; Kahn, Daniela (2003): *Einführung in die Technische Informatik.* Wien: Springer-Verlag.

[Schmi03] Schmidt, Ulrich. (2003): *Professionelle Videotechnik.* Berlin: Springer-Verlag.

[Sea00] Seale Studio (2000): *e-Pan - The Digital Panoramic Camera for Virtual Reality.* Online: http://www.epan.net (Stand: 17.9.2007).

[Sum06] Summerer, Leopold, et al. (2006): *First Results of Robots Crawling on a Loose Net in Micro-gravity during a Sounding Rocket Experiment*. Valencia: 57th International Astronautical Congress.

[Sun99] Sun Microsystems, Inc. (1999): *Java Media Framework API Guide.* Online: http://java.sun.com/products/java-media/jmf/2.1.1/guide/index.html (Stand: 12.9.2007)

[Xil07a] Xilinx, Inc. (2007): *Xilinx: The Programmable Logic Company.* Online: http://www.xilinx.com/ (Stand: 19.9.2007).

[Xil07b] Xilinx, Inc. (2007): *Xilinx : ISE WebPACK.* Online: http://www.xilinx.com/ise/logic_design_prod/webpack.htm (Stand: 19.9.2007).

[Xil07c] Xilinx, Inc. (2007): *Xilinx : ModelSim Xilinx Edition-III.* Online: http://www.xilinx.com/ise/optional_prod/mxe.htm (Stand: 19.9.2007).

[Xil07d] Xilinx, Inc. (2007): *Virtex-4 Data Sheet: DC and Switching Characteristics.* Online: http://direct.xilinx.com/bvdocs/publications/ds302.pdf (Stand: 17.9.2007).

[Xil07e] Xilinx, Inc. (2007): *Virtex-4 User Guide.* Online: http://direct.xilinx.com/bvdocs/userguides/ug070.pdf (Stand: 18.9.2007).

[Xil07f] Xilinx, Inc. (2007): *Answer Record # 23624: Virtex-4 DCM - DCM does not lock when DFS outputs are used and input clock is outside of DLL output range.* Online: http://www.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=23624 (Stand: 18.9.2007).

9 Abkürzungsverzeichnis

BAS	Bild-Austast-Synchron-Signal
CCIR	Comité Consultatif International des Radiocommunication
CMOS	Complementary Metal Oxide Semiconductor
CVBS	Colour Video Blanking Signal
DCM	Digital Clock Manager
DFS	Digital Frequency Synthesizer
FBAS	Farb-Bild-Austast-Synchron-Signal
FIFO	First In First Out
FPGA	field-programmable gate array
ITU	International Telecommunication Union
JMF	Java Media Framework
JPEG	Joint Photographic Experts Group
LED	Light Emitting Diode
MJPEG	Motion JPEG
MPEG	Moving Picture Experts Group
NRZ	Non Return to Zero
RAM	Random Access Memory
RS-232	Recommended Standard 232
SDRAM	Synchronous Dynamic Random Access Memory
SDR-SDRAM	Single Data Rate -Synchronous Dynamic Random Access Memory
VHDL	High Speed Integrated Circuit Hardware Description Language

A Anhang

A.1 Verbindung CMUCam und HydraXC