Technische Universität Clausthal
Institut für Informatik

Studienarbeit

Testen und bearbeiten

der Spartan3-Platinen

CamA und MonA

Michael Gremmel, Matrikel-Nr.: 344441

Abgabetermin: 16.09.2010

Betreuer und Gutachter:

Prof. Dr. Günter Kemnitz, Dipl. Ing. Carsten Giesemann

Versicherung zur selbstständigen Arbeit

Ich versichere, dass ich diese Arbeit ohne unzulässige fremde Hilfe und nur unter Benutzung der in der Arbeit angegebenen Literatur und sonstigen Hilfsmittel angefertigt habe.

_____

Michael Gremmel

Inhaltsverzeichnis

1. Aufgabenstellung

Diese Studienarbeit befasst sich mit zwei an dem Institut für Informatik - Bereich Hardwareentwurf und Robotik - der TU-Clausthal entworfenen Modulen für das SPARTAN-3 Starter Kit Board der Firma XILINX. Dieses Evaluation Bord verfügt über einen FPGA (XC3S1000), der mit einem Takt von 50MHz betrieben wird. Auf dem Board sind diverse Switches und Buttons zur Eingabe, sowie einige Schnittstellen (8-Farben VGA, RS232 und PS/2) zu finden. Zur Anzeige von Programmzuständen können LEDs und eine vierstellige 7-Segmentanzeige herangezogen werden. Ein Flash-Speicher erlaubt es, Daten zu speichern, die den Programmcode enthalten. Optional können zwei asynchrone SRAM-Module bedient werden. Daneben ist es möglich, das Board über drei Erweiterungs-Ports mit zusätzlichen Modulen zu bestücken.

Einer dieser Ports wird mit dem Modul „MonA“ (Monitor Adapter) belegt, es hat eine VGA-Schnittstelle, die über einen Digital-Analog Konverter von Texas Instruments (THS8134B) angesprochen wird. Dieser Adapter soll hinsichtlich seiner Funktionalität getestet werden. D. h. mit diesem Modul als Schnittstelle zwischen FPGA und Bildschirm soll auf einem Monitor ein VGA-Bild (640 x 480 Pixel) mit 256 Farbabstufungen je Kanal (rot, grün und blau) erzeugt werden. Der Ausgangspunkt sind die Datenblätter des Digital-Analog Konverters und eine Übung aus dem Softwaretechnikpraktikum (eine Aufgabenstellung zur Ansteuerung der 8-Farben VGA-Schnittstelle des Spartan-Boards), sowie eine UCF-Datei, die die Pin-Konfiguration des Adapters festlegt.

Es sind hierfür drei Arbeitsschritte vorgesehen, die in VHDL realisiert werden sollen:

- Erstellen eines Testprogramms zur Messung der VGA-Steuer- und Synchronisationssignale.
- Einfarbige Ausgabe des Bildes (Rot, Grün, Blau, Gelb, Cyan und Magenta), zum Umschalten sollen die Switches des Spartanboards genutzt werden.
- Erzeugung verschiedener Testbilder (Farbstreifen, Farbverläufe etc.)

Ziel der Aufgabenstellung zu dem Monitor-Adapter ist es, kleine, möglichst übersichtliche und selbsterklärende Testprogramme zu entwerfen.

Ein zweiter Port soll mit dem CamA (Camera Adapter) bestückt werden. Dieses Modul ist mit zwei Handy-Kameras des Typs OV7725 von Omnivison ausgestattet. Über eine Kamera soll ein möglichst flüssiges und farbechtes Bild aufgenommen und über den Monitor-Adapter an einem

Bildschirm ausgegeben werden. Die zweite Kamera soll mit einem Infrarotfilter (Filterscheibe vor der Kamera) versehen werden, über sie sollen blinkende Infrarot-LEDs (am dritten Zusatz-Port angeschlossen) detektiert und je nach Frequenz unterschieden werden. Die Position der jeweiligen LED ist auf einem Monitor durch ein Icon darzustellen. Diese Funktion ist eine Teilaufgabe eines Projektes, das vom Institut für Informatik geleitet wird.

Im Groben handelt es sich dabei um mehrere Roboter, die auf einem beschränkten Feld selbstständig gegeneinander Fußball spielen sollen. Um die Roboter lokalisieren und deren Richtung bestimmen zu können, sollen auf ihnen zwei Infrarot-LEDs montiert werden, die für jeden Roboter in einer unterschiedlichen Frequenz blinkt. Über das Kamera-Modul soll die Blinkfrequenz aufgezeichnet werden und über den FPGA die Frequenz einem Roboter zugeordnet werden. Über ein Funkmodul soll dann der Roboter Instruktionen erhalten, wie er sich auf dem Spielfeld zu verhalten hat.

Zur Implementierung der LED-Detektion soll in dieser Arbeit angenommen werden, dass zwei Roboter je eine LED besitzen. D.h. es wird nur die Position einer LED bestimmt, nicht die Richtung eines Roboters.

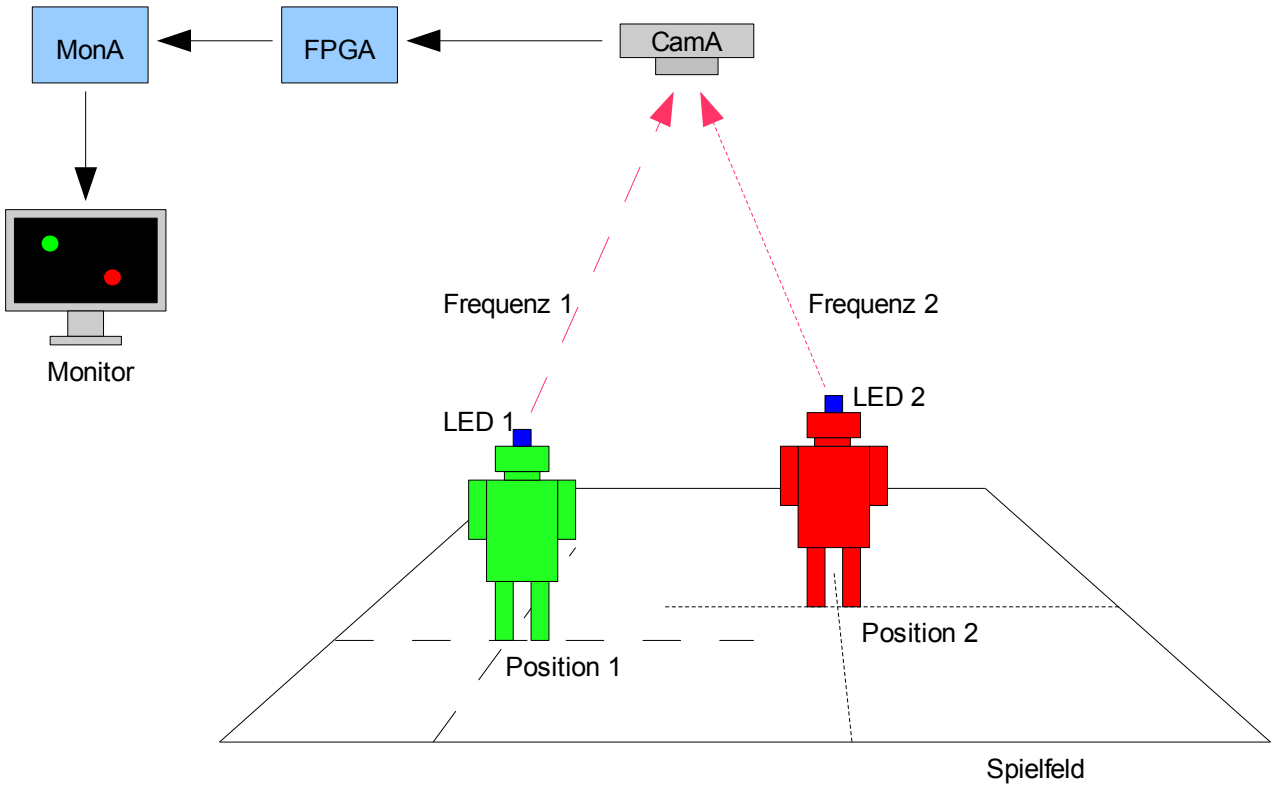

Abbildung 1: Projektaufbau CamA und MonA

Die Ausgangsposition zur Bearbeitung des Camera Adapters ist folgende: Es existiert ein Prototyp des Moduls mit einer Kamera, zu dem ein bereits zum Teil implementierter VHDL-Code bezüglich Kommunikation mit der Kamera und Bildausgabe existiert. Daher ergeben sich zu CamA diese Arbeitsschritte:

- Kommentieren des bisher unkommentierten Codes.
- Erläuterung der Funktionsweise des Codes hinsichtlich Initialisierung und Bilderfassung der Kamera und Verarbeitung der gelieferten Bilddaten bis zur Ausgabe auf dem Monitor.
- Beseitigen der Fehler im Code, um ein stabiles Programmverhalten zu erreichen und ein möglichst flüssiges Bild (maximale Bildwiederholrate) auszugeben.
- Anpassen des Codes und der UCF-Datei (Pin-Ansteuerung), damit statt des Prototypenadapters ein Kamera-Modul betrieben werden kann, das den Betrieb von zwei Kameras erlaubt. Die betreffende Platine ist bereits vorhanden.
- Erweiterung des VHDL-Codes um die oben beschriebene Funktion zur Detektion zweier mit unterschiedlicher Frequenz blinkenden LEDs.

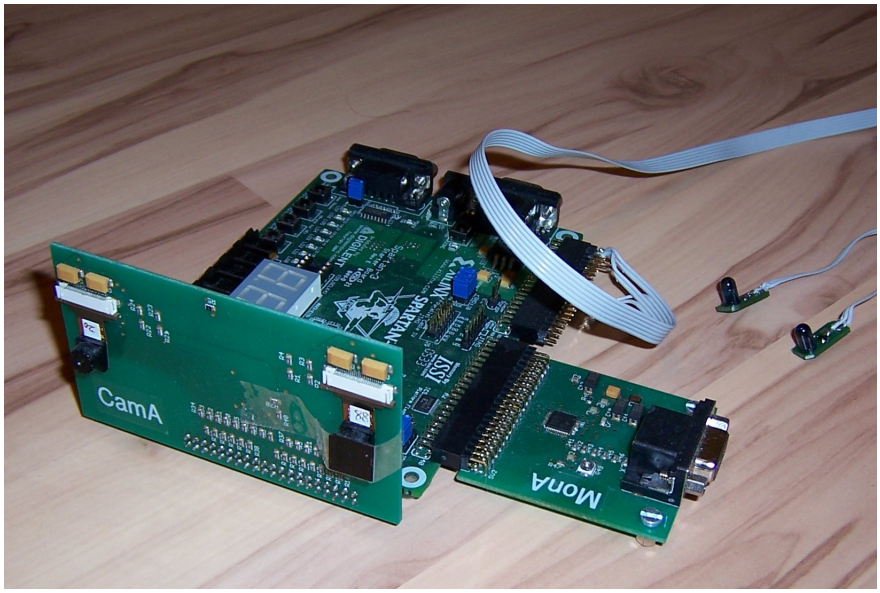

Abbildung 2: Versuchsgruppe Evaluationsboard mit CamA und MonA

2. Monitor-Adapter

Der Monitor-Adapter („MonA“) verfügt über eine VGA-Schnittstelle, wie das Spartan 3 Board selbst. Der Unterschied in der Funktionalität liegt darin, dass über den VGA-Port des Boards pro Farbkanal (Rot, Grün, Blau) nur ein Bit vorgesehen ist. Das heißt, entweder ist ein Kanal aktiviert oder deaktiviert. Damit ergibt sich eine maximale Anzahl an unterschiedlich darstellbaren Farben von $2^3 = 8$. Auf dem Monitor-Adapter ist ein Triple 8-Bit D/A-Konverter (THS8134 von Texas Instruments) verbaut, der es erlaubt, pro Kanal 256 Farbabstufungen (hell zu dunkel) anzeigen zu lassen. Damit kann ein viel größerer Farbraum abgedeckt werden, nämlich $(2^8)^3 = 16777216$ unterschiedliche Farbzustände. Für gewöhnlich wird der THS8134 für hochauflösende Bildwiedergabe verwendet (z.B. HDTV), für dieses Projekt wird jedoch nur die Funktion des größeren Farbraumes benutzt. Alle weiteren Funktionen werden deaktiviert, bzw. auf den Standardeinstellungen belassen. Der Grund liegt darin, dass für die Video-Aufnahme über die Handy-Kamera (siehe zweite Aufgabenstellung) lediglich ein Bild mit natürlicher Farbwiedergabe in der VGA-Auflösung von 640 x 480 Pixel benötigt wird.

2.1 Ansteuerung der VGA-Schnittstelle

Die VGA-Schnittstelle ist ursprünglich für die Ansteuerung eines Röhrenmonitors vorgesehen. D. h. nach dem Prinzip der Braunschen Röhre werden für jeden Farbkanal von einer Kathode Elektroden ausgedampft, die mittels eines Wehneltzylinders zu einem Strahl gebündelt und über eine Anode Richtung Bildschirm beschleunigt werden. Der Strahl durchläuft dabei Kondensatoren zur horizontalen und vertikalen Ablenkung, bevor er auf auf den Bildschirm trifft und dort einen fluoreszierenden Punkt ergibt. Vom Betrachter aus gesehen verläuft der Elektronenstrahl pro Zeile von links nach rechts, von oben links beginnend. Wenn der Strahl in die nächste Zeile springen muss, wird der Kathodenstrom abgeschaltet, der Rücklauf zum nächsten Zeilenanfang ist also nicht sichtbar.

Ein Bild besteht aus 640x480 Pixel, der Elektronenstrahl beginnt bei Pixel (1,1) und durchläuft den Bildschirm zeilenweise bis Pixel(640,480). Nach jeder Zeile erfolgt die horizontale Synchronisation, nach einem gesamten Bilddurchlauf die vertikale Synchronisation.

Um das eigentliche Bild befindet sich ein Schwarzbereich, in dem kein Bild dargestellt wird. Dieser Schwarzbereich zur Dunkeltastung muss bei der Berechnung des Pixeltaktes berücksichtigt werden.

Zur Darstellung eines Bildes in VGA-Auflösung auf einem Röhren- oder TFT-Monitor ist eine horizontale Bildfrequenz von 31250Hz und eine vertikale Frequenz von 60Hz gefordert. Eine Bildzeile ist also in 32µs durchlaufen, diese Zeitspanne ist in vier Bereiche unterteilt:

- t_1 (horizontale Synchronisation) : 3,84µs
- t_2 (Bildvorlauf) : 1,92µs
- t_3 (Bilddarstelleung) : 21,76µs
- t_4 (Bildnachlauf) : 4,48µs

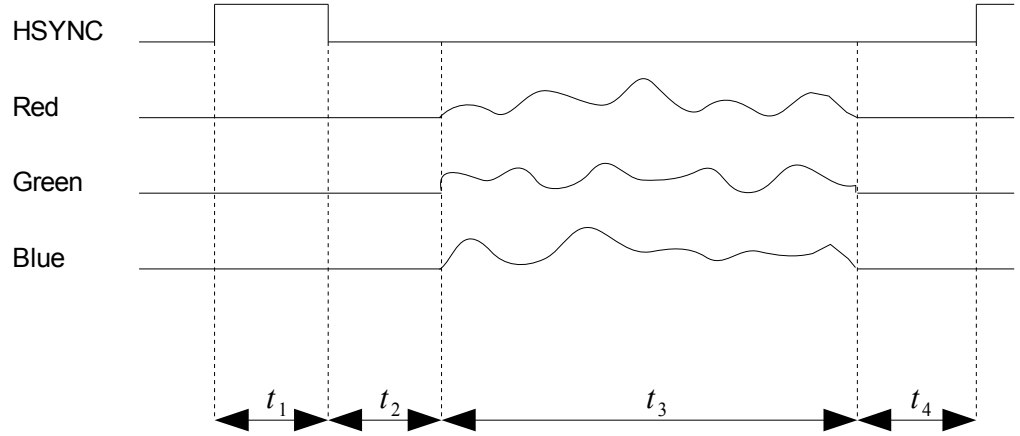

Abbildung 3: MonA horizontale Synchrnisation

In der Zeit t_3 sollen 640 Pixel dargestellt werden, daher entsprechen t_1 96 Pixel, t_2 48 Pixel und t_4 16 Pixel, die den Schwarzbereich bilden. Insgesamt müssen also Daten für 800 Pixel in 32µs in horizontaler Richtung an die VGA-Schnittstelle gesendet werden. Mit 3,84µs pro 96 Pixel liegt der Pixeltakt bei 25MHz.

Für die vertikale Ansteuerung ergibt sich entsprechend:

- t_1 (vertikale Synchronisation) : 64µs / 2 Zeilen
- t_2 (Bildvorlauf) : 928µs / 29 Zeilen
- t_3 (Bilddarstelleung) : 15360µs / 480 Zeilen
- t_4 (Bildnachlauf) : 348µs / 10 Zeilen

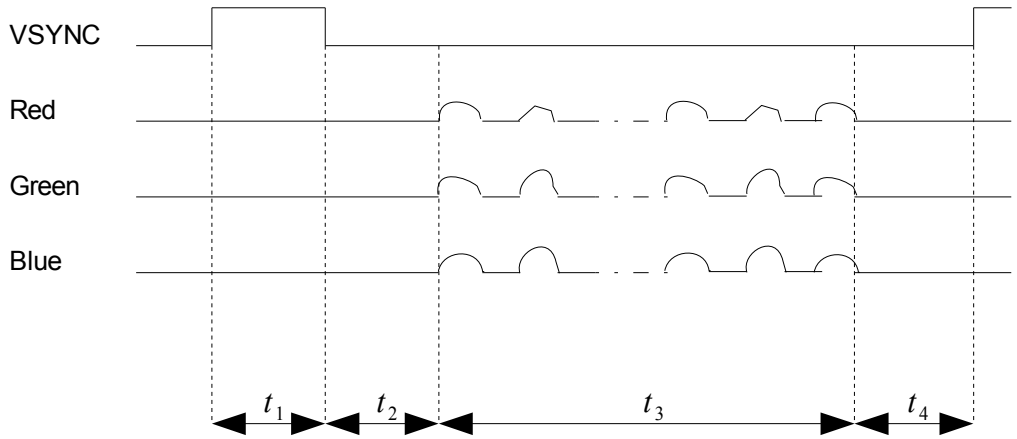

Abbildung 4: MonA vertikale Synchronisation

In vertikaler Richtung werden somit 521 Zeilen dargestellt.

Die beiden Synchronisationssignale werden direkt vom FPGA ausgegeben, die des THS8134 werden nicht genutzt:

- syncT = 0
- nsync = 1
- nblank = 0

Die Konfigurationssignale m1 und m2 werden beide mit dem Wert 0 initialisiert, laut dem Datenblatt des THS8134 wird hiermit der GBR-Modus 4:4:4 gewäht. Genutzte Eingänge sind Rpr, GY und BPb (die drei Farbkanäle zu je 8 Bit) und die Taktleitung mit 25MHz.

In Software durchläuft ein Automat horizontal 800 Pixel und springt anschließend in die nächste Zeile. Er schaltet wie im oben dargestellten Diagramm die Synchronisationssignale, Schwarzbereich und Bilddarstellung.

Für den Monitor-Adapter sind fünf Testprogramme vorgesehen. Die Demonstrationsprogramme können mit dem Button BTN(3) auf dem Evaluation Board der Reihe nach durchgeschaltet werden. Auf der Siebensegmentanzeige sind die Buchstaben „t E S“ (für „Test“) und die Nummer des aktuellen Programms dargestellt.

Für die Siebensegmentanzeige ist ein eigener Prozess (seg7process) vorgesehen, der mit einer Frequenz von ca. 380Hz die vier Segmente ansteuert. Durch diese Frequenz ist eine flimmerfreie und nicht verschwommene Anzeige gewährleistet. Ein kontinuierlich durchlaufender Automat mit vier Zuständen zeigt die Buchstaben „t E S“ und die Nummer des jeweiligen Programms an.

Um mit dem Taster von einem Programm zum nächsten schalten zu können, muss der Taster entprellt werden. Dies geschieht mit Hilfe eines dreistelligen Schieberegisters, wobei das Umschalten zum nächsten Programm durch die fallende Taktflanke (loslassen des Tasters durch den Benutzer) erkannt wird. Ist das letzte Programm erreicht, gelangt man bei erneutem Tastendruck wieder in das erste Programm.

Testprogramm 0:

Alle drei Farbkanäle sind auf schwarz geschaltet. Dieser Test dient lediglich zum Messen der Steuersignale des Monitor-Adapters. Ein Kanal ist dann dunkel (schwarz), wenn alle acht Bits 0 sind und erreicht die maximale Helligkeit der jeweiligen Farbe, wenn alle Bits 1 sind.

Testprogramm 1:

Über die Schalter (SW0 bis SW7) kann ein beliebiger achtstelliger Bitwert eingestellt werden und mit den Tastern BTN2, BTN1 und BTN0 auf die Kanäle Rot, Grün und Blau geschaltet werden. Damit lassen sich alle möglichen 2^{24} Farben darstellen.

Testprogramm 2:

Es wird ein Gitter erzeugt, dessen Farben wie in Testprogramm 2 veränderbar sind. Der Hintergrund des Gitters ist in dem invertierten Bitcode der Farbkanäle gehalten.

Testprogramm 3:

Der Monitor ist in sechs gleichgroße Bereiche unterteilt (3x2 Felder), in denen die Farben rot, grün, blau und die Sekundärfarben gelb, magenta und cyan angezeigt werden.

Testprogramm 4:

Der Monitor ist in sieben gleich breite vertikale Streifen unterteilt, in den sechs Farben wie in Testprogramm 3, zuzüglich weiß. Pro Zeile wird die jeweilige Farbe um ein Bit dekrementiert, bis der Farbbalken schwarz ist (Wert 0). Ab dann wird der Farbwert eines jeden Balkens wieder pro Zeile bitweise inkrementiert. Somit ergibt sich ein Bild mit den sieben senkrechten Balken, die im

oberen Bildbereich volle Helligkeit haben, zur Bildschirmmitte dunkler und im unteren Bildschirmbereich wieder heller werden.

Testprogramm 5:

Dieser Test enthält eine Animation, daher ist das Programm in einen eigenen Prozess ausgelagert. Dieser Prozess wird mit einer Frequenz von ca. 50Hz getaktet. Wenn der Schalter SW2 auf „on“ gestellt wird, wird der Rotkanal mit 50Hz von 0 bis 256 bitweise inkrementiert, bis er den Maximalwert erreicht. Ab 256 wird der Wert für diesen Kanal dann wieder dekrementiert, anschließend beginnt der Durchlauf von vorn. Wird der Schalter auf „off“ gestellt, hält der Durchlauf an, und der zuletzt auf den Kanal gegebene Wert wird beibehalten. Das gleiche geschieht für den Grünkanal (SW1) und den Blaukanal (SW0). Mit diesem kurzen Testprogramm lassen sich vielfältige Farbänderungen demonstrieren.

3. Kamera-Adapter

In diesem Kapitel wird erläutert, wie die Kamera des Moduls „CamA“ angesteuert wird. Die Kamera kann ein Bild von 640 x 480 Pixel (VGA-Auflösung) bei einer Bildwiederholrate von 60Hz aufnehmen. Die Initialisierung erfolgt über einen standardisierten SCCB (Serial Camera Control Bus), über den die Kamera mittels eines Python Skriptes Befehle erhalten soll. Eine Implementierung über den I²C-Bus wäre ebenso möglich und wird z.T. auch so realisiert. Die von der Kamera an den FPGA gesendeten Daten werden in diesem Projekt im Bayer-Pattern Modus ausgewertet und über den Monitor-Adapter an einen Bildschirm übergeben. Da die Daten nicht direkt zu dem Monitor-Adapter gegeben werden können, werden sie erst im S-RAM des Evaluation Boardes zwischengespeichert. Das hat den Vorteil, dass die gespeicherten Werte analysiert und ausgewertet werden können, wie es bei der Detektion von Infrarot-LEDs vonnöten ist. Über Buttons und Switches des Spartan3 Boardes sollen diverse Kameramodi aktivierbar sein.

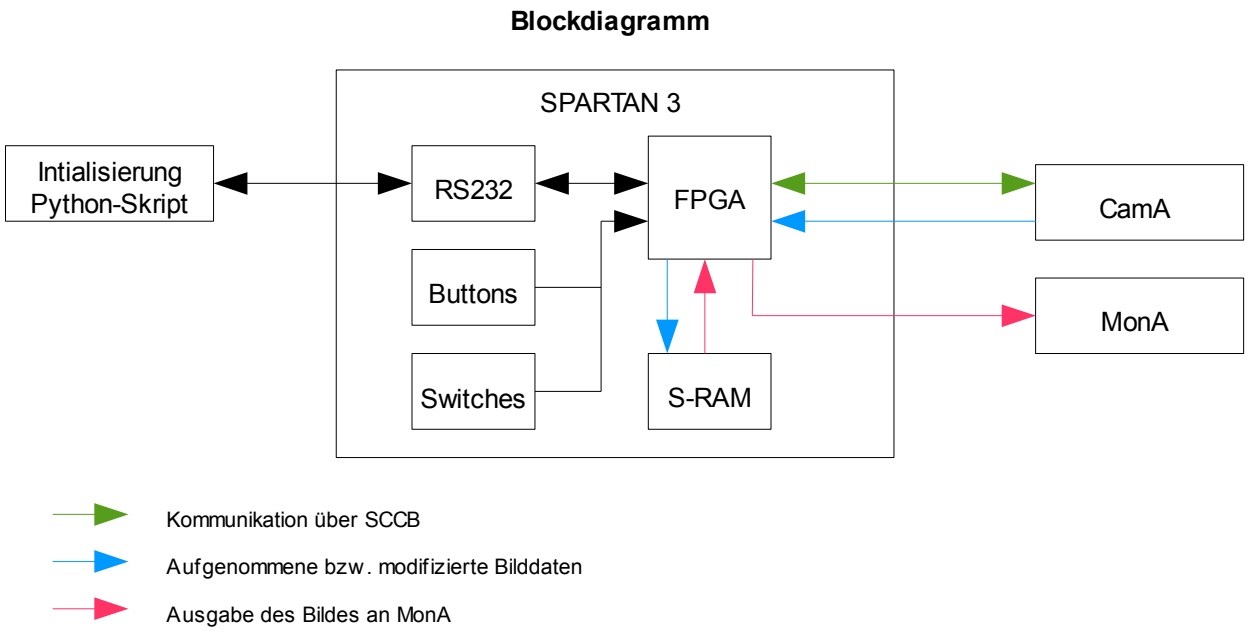

Abbildung 5: Kommunikationsstruktur zischen Spartan3 Board, CamA und MonA

Das in Xilinx ISE erstellte Projekt besteht aus folgenden Komponenten:

- I2C.vhd und SCCB.vhd beinhalten die Festlegung der Datenstruktur und Kommunikationsart mit der Kamera.
- CamControl.vhd dient zur Verarbeitung der Befehle, die zur Ansteuerung der Kamera notwendig sind. Dazu wird ein Sende- und Empfangsautomat für die RS232-Schnittstelle zur Hilfe genommen, denn darüber sollen mittels eines Python-Skriptes

Initialisierungswerte an das Spartan3-Board, respektive an die Kamera gesendet werden.

- clk2x_dcm.vhd ist ein Hilfsmodul zur Taktgenerierung.
- EdgeDetection.vhd wird zur Erkennung von fallenden und steigenden Taktflanken verschiedener Signale herangezogen.
- MemPack.vhd sollte als Speicherverwaltungseinheit genutzt werden, die allerdings in keinem Modul angewendet wird. In ihr ist im Groben festgelegt, wann welche Daten der Kamera in den S-RAM geschrieben werden und wann gelesen wird. Beides darf nicht gleichzeitig stattfinden.
- Mit Seg7Display.vhd kann die Siebensegmentanzeige des Boards angesteuert werden, es sind hierfür jedoch keine weiteren Funktionen vorgesehen. Die Ansteuerung ist sehr übersichtlich und nicht komplex, daher wird auf eine detaillierte Beschreibung verzichtet.
- Kamera.vhd enthält die Ansteuerung der VGA-Schnittstelle, Schreib- und Lesezugriffe auf den Speicher, einen Automaten zur Erkennung der LEDs sowie die Berechnung des Differenzbildeffektes.
- cam_bram.xco legt die Parameter zur Ansteuerung des Speichers fest.
- constraints_2cams.ucf beinhaltet das Portmapping der Signale, d.h. welche Signale auf welchen Pin geschaltet werden. Die Datei ist mit Hilfe des Schaltplanes dahingehend geändert, dass statt des Prototypen des Kamera-Adapters die finale Platine mit zwei Kameras angesteuert werden kann.

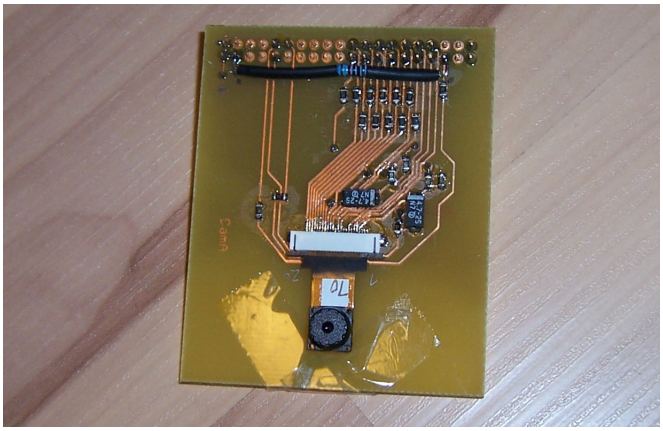

Abbildung 6: CamA Prototyp mit einer Kamera

3.1 Kommunikationsstruktur

Der Zugriff auf die Kamera erfolgt durch das SCCB-Interface (SCCB.vhd), das wiederum auf eine I²C Implementierung aufsetzt (I2C.vhd). Letztere beinhaltet nur die Kommunikation vom Master aus, der Slave kann also die Übertragungsgeschwindigkeit des Masters nicht beeinflussen. Dies geschieht im Normalfall dann, wenn der Master zu schnell sendet und der Slave die Daten nicht mehr verarbeiten kann. Der Slave kann in diesem Fall den Master „bremsen“. Der Master ist bei diesem Projekt der FPGA und der Slave die Kamera. Auf eine Erläuterung des I²C-Busses wird in dieser Studienarbeit nicht weiter eingegangen, dazu kann die Quelle [I2CBUSPH] im Literaturverzeichnis herangezogen werden.

3.1.1 Modul I2C.vhd

Es wird für den I2C-Bus eine Taktleitung und eine Datenleitung festgelegt. Die Funktionen von **I2C.vhd** beschreiben das Verhalten der Signalverläufe der beiden Leitungen.

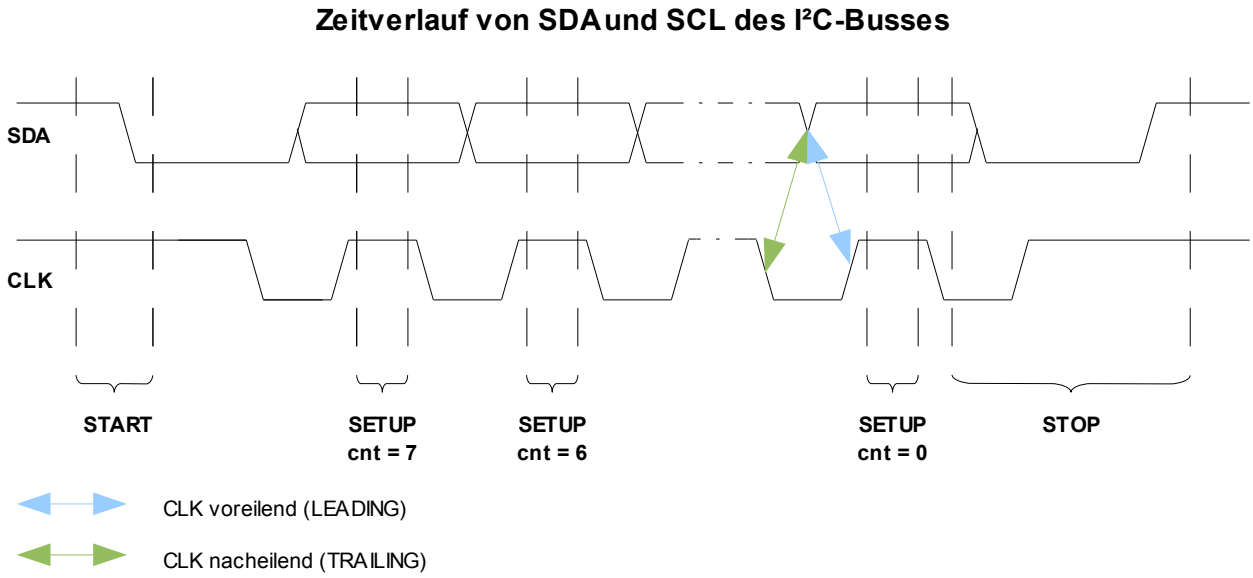

Abbildung 7: I²C BUS-Struktur

- Zu Beginn des Moduls werden neben dem Datenformat und Initialisierungen von Prozeduren diverse Signal-Stati definiert:
    - LEADING: Der Datenleitung voreilendes Taktsignal

- TRAILING: Der Datenleitung nacheilendes Taktsignal
- SETUP: Übergeben des aktuellen Bitwertes auf die Datenleitung
- SAMPLE: Abfrage, ob ein Bit gesendet werden soll

- **initMaster**

  Die Initialisierung des Masters findet dadurch statt, dass die Taktleitung auf high gesetzt wird und die Datenleitung auf einen undefinierten Zustand. Dieser undefinierte Zustand wird in VHDL mit 'Z' zugewiesen. Mit „data.initial <= False“ wird gekennzeichnet, dass die Initialisierungssequenz abgeschlossen ist.

- **ioMasterBase**

  Es wird der Status SAMPLE (Datensequenz) abgefragt, und dabei ein Zähler (data.cnt) von 7 bis 0 dekrementiert, der das aktuell zu bearbeitende Bit festlegt. Ist das letzte Bit bearbeitet, wird der Zähler zurückgesetzt. Die Taktleitung wird bei voreilendem Signal relativ zur Datenleitung auf low gesetzt und nacheilend entsprechend auf high. Es erfolgt ein ständiger Statuswechsel von LEADING nach SETUP, nach TRAILING, nach SAMPLE und wieder nach LEADING (alternierendes Verhalten der Taktleitung und Festlegung des Zeitpunktes zur Datenübergabe (SETUP)). Dieser Automat legt den Signalablauf der beiden Leitungen fest.

- **ioMasterSend**

  Die Datenleitung wird entsprechend dem zu sendenden Bitwert mittels der Hilfsfunktion to_std_logic (liefert je nach zu sendendem Bitwert 1 oder 0) während des Status SETUP gesetzt. Anschließend wird ioMasterBase aufgerufen, wobei der Zählerstatus durch doneVar abgefragt wird. Wird der Zählerstatus 0 übergeben, so wird erkannt, dass das letzte der acht Bits gesendet wurde.

- **ioMasterRecv**

  Die Empfangsprozedur funktioniert ähnlich wie die Sendefunktion. Während des Status SAMPLE wird die Datenleitung auf high oder low abgefragt und der Wert der entsprechenden Bitstelle des zu empfangenden Bytes gesetzt. Durch den Aufruf von ioMasterBase wird wieder die aktuelle Position des Bits abgefragt.

- **ioMasterAck**

  Laut Definition des I²C-Busses muss nach Beendigung des Datentransfers eine Bestätigung

des Empfängers erfolgen. Dies geschieht mit einer Übergabe einer 0 vom Slave an den Master, wenn der Slave das letzte Daten-Bit vom Master erhalten hat. Nach dem letzten gesendeten Bit werden also nochmal entsprechend die Stati LEADING, SETUP, TRAILING und SAMPLE durchlaufen, bei SETUP werden keine Daten mehr übergeben, im Status SAMPLE wird das Acknowledge registriert.

- **ioMasterStart**

  Die Übertragungssequenz von acht Bits beginnt in einem Startzustand, der dadurch erkannt wird, dass die Taktleitung auf high gelegt ist und der Pegel der Datenleitung in diesem Moment von high auf low wechselt.

- **ioMasterStop**

  Die Stopsequenz einer Übertragung ist durch einen Automaten mit fünf Zuständen realisiert. Die Taktleitung wird auf low gesetzt, anschließend wird die Datenleitung auf low gezogen, das Taktsignal geht daraufhin wieder auf high. Im folgenden Zustand wechselt die Datenleitung wieder zu high, das Taktsignal bleibt jedoch auf high und alterniert nicht mehr. Die Datenleitung kann nun wieder in den nicht definierten Datenzustand wechseln. Der Automat wird zurückgesetzt.

3.1.2 Modul SCCB.vhd

Das Modul **SCCB.vhd** beschreibt zwei Schreibmodi und eine Leseroutine, die die Elemente der I²C-Implementierung nutzen. In CamControl.vhd werden diese drei Prozeduren dann verwendet.

Der Schreibzyklus mit drei Phasen sendet drei zu schreibende Bytes: ID-Adresse, Subadresse und Daten. Dieser Modus (Sccb3PhaseWriteCycle) ist als ein Automat mit acht Zuständen programmiert.

1. **SccbWrite (Startphase)**

   Es wird abgefragt, ob die aktuelle Zustand beendet ist, ist das nicht der Fall, wird die I²C-Startsequenz (ioMasterStart) aufgerufen. Ist diese beendet, so wird in den nächsten Zustand gewechselt.

2. **SccbWrite_IDAddr**

   Es erfolgt die Abfrage ob, der aktuelle Zustand beendet ist, im negativen Fall wird die

Sendeprozedur des I²C-Moduls (ioMasterSend) aufgerufen. Dies wird automatisch erst dann beendet, bis dieser Aufruf „done“ liefert, also wenn alle acht Bits gesendet worden sind. Es folgt der Aufruf des nächsten Zustandes.

3. **SccbWrite_IDAddr_Akk**

   Die I²C-Acknowledge Prozedur wird aufgerufen, um zu signalisieren, dass das zuvor gesendete Byte beim Empfänger angekommen ist.

4. **SccbWrite_SubAddr**

   Es wird die Subadresse auf gleiche Weise wie die ID-Adresse gesendet.

5. **SccbWrite_SubAddr_Ack**

   Acknowledge der ID-Adresse.

6. **SccbWrite_Data**

   Schreiben des Dateninhaltes wie in Zustand 2.

7. **SccbWrite_Data_Ack**

   Acknowledge des Datenbytes.

8. **SccbWrite_Stop**

   Aufruf der I²C-Stopsequenz, der Automat wird für die nächste Verwendung zurückgesetzt und ein Flag gesetzt, dass die dreiphasige Schreibprozedur beendet ist.

Vereinfacht ergibt sich der auf der nächsten Seite dargestellte Automat.

Dreiphasiger Schreibzyklus (Sccb3PhaseWriteCycle)

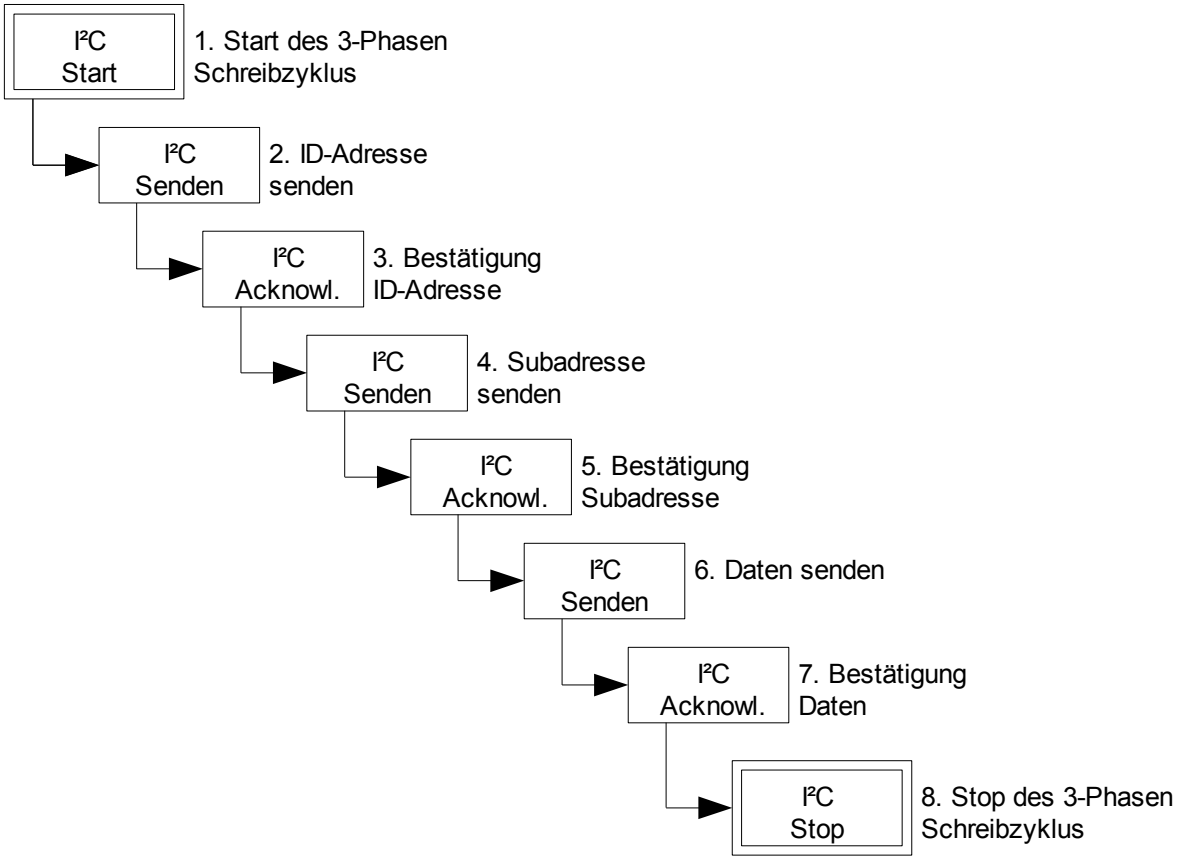

Abbildung 8: Dreiphasiger SCCB-Schreibzyklus

Der zweite Schreibmodus ist um eine Phase verkürzt, es werden nur ID-Adresse und Subadresse geschrieben, der Automat hat demnach auch nur sechs Zustände. Die Implementierung ist analog zum dreiphasigen Schreibzyklus.

Zweiphasiger Schreibzyklus (Sccb2PhaseWriteCycle)

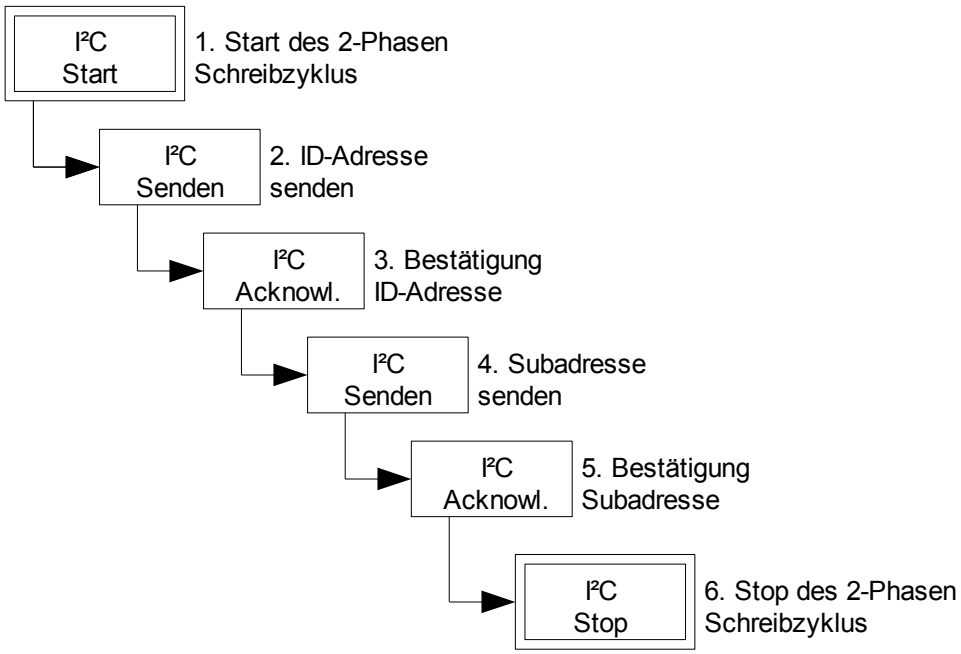

Abbildung 9: Zweiphasiger SCCB-Schreibzyklus

Der Lesezyklus ist entsprechend den beiden Schreibmodi programmiert. Er ist zweiphasig, d.h. es wird eine zu lesende Adresse gesendet und die entsprechenden Daten erhalten. Der Unterschied zum zweiphasigen Schreibzyklus besteht darin, dass im vierten Zustand die I²C-Empfangsprozedur genutzt wird, also die empfangenen Daten gelesen werden.

Zweiphasiger Lesezyklus (Sccb2PhaseReadCycle)

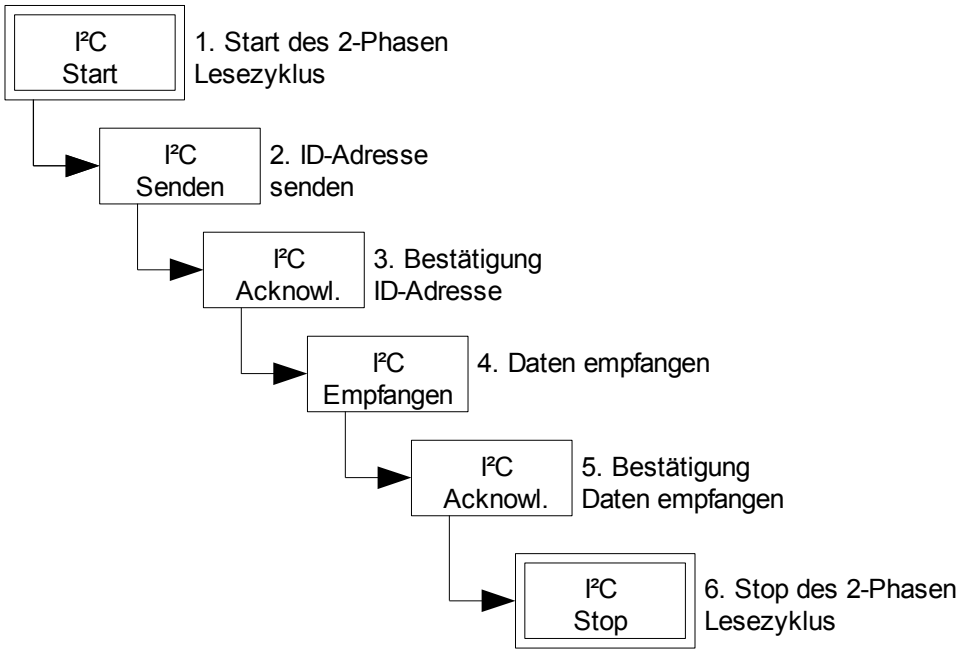

Abbildung 10: Zweiphasiger SCCB-Lesezyklus

3.1.3 Modul CamControl.vhd

Dieses Modul ermöglicht die Initialisierung der Kamera und das Auslesen und Beschreiben von Registern, um die Kamera in bestimmte Modi versetzten zu können. Dem Modul CamControl.vhd ist ein Sende- und Empfangsautomat für die RS232 serielle Schnittstelle untergeordnet, auf dessen Funktionsweise hier nicht näher eingegangen wird. Beide Automaten entsprechen weitestgehend denen, die in der Aufgabe 10 des Softwaretechnikpraktikums für den Studiengang Informationstechnik behandelt werden. Mit ihnen ist es möglich, über den RS232-Port des Evaluation Bords Daten von einem PC an das Bord zu senden und zu empfangen.

Durch das Python-Skript „kam_cmd.py“ (Unix) bzw. „kam_cmd_win.py“ (Windows) lassen sich die Initialisierungsdaten über RS232 an die Kamera senden. Es ist mit dem Skript möglich, die Kamera zu aktivieren und deaktivieren, verschiedene Kamera-Register zu beschreiben und auszulesen.

Nach dem Starten des Skriptes können diese Befehle ausgeführt werden:

1. „on“ - Einschalten der Kamera, es wird dazu das Hexadezimalzeichen 0x04 gesendet.
2. „off“ - Ausschalten der Kamera, der entsprechende Hex-Wert lautet 0x03.

3. „flush“ - Löschen des Puffers der seriellen Schnittstelle.
4. „w:0xXX:0xYY“ - Schreiben des Hex-Wertes YY in das Register XX. Das „w“ wird als 0x01 gesendet.
5. „r:0xXX“ - Lesen des Registerinhaltes von Register XX. Für „r“ wird 0x02 gesendet.

Bei jedem Befehl wird der Hex-Wert 0x42 vorangehend gesendet, diese Kennung ist für die Kamera als Slave-Adresse erforderlich. So wird erkannt, dass Daten (zu beschreibende oder auszulesende Adressen) an die Kamera gesendet werden.

Datenrahmen Schreiben			
Byte1	**Byte2**	**Byte3**	**Byte4**
Schreibmodus	ID-Adresse	Subadresse	Daten
0x01	0x42	0xXX	0xYY

Datenrahmen Lesen		
Byte1	**Byte2**	**Byte3**
Lesemodus	ID-Adresse	Subadresse
0x02	0x42	0xXX

Über CamControl.vhd werden die Befehle in Aktionen wie Lesen und Schreiben umgesetzt und die Daten aus den entsprechenden Kamera-Registern entnommen und an den PC gesendet bzw. die empfangenen Daten in die Register der Kamera geschrieben. Es werden dazu die beiden Packages I2C und SCCB genutzt.

Das Modul ist wie folgt aufgebaut:

- Für den I²C-Bus wird ein Takt generiert, der unter der laut I²C-Spezifikation maximalen Frequenz von 400kHz liegt. In diesem Fall 10kHz. Der Zähler des Taktteilers hat einen Bereich von 1 bis 625 (entspricht 80kHz), weil ein Byte pro Takt übertragen werden soll – daher der achtfache Takt.
- Im Hauptprozess wird die I²C-Initialisierung durch den Aufruf von I2C.Initmaster( ) vorgenommen.
- Es folgt eine Zustandsunterscheidung, die die empfangenen Daten auswertet:
    - CmdRecv: Es wird das gesendete Byte (Schreib-/Lesemodus oder De-/Aktivierung)

von der RS232 empfangen.

Zustandswechsel in den nächsten Zustand.

- CmdRecv_Gra: Bestätigung des Datenempfangs.

  Zustandswechsel in den nächsten Zustand.

- CmdRecv_Fin: Warten bis Handshake der RS232 vollzogen ist. Es wird unterschieden, welche Bedeutung der Datenwert des empfangenen Bytes hat:

  - 3-Phasen Schreibzyklus (Datenwert 0x01)

    Hiermit werden Daten in die Kamera-Register geschrieben.

    Zustandswechsel in den Zustand für den Empfang der ID-Adresse.

  - 2-Phasen Schreibzyklus und 2-Phasen Lesezyklus (Datenwert 0x02)

    Hiermit werden die Daten der gewünschten Kamera-Register gelesen.

    Zustandswechsel in den Zustand für den Empfang der ID-Adresse.

  - Kamera deaktivieren (Datenwert 0x03)

    Zustandswechsel in den Zustand für deaktiviert (Reset_0).

  - Kamera aktivieren (Datenwert 0x04)

    Zustandswechsel in den Zustand für aktiv (Reset_1)

  - Falls keiner dieser vier Werte zutreffend sein sollte, Zustandswechsel nach CmdRecv.

- IDAddrRecv: Empfang des Hexadezimalwertes 0x42, durch den die Kamera erkennt, ob das nächste Byte (Subadresse, aus der gelesen oder in die geschrieben werden soll) an die Kamera gesendet werden soll.

  Zustandswechsel in den nächsten Zustand.

- IDAddrRecv_Gra: Bestätigung des Datenempfangs.

  Zustandswechsel in den nächsten Zustand.

- IDAddrRecv_Fin: Warten bis Handshake der RS232 vollzogen ist.

  Zustandswechsel in den nächsten Zustand.

- SubAddrRecv: Empfang der Subadresse, also des Wertes, der der Subadresse

entspricht, in die geschrieben werden soll.

Zustandswechsel in den nächsten Zustand.

- SubAddrRecv_Gra: Bestätigung des Datenempfangs.

  Zustandswechsel in den nächsten Zustand.

- SubAddrRecv_Fin: Warten bis Handshake der RS232 vollzogen ist. Es wird nun unterschieden, ob Daten des betreffenden Registers (Subadresse) gelesen oder geschrieben werden sollen, d.h. welchen Status CmdRecv_Fin hat.

  Für den Lesemodus Zustandswechsel nach Lesen (SccbRead),

  für den Schreibmodus Zustandswechsel in den nächsten Zustand.

- DataRecv: Empfang des Bytes mit dem Dateninhalt, der in das betreffende Register geschrieben werden soll.

  Zustandswechsel in den nächsten Zustand.

- DataRecv_Gra: Bestätigung des Datenempfangs.

  Zustandswechsel in den nächsten Zustand.

- DataRecv_Fin: Warten bis Handshake der RS232 vollzogen ist. Die Daten sollen als nächstes in die Register geschrieben werden, daher erfolgt der Zustandswechsel nach SccbWrite.

- Aktivieren und deaktivieren der Kamera:

  - Reset_0: Die Resetleitung des Kamera-Adapters wird auf low gesetzt.

    Zustandswechsel nach CmdRecv, es wird also eine Aktivierung bzw. Neuintialisierung gewartet.

  - Reset_1: Die Resetleitung des Kamera-Adapters wird auf high gesetzt.

    Zustandswechsel nach CmdRecv, warten auf eine Initialisierung.

- Schreibmodus, bei dem die empfangenen Daten in die Register der Kamera geschrieben werden:

  - SccbWrite: Starten des SCCB 3-Phasen Schreibzyklus. Wenn dieser beendet ist,

Zustandswechsel nach CmdRecv. Es werden drei Bytes an die Kamera gesendet: 0x42 (ID-Adresse), die Subadresse (zu beschreibendes Register) und die Daten.

- Leseroutine, bei der der Registerinhalt der gesendeten Subadresse ausgegeben wird.
    - SccbRead: Starten des SCCB 2-Phasen Schreibzyklus: Es wird die ID-Adresse (0x42) und die zu lesende Adresse an die Kamera übergeben. Wenn dies beendet ist, wird das Byte für die ID-Adresse um 0x01 inkrementiert, also auf den Hex-Wert 0x43. Diese Slave Adresse signalisiert der Kamera, dass fortan gelesen wird (also Daten von der Kamera ausgegeben werden.

      Zustandswechsel in den nächsten Zustand.
    - SccbRead_2: Starten des SCCB 2-Phasen Lesezyklus. Es wird jetzt als ID-Adresse 0x43 übergeben und der Dateninhalt der im Zustand zuvor gesendeten Subadresse gelesen.

      Wenn dies beendet ist, wechselt der Zustand wieder in den Startmodus CmdRecv.

      Die empfangenen Daten werden in ansByte geschrieben.
    - Es folgt ein Automat mit drei Zuständen, der die aus dem Kamera-Register gelesenen Daten über die serielle Schnittstelle zurücksendet und welche dann im Python-Skript als Antwort ausgegeben werden.

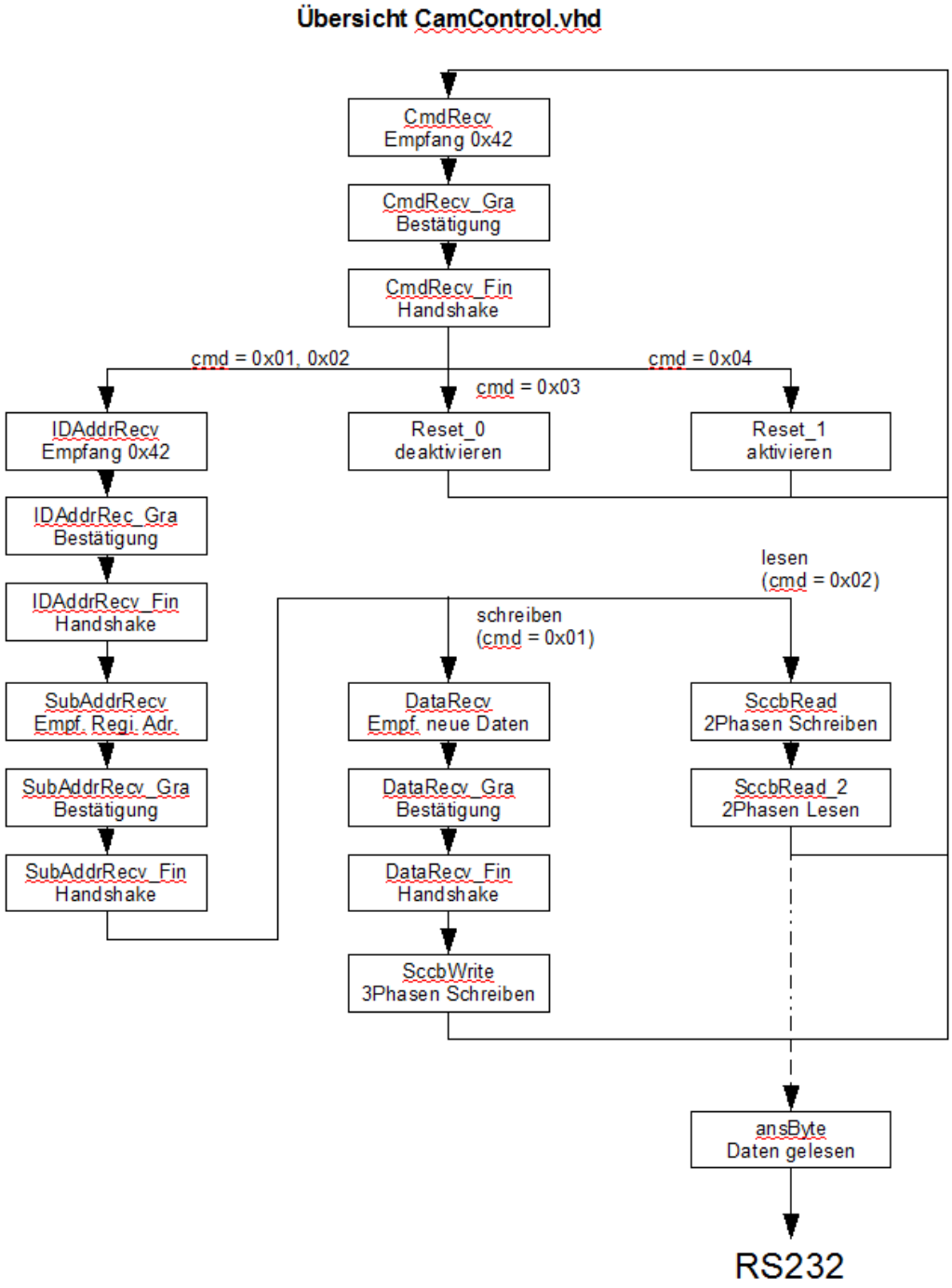

Abbildung 11: Übersicht zum Modul camControl.vhd

3.2 Datenverarbeitung und Bildausgabe

Im weiteren wird beschrieben, wie die Daten der Kamera im SRAM zwischengespeichert werden und deren Ausgabe auf einem Monitor erfolgt. Dazu werden zuerst die nötigen Hilfsmodule EdgeDetection.vhd, MemPack.vhd und und Clk2x_dcm.vhd beschrieben, bevor auf die Funktionen von kamera.vhd (Bedienung der Kamera, Bildausgabe, Diodendetektion, etc.) eingegangen wird.

3.2.1 Modul EdgeDetection.vhd

Dieses Package dient zur Taktflankenerkenung von Signalen. Es ist ein zweistelliges Schieberegister implementiert, in das von rechts immer ein neuer Bitwert als Signalzustand „eingeschoben“ wird. Es lassen sich dadurch vier Zustände abbilden:

- 0 1 : steigende Taktflanke, von low auf high
- 1 0 : fallende Taktflanke, von high auf low
- 1 1 : stabiler High-Pegel
- 0 0 : stabiler Low-Pegel

Für die vier Fälle ist je eine Funktion vorgesehen, die das zweistellige Schieberegister als Wert zurückliefert. Für eine beliebige Signaländerung ist noch die Funktion anyEdge vorhanden, die je nach vorliegender Situation den Registerwert für fallende oder steigende Taktflanke zurückliefert. Dieses Modul findet in dem Hauptmodul kamera.vhd Anwendung.

3.2.2 Modul MemPack.vhd

Dieses Modul wird aktuell im Projekt nicht genutzt, es kann jedoch für spätere Modifikationen mit eingebunden werden. Es dient zur Koordination der von der Kamera eingehenden Bilddaten (drei Bytes pro Pixel), die systematisch im Speicher abgelegt werden sollen. Das Hauptaugenmerk liegt hierbei darauf, dass pro Speicherzelle 32 Bit gespeichert werden, jedoch nur 24 Bit pro Pixel von der Kamera eintreffen. Es muss darauf geachtet werden, dass sich der Schreibprozess mit dem Lesezugriff nicht überschneidet.

Dazu wird ein Array mit zwölf Bytes angelegt, das dann drei Speicherzellen belegen kann und aus 4 x 3 Datenblöcken mit Bildinformationen besteht. Pro Pixel werden drei Bytes abgelegt (für die Kanäle Rot, Grün und Blau), es werden also in dem Array die Bildinformationen für vier Pixel geschrieben.

Array

Bytes	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Speicherzelle	**2**				**1**				**0**			
Byte pro Zelle	3	2	1	0	3	2	1	0	3	2	1	0
Pixel	**3**			**2**			**1**			**0**		
Byte pro Pixel	2	1	0	2	1	0	2	1	0	2	1	0
Dateninhalt	**R**	**G**	**B**	**R**	**G**	**B**	**R**	**G**	**B**	**R**	**G**	**B**

Die Prozedur „Mem3to4_writeSetInpData“ schreibt die Daten von vier Pixeln in vier Takten als Pakete von drei Bytes nacheinander in das Array. Die Funktion „Mem3to4_writeCommitRequired“ liefert als Wahrheitswert, ob eine Speicherzelle aktuell ausgelesen werden darf. Im positiven Fall gibt die Funktion den Wert „True“ zurück, anderenfalls „False“. Wenn Pixel 0 geschrieben ist und Pixel 1 noch nicht, darf Zelle 0 nicht ausgelesen werden, weil Byte 3 von Zelle 0 noch alte Daten vom vorangegangenen Schreibzyklus enthält. Sind die Daten von Pixel 1 in das Array geschrieben, dann liefert die Funktion „True“, weil dann Speicherzelle 0 vollständig mit neuen Daten versehen ist und ausgelesen werden darf.

Wann zu welchem Zeitpunkt eine bestimmte Speicherzelle auszugegeben ist, ist in der Funktion „Mem3to4_writeGetOutpData“ festgelegt. Wenn, wie bereits geschildert, Pixel 1 noch nicht in den Speicher geschrieben ist, dürfen keine Daten des Arrays gelesen werden. Wenn Pixel 2 geschrieben wird, wird Speicherzelle 0 ausgelesen, weil diese mit aktuellen Daten vollständig beschrieben ist. Wenn Pixel 3 geschrieben wird, wird Speicherzelle 1 gelesen, wenn Pixel 0 geschrieben wird, wird Speicherzelle 2 gelesen.

Zeitlicher Ablauf Speichern

Speicherzelle	**2**				**1**				**0**			
Byte pro Zelle	3	2	1	0	3	2	1	0	3	2	1	0
Takt 0										Pixel 0		
Takt 1							Pixel 1					
Takt 2				Pixel 2								
Takt 3	Pixel 3											

Takt 0: Speicherzelle 2 lesen

Takt 1: Nichts lesen, ungültiger Zustand

Takt 2: Speicherzelle 0 lesen

Takt 3: Speicherzelle 1 lesen

Eine Speicherzelle darf nicht ausgelesen werden, wenn sie gerade beschrieben wird oder noch alte Daten vom letzten Schreibzyklus enthält.

Die Prozedur „Mem3to4_readGetOutpdata“ erlaubt das Lesen pro Pixel der drei Bytes, die die RGB-Werte enthalten. Wenn diese Prozedur genutzt wird, wird nichts in die betreffenden Speicherzellen geschrieben, es handelt sich um einen reinen Lesevorgang (beispielsweise um einen zuvor im Speicher abgelegten Screenshot anzuzeigen). Dies wird durch einen Automaten mit vier Zuständen realisiert, die Pixeldaten des Arrays werden in Dreierblöcken ausgelesen.

Die Prozedur „Mem3to4_readSetInpData“ ermöglicht das Zurückschreiben der zuvor mit „Mem3to4_readGetOutpdata“ ausgelesenen Daten in das Array. Diese Daten können somit zwischenzeitlich modifiziert werden. Somit könnte theoretisch ein Bild im Speicher mit einem Bildeffekt versehen werden, der durch Veränderung der Pixeldaten umgesetzt wird.

Die Funktion „Mem3to4_readFetchRequired“ liefert „True“, wenn nicht alle vier Pixel-Daten ausgelesen sind, also noch Daten benötigt werden. Sind die drei Bytes des letzten Pixels erfasst, liefert die Funktion „False“.

Wie bereits erwähnt, wird dieses Modul in diesem Projekt zur Zeit nicht verwendet. Der Lese- und Schreibzugriff ist in dem Modul kamera.vhd implementiert. Sollte für die Kamera zukünftig der RGB-, statt den bisher verwendeten Bayer-Pattern-Modus verwendet werden, so könnte dieses Modul von Vorteil sein.

3.2.3 Modul clk2x_dcm.vhd

Der DCM (Digital Clock Manager) ermöglicht es, das Spartan3-Evaluation Board außerhalb der spezifizierten 50MHz zu betreiben. Durch den DCM kann das Board maximal mit einem Takt von 280MHz betrieben werden. In diesem Projekt soll eine Taktfrequenz von 100MHz genutzt werden. Dies ist darin begründet, dass für die VGA-Ausgabe eine Frequenz von 25MHz benötigt wird (siehe Kapitel 2.1) und die Ausgabe in einen Automaten mit vier Zuständen integriert werden soll. Eine Erklärung dazu befindet sich in der Erläuterung zu kamera.vhd.

Über den DCM können vielfältige Taktfrequenzen erzeugt werden:

$$F_{ClkFX} = F_{ClkIN} \cdot \frac{ClkFX\_Multiply}{ClkFX\_Divide}$$

Diese Funktion wird jedoch in der finalen Version des Projektes nicht mehr genutzt. Desweiteren

lassen sich zum Beispiel Phasenverschiebungen eines Taktsignals um 90°, 180° und 270° bei Bedarf einstellen.

Über CLKIN_PERIOD wird die Grundfrequenz festgelegt, in diesem Fall mit 20ns. Über CLK2X kann das doppelt so schnelle Taktsignal abgegriffen werden, was 100MHz entspricht und in kamera.vhd genutzt wird.

3.2.4 Modul Kamera.vhd

Kamera.vhd setzt auf alle vorgestellten Module auf, es stellt die direkte Verbindung zwischen Kamera und Monitorausgabe dar. Im Folgenden ist aufgeführt, welche Funktionen in Kamera.vhd implementiert sind:

- Bedienung der Kameras:
  - Umschalten zwischen Kamera 1 und 2 mittels Switch (6)
  - Switch (0) und (1) = „00“ oder „11“ : Live-Bild der aktuellen Kamera
  - Switch (0) und (1) = „01“ : Live-Bild mit Differenzbildeffekt
  - Switch (0) und (1) = „10“ : Anzeige des zuletzt erstellten Screenshots
  - Switch (2) : Icon-Darstellung für LEDs aktivieren/deaktivieren
  - Button (3) : Screenshot im SRAM ablegen
- Speichern der Bilddaten im SRAM
- Lesen der Daten aus dem SRAM
- Modifizieren des Live-Bildes zu einem Differenzbildeffekt: Nur sich ändernde Kanten und Flächen werden dargestellt.
- Ausgabe des Bildes über den Monitor-Adapter
- Wenn vor einer Kamera ein Infrarotfilter montiert ist, kann ein Infrarotpunkt entdeckt werden. Diese Funktion ist dahingehend modifiziert, dass zwei Infrarot-LEDs detektiert werden können, die mit zwei unterschiedlich vorgegebenen Frequenzen blinken. Für LED1 wird ein grünes Icon auf dem Monitor an entsprechender Stelle ausgegeben, für LED2 ein blaues Icon.

Ausgangsposition:

kamera.vhd ist für einen Prototypen der CamA-Platine ausgelegt, auf dem nur eine Kamera angebracht ist. D.h. es muss nach Modifizieren der UCF-Datei eine Funktion eingebaut werden, die es erlaubt, zwischen beiden Kameras der Finalen Version der CamA-Platine hin- und herzuschalten.

Die Kamera stürzt bei dem Bootvorgang sofort ab. Es stellt sich heraus, dass sie einen falschen Takt erhält, bzw. das Taktsignal an falscher Stelle gesetzt ist, dies muss korrigiert werden. Ein weiteres Problem besteht darin, dass das Bild leicht verrauscht dargestellt wird. Die Ursache liegt darin, dass das Taktsignal der Kamera und die des Monitor-Adapters nicht synchronisiert sind. Letzterer Fehler ist gänzlich behoben, der Absturz der Kamera tritt unter bestimmten Umständen temporär auf und ist reproduzierbar. Dies wird in der Fehlerbeschreibung näher betrachtet. Desweiteren ist die Differenzbildfunktion und die Monitorausgabe bereits implementiert. Alle weiteren Funktionen sind von mir hinzugefügt worden.

In dem Prozess „process(clk1x)“ werden die Steuersignale des Monitor-Adapters gesetzt: mclk (25MHz) und der Takt für das Kamera-Modul (6,25Mhz). Weiterhin werden noch die Frequenzen der LEDs (~3Hz und ~1,5Hz) erzeugt, ebenso ein Signal für ein blinkendes Icon, das auf dem Monitor in bestimmten Situationen (wenn eine LED gefunden wird) angezeigt werden soll.

Der Kameratakt muss bei 6,25MHz liegen, weil der AD-Converter der Kamera laut Datenblatt mit maximal 12MHz betrieben werden darf. Der Takt des FPGA lässt sich allerdings nicht genau auf 12MHz herunterteilen, sondern nur bis 12,5MHz, was sich in den Tests als unbrauchbar herausstellt (zu schnell). Die Kamera schaltet sich bei dieser Frequenz ab. Daher kommt lediglich die nächste Taktstufe von 6,25MHz als Betriebsfrequenz in Frage. Mittels Kamera-internem Prescaler kann der Takt dann wieder erhöht werden. Dies wird im Kapitel „Inbetriebnahme“ genauer erläutert.

In dem Hauptprozess, der mit 100MHz durch den DCM betrieben wird, befindet sich ein Automat mit vier Zuständen (VGAState), der pro Takt den Zustand wechselt. D.h. der Code eines jeden Zustandes wird mit 25MHz bearbeitet. Dies ist nötig, weil in Zustand 1 die Monitorausgabe durchgeführt wird, die einen 25MHz Takt verlangt. In Zustand 2 werden die Bilddaten, die die Kamera liefert, im SRAM gespeichert. Zustand 3 enthält den Leseprozess der Daten aus dem SRAM. Diese drei Taske müssen getrennt voneinander ausgeführt werden, um Kollisionen zu vermeiden.

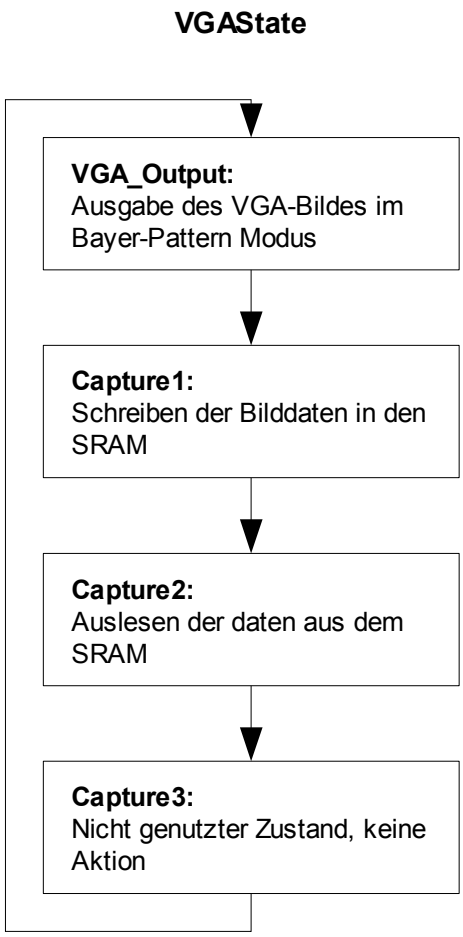

Abbildung 12: Automat für VGA-Ausgabe, Bilddaten speichern und lesen

Monitorausgabe (VGA_Output)

Die Ansteuerung des Monitor-Adapters erfolgt wie in Kapitel 2 beschrieben. Die Bilddaten werden jedoch nicht im RGB-Modus, sondern im Bayer-Pattern Modus ausgegeben.

Bayer Pattern

Für jeden Pixel wird nur eine Farbinformation an seiner dementsprechenden Stelle aufgenommen. In der ersten horizontalen Pixelreihe wird alternierend grün und blau registriert, in der darauffolgenden Pixelreihe abwechselnd rot und grün. Die dargestellten Pixel unterscheiden sich in ihrer Helligkeit. Als ganzes betrachtet ergibt sich dann für den Betrachter ein Bild.

Bayer Pattern

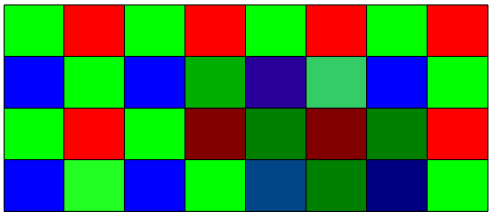

Abbildung 13: Aufbau Bayer Pattern

Es ist ein Automat mit vier Zuständen implementiert, der eine Matrix erzeugt, die bei gerader horizontaler Pixelreihe abwechselnd grüne und rote, bei einer ungeraden Pixelreihe blaue und grüne Werte auf die Farbkanäle schaltet.

	Ausgabe			
Wenn Pixel-Position in X-Richtung (1 downto 0)	00	01	10	11
Wenn gerade horizontale Pixelreihe	G	R	G	R
Wenn ungerade horizontale Pixelreihe	B	G	B	G

Dabei muss berücksichtigt werden, dass das Bild einen starken Grünstich erhält. Um dies zu umgehen, wird das gesamte Byte des Grünkanals um eine Stelle nach rechts verschoben, das most significant Bit ist also in dem Fall des Grünkanals immer 0. Die Daten für den Automat stehen in dem 32Bit-stelligen Signal readDataTmp, das vor jeder Ausgabe Daten aus dem SRAM zugewiesen bekommt. Es werden also diese vier Bytes durch den Automaten ausgegeben.

Nach Beendigung der Ausgabe wird der Zeiger für die Leseposition um eins inkrementiert, d.h. es wird bei der nächsten Ausgabe der Dateninhalt der nächsten Speicherzelle ausgegeben (ebenfalls 32 Bit).

Im nächsten Zustand (Capture1) sollen die von der Kamera erhaltenen Daten in den SRAM geschrieben werden. Somit muss die zu beschreibende Adresse festgelegt werden. Da es sich hierbei um ein Signal handelt, muss die Adresse einen Takt vor dem Schreiben gesetzt werden, damit sie gültig ist. Es sollen drei Bilder gespeichert werden, also wird zwischen drei Schreibpostionen im SRAM unterschieden: Der Position für das Live-Bild der Kamera (wPos), der Adresse für das Differenzbild (wDiffPos) und die des Screenshots (wScreenPos).

Daten im SRAM speichern (Capture1)

Wenn eine Schreibaufforderung (writeRequest) gestellt wird, werden die Daten des Live-Bildes (writeData) auf den IO-Port des SRAM gesetzt. Der SRAM wird im Modus für 32Bit-Adressen verwendet. Dementsprechend wird „write enable“ beider RAM-Module aktiviert, die Daten werden jetzt gespeichert. Anschließend wird die Position des Zeigers für die als nächstes zu beschreibende Adresse inkrementiert.

Die drei Bilder (Live-Bild, Differenzbild und Screenshot) sollen nacheinander im Speicher liegen. In einer Speicherzelle (32 Bit) können also vier Pixel geschrieben werden. Daher wird für das Live-Bild der Raum von Adresse 0 – 76799 genutzt.

Dies ergibt sich aus folgender Berechnung:

$$\frac{640 \cdot 480\ Pixel}{4\ Pixel / Adresse} = 76800\ Adressen$$

Demnach muss die Position des Schreibzeigers des Differenzbildes bei Position des Zeigers des Live-Bildes + 76800 liegen. Da der Screenshot noch hinter dem Differenzbild im Speicher ist, befindet sich die Schreibposition bei Position des Live-Bildes + (2 x 76800). Dementsprechend werden die Adresspositionen der drei Bilder um eins inkrementiert. Danach wird die Schreibanfrage für das Live-Bild deaktiviert und die Aufforderung für das Differenzbild (writeDiffRequest) gesetzt. Die Daten werden auf gleiche Weise wie das Live-Bild gespeichert.

Die Schreibanforderung für den Screenshot (writeScreenRequest) erfolgt nur, wenn der Button btn(3) gedrückt wird. Die Bilddaten werden ebenso wie das Live-Bild gespeichert. Um den Raum von 3 x 76800 = 230400 Adressen abbilden zu können, muss das Signal 18-Bitstellen haben (2^{18} = 262144).

Im nächsten Automatenzustand sollen die Bilddaten aus dem SRAM gelesen werden. Hierzu müssen die Lesepositionen, wie auch schon die Schreibpositionen, einen Takt vorher gesetzt werden.

Mit den Schaltern SW(1) und SW(0) wird zwischen den Bildern umgeschaltet, wie bereits eingangs Kapitel 3.2.4 erwähnt wurde. Die Lesepositionen für die SRAM-Adressen werden analog zur Schreibpostion berechnet.

Daten aus dem SRAM lesen (Capture2)

Hierzu wird „write enable“ auf high gesetzt, also deaktivert. Als nächstes wird die zu lesende

SRAM-Adresse auf den betreffenden Output-Port gesetzt und der I/O-Port auf hochohmig geschaltet (Signal: „ZZZZZZ … ZZZ“). In diesem Zustand kann nun über den I/O-Port des SRAMs gelesen werden. Dies geschieht immer im letzten Zustand zur Ausgabe des Bayer-Patterns (wenn readRequest gesetzt ist, wird srio readDataTmp zugewiesen).

Der Zustand Capture3 wird nicht genutzt, es findet die Übergabe zu Zustand1 (VGA_Output) statt.

Taktsignale der Kamera setzen

Wenn Switch SW(6) auf '0' geschaltet ist, werden die Signale der linken Kamera (CamA von vorn betrachtet) übernommen. Es handelt sich dabei um die Signale PCLK1, HREF1 und camVSYNC1, deren Taktflanken durch das Modul EdgeDetect detektiert werden. Weiterhin wird der Farbwert (D1) eines Pixels als 8-Bit Wert von der Kamera ausgelesen.

Einbinden des Moduls CamControl (CamControl_Inst)

Dem Modul wird der Takt von 50MHz (clk1x), der Takt des I²C-Busses (SCL), die seriellen Sende- und Empfang-Ports (rs232TX und rs232RX), das Byte zum I²C-Datentransfer (SDA), sowie das Reset-, Powerdownsignal übergeben.

Bilddatenerfassung

Wechselt das Signal zur vertikalen Synchronisation von low auf high (EdgeDet.risingEdge(vsyncBuf)), so wird die Schreibposition für die Datenspeicherung im SRAM (wPos) auf 0 zurückgesetzt. D.h. es soll ein neues Bild gespeichert werden. Mit dem Wechsel des vertikalen Synchronisationssignals von high auf low (EdgeDet.fallingEdge(vsyncBuf)) beginnt das neue Bild (isFrame = True).

Wenn das horizontale Referenzsignal von low auf high wechselt (EdgeDet.RisingEdge(hrefBuf)), so beginnt eine neue horizontale Pixelzeile (isLine = True). Diese wird mit einem Pegelwechsel von high auf low beendet.

Wenn eine neue Zeile beginnt (isLine = True), wird bei jedem Pixeltakt (EdgeDet.risingEdge(pclkBuf)) ein Automat (packPos) ausgeführt, der 32 Bit Bilddaten zwischenspeichert:

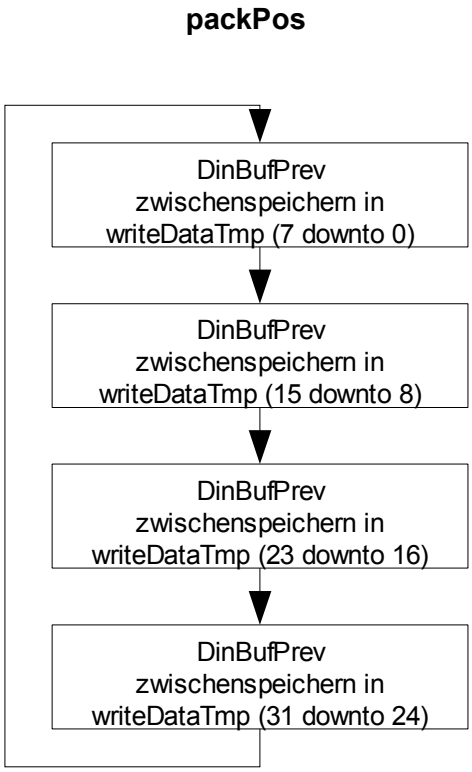

Abbildung 14: Einlesen der Kameradaten

writeDataTmp wird an writeData übergeben, das im Automat VGAState im Zustand Capture1 in den SRAM geschrieben wird.

Aus den zwischengespeicherten Daten soll ein Differenzbild erstellt werden.

Differenzbild

Das Differenzbild soll dazu dienen, Bewegungen zu erkennen und auch nur diese darzustellen. Das soll heißen, befindet sich vor der Kamera eine unbewegliche Szenerie, so bleibt das ausgegebene Bild schwarz. Sobald sich auf dem Bild etwas bewegt, also eine Veränderung auftritt, so soll diese hell an dieser Stelle auf dem Monitor dargestellt werden. Die Überprüfung auf eine Bildänderung entsteht durch einen Vergleich der aktuellen Daten mit vorangegangenen Daten. Die aktuellen Bilddaten werden als Variable von der Kamera ausgelesen (writeDataTmp). Um die aktuellen Daten mit früheren Daten vergleichen zu können, werden die Bilddaten als Signal zwischengespeichert (writeDataLastframe), das sich nicht so schnell ändert wie die Variable wrieDataTmp, sondern erst mit jedem Takt. An dieser Stelle gibt es ein Problem, das in Kapitel „Auswertung“ näher erläutert wird.

Aus den aktuellen Daten und den früheren Daten wird die Differenz gebildet und an die Funktion twoscompToAbs übergeben. Diese Funktion überprüft, ob ein großer Unterschied zwischen den alten und neuen Daten besteht, d.h. ob bei der Differenz des 8-Bit Wertes bei dem most significant Bit eine 1 vorliegt. In diesem Fall wird ein heller Bildwert an der Stelle mit großen Änderungen im Bild zurückgegeben und als Differenzbild gespeichert (VGAState, Zustand Capture1). Im Falle einer 0, also ein sehr kleiner Wert und somit geringe Bildänderung, wird dieser Wert als Bilddaten in den SRAM gespeichert. Ein sehr kleiner Wert bedeutet eine dunkle Ausgabe auf dem Monitor an dieser Stelle.

Detektion zwei blinkender Infrarot-LEDs

Das Erkennen von zwei Infrarot-Dioden kann nur dann sichergestellt sein, wenn sich vor der aktivierten Kamera ein Infrarotfilter befindet und keine störenden Lichtquellen im Bildbereich der Kamera sind, deren Spektrum große Frequenzanteile im Infrarotbereich haben. Weiterhin müssen sich die Dioden in einem Abstand von 1m bis ca. 2m befinden. Diese Forderung ist darin begründet, weil eine Leuchtquelle zu nah vor der Kamera einen zu großen Lichtfleck bilden würde. Eine LED, die zu weit weg von der Kamera ist, würde einen zu kleinen oder gar keinen Lichtfleck erzeugen. Ist dieser zu klein, könnte es sich beispielsweise auch um einen Pixelfehler handeln und der Automat würde die LED falsch erkennen.

Um diverse Störquellen zu eliminieren, wird ein Schwellwert festgelegt, ab dem ein Infrarotobjekt detektiert wird. Dieser Schwellwert liegt bei x“bb“, alles darüber wird als Infrarotquelle angesehen (dinBufPrev > x“bb“). Ein Problem ergibt sich darin, dass die Kamera einen falschen Bildwert liefern könnte. Das geschieht zum Beispiel, wenn die Kamera einen Pixelfehler hat. Dann könnte an dieser Pixelposition ein sehr heller Farbwert ausgewertet werden, obwohl sich an dieser Stelle kein Infrarot-Leuchtpunkt befindet. Zu diesem Zweck startet die Erkennung der LEDs erst dann, wenn zwei Infrarot-Pixel hintereinander gefunden werden. Realisiert ist dieser Mechanismus in einem Schieberegister (startDetection), das bei einem gefundenen Pixel eine 1 erhält, ansonsten eine 0. In dem Schieberegister muss also „11“ stehen. Dies ist die einfachste Vorgehensweise, um die Wahrscheinlichkeit fälschlicher Detektion zu verringern (es müssten in diesem Fall zwei Pixelfehler nebeneinander auftreten). Wie man dies prinzipiell verbessern kann, ist im Kapitel „Ausblick“ geschildert.

Damit die Position auf dem aktuellen Bild registriert wird, existieren zwei Signale, die bei jedem neu eingelesenen Pixel in horizontaler und vertikaler Richtung inkrementiert werden.

CurCamPixelPosX wird in jeder Bildspalte von 0 bis 639 gezählt, curCamPixelPosY in jeder Zeile von 0 bis 479. Wenn eine Zeile beendet ist, wird der Zähler in X-Richtung zurückgesetzt (EdgeDet.fallingEdge(hrefBuf)), das gleiche geschieht bei dem Zähler für die Y-Richtung bei Beginn der vertikalen Synchronisationssignal, also kurz bevor das neue Bild beginnt (EdgeDet.fallingEdge(vsyncBuf)). Die allgemeine Pixelposition bewegt sich also von oben links Zeile für Zeile nach rechts unten.

Der Automat „detectedLeds“ erwartet im Bild die erste LED. Für jede LED werden die Koordinaten aufgezeichnet (curCamPixelPosXLed1 und curCamPixelPosYLed1 für die erste im Kamerabild befindliche LED und curCamPixelPosXLed2 und curCamPixelPosYLed2 für die zweite im Bild gefundene LED). Ist eine LED nicht gefunden, so sind die betreffenden Werte auf 0 gesetzt. Wird eine LED erkannt, so bleibt die Position so lange gespeichert, bis die LED bei dem nächsten Frame in einem bestimmten Bereich um seine alte Position wiedergefunden wird. Diese Detektionszone hat eine bestimmte Lebenszeit. Wird die LED in diesem Bereich innerhalb einer bestimmten Zeit nicht wiedergefunden, so wird die Position der LED auf X=0 und Y=0 zurückgesetzt und die Detectionszone (area1LifeTime für LED1 bzw. areaLifeTime für LED2) ist nicht mehr gültig. Die LED wird dann im ganzen Bild neu gesucht. Dieser Mechanismus ist notwendig, da es erlaubt sein soll, dass sich die LED im Bild langsam bewegen darf (Roboter läuft) und die LED nicht durchgehend leuchtet, sondern blinkt. D.h. das Programm muss imstande sein, eine LED wiederzufinden, wenn diese sich plötzlich an einer neuen unerwarteten Stelle befindet (Roboter ist umgefallen und wird wieder hingestellt).

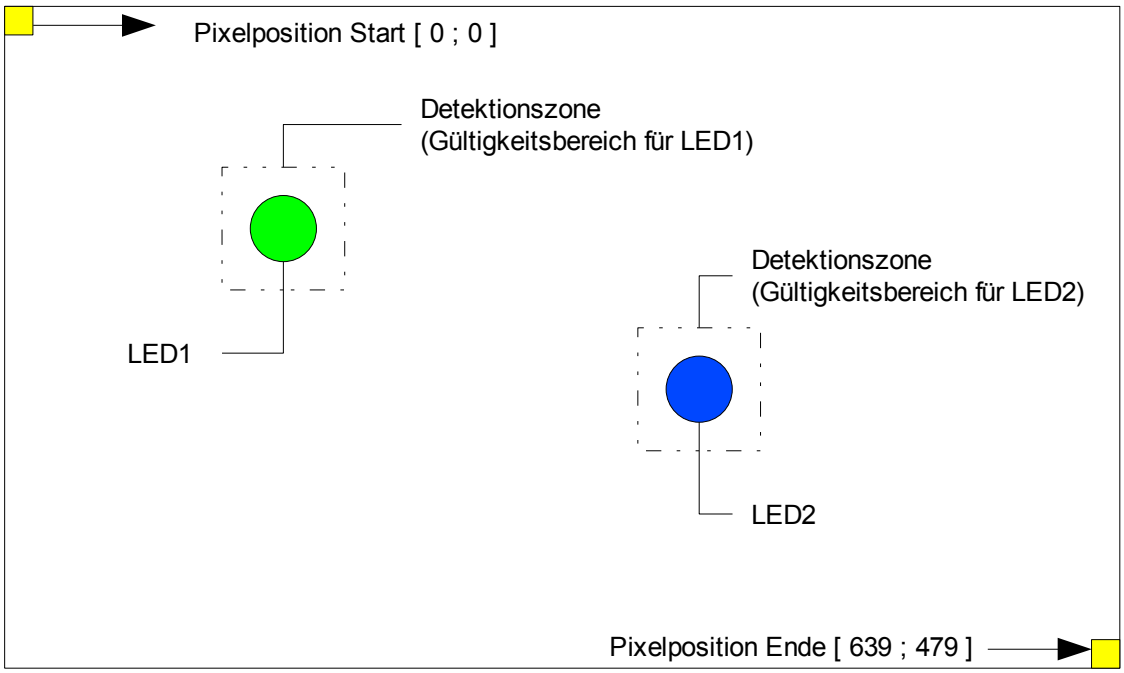

Abbildung 15: Detektion der Ir-LEDs

Der Automat ist zur Zeit für zwei LEDs ausgelegt, kann jedoch um die betreffenden Signale und Zustände erweitert werden, um noch mehr Leuchtdioden detektieren zu können. Die Funktionsweise ist im einzelnen Folgende:

Es müssen zwei Infrarot-Pixel nebeneinander gefunden werden. Ist dann die erste LED noch nicht gefunden (hitLed1 = '0'), und befindet sich die Position der LED im Startzustand (curCamPixelPosXLed1 und curCamPixelPosYLed1 = 0), so wird die aktuelle Position der LED1 gespeichert (detectedLed1X/Y), dieses Signal wird später für die Anzeige des Icons für LED1 verwendet. CurCamPixelPosXLed1 und curCamPixelPosYLed1 erhalten ebenfalls die aktuellen Koordinaten, ebenso areaPixelPosXLed1 und areaPixelPosYLed1. Die beiden letzteren Signale dienen bei der nächsten Abfrage zu Berechnung der Detektionszone um die LED. Weiterhin wird vermerkt, dass im letzten Bild LED1 gefunden wurde. Dies wird durch einen Zähler bewerkstelligt, der auf den Wert 840000 gesetzt und mit jedem weiteren dargestellten Pixel dekrementiert wird. Dieser Wert ergibt sich daraus, dass in dem Bereich der VGA-Darstellung (800 x 520 Pixel), also den letzten 416000 Pixeln, der Pixel von LED1 detektiert worden ist. Dieser Wert ist mit zwei multipliziert, da sich die LED bewegen kann und im ungünstigen Fall so die 416000 Pixel überschritten werden könnte. Außerdem wird die Lebenszeit der Detektionszone der LED1 auf 10 gesetzt. Beginnt ein neues Bild (EdgeDet.risingEdge(vsyncBuf)), wird der Zähler für die Detectionszone um 1 dekrementiert. So lange der Zähler größer als 0 ist, ist die Detektionszone gültig. Zum gleichen Zeitpunkt wird auch hitLed1/2 stets auf 0 zurückgesetzt, d.h. es muss in jedem neuen Bild immer nach den LEDs gesucht werden.

Ist die LED gefunden, so wird bei der nächsten Abfrage geprüft, ob sich die LED in der Detektionszone von 20 x 20 Pixel befindet. Ist das nicht der Fall, wird der Zähler der Detektionszone bis auf 0 heruntergesetzt, der Automat schaltet in die Abfrage zuvor und die LED wird neu gesucht. Ist die LED in dem Gültigkeitsbereich, so werden alle Werte der Signale und Zähler wie bei der Abfrage zuvor gesetzt. An dieser Stelle könnte man den Code des Automaten verkürzen, jedoch würde dann die If-Abfrage eingangs zu lang und unübersichtlich werden. Daher ist die Abfrage aufgeteilt worden.

Für die Detektion der LED2 schaltet der Automat (detectedLeds) in den nächsten Zustand. Die Abfrage nach der zweiten LED erfolgt wie bei der LED1, die Zähler und Signale sind dementsprechend anders benannt.

Der Gültigkeitsbereich der beiden LEDs hat den Vorteil, dass sich die LEDs horizontal und vertikal auf gleicher Höhe befinden dürfen. Weiterhin ist so eine Bewegung der LED erlaubt. Ist der Gültigkeitsbereich etwas größer als der Leuchtfleck der LED dimensioniert, so ist eine lokale

Varianz des Leuchtflecks erlaubt.

Erkennen der Blinkfrequenz der jeweiligen LED und Icon-Ausgabe

Für das Erkennen der Blinkfrequenz existieren zwei Schieberegister controlReg1 und controlReg2 zu je 8 Bit, die mit einem Takt von 6Hz (clk_controlReg) inkrementiert werden. Wird im letzten Bild ein IR-Pixel einer LED gefunden, so wird auf das Schieberegister eine 1 gesetzt, anderenfalls eine 0. Somit ergibt sich ein Muster in den beiden Registern, je nach Blinkfrequenz. Je nachdem welche der beiden LEDs (langsam (1,5Hz) blinkende oder schnell (3Hz) blinkende LED) zuerst im Bild erkannt wird, dessen Blinkmuster steht in controlReg1. Also muss untersucht werden, welches Muster in welchem Register vorliegt. Dazu werden folgende Variablen entsprechend dem jeweiligen Fall gesetzt:

foundLedFreq1InReg1 = True, wenn Blinkmuster der mit 3Hz blinkenden LED in Register 1 gefunden wird.

foundLedFreq2InReg1 = True, wenn Blinkmuster der mit 1,5Hz blinkenden LED in Register 1 gefunden wird.

foundLedFreq1InReg2 = True, wenn Blinkmuster der mit 3Hz blinkenden LED in Register 2 gefunden wird.

foundLedFreq2InReg2 = True, wenn Blinkmuster der mit 1,5Hz blinkenden LED in Register 2 gefunden wird.

In dem Automaten für die Monitor-Ausgabe VGAState, Zustand VGA_Output, wird dann je nach gefundenem Blinkmuster an der Stelle detectedLed1X und detectedLed1Y (analog für die zweite gefundene LED ein Icon ausgegeben. Für die mit 3Hz blinkende Diode ist ein grünes Icon, für das Blinkmuster von 1,5Hz ein blaues Icon vorgesehen. Das Zeichen besteht jeweils aus einem Quadrat, dessen obere linke Ecke dem gefundenen Pixel entspricht und einem blinkenden Schriftzug „POS“ für „Position“.

Das jeweilige Icon wird also nur dann angezeigt, wenn das Blinkmuster im einen der Schieberegister mit dem für die Frequenz entsprechenden Bitcode übereinstimmt.

Auf der achtstelligen LED-Anzeige des Evaluationsboardes wird das Kontrollregister1 ausgegeben, also das Frequenzmuster der zuerst gefundenen LED.

4. Inbetriebnahme

Kamera-Adapter

- Über das von XILINX mitgelieferte Tool IMPACT ist die kompilierte Datei „kamera.bit“ des Projektes auf das Evaluation Board zu laden.
- Starten des Python-Skripts je nach Betriebssystem kam_cmd_win.py (Windows) oder kam_cmd.py (Unix). Es ist darauf zu achten, dass das Python-Skript Befehle über die COM1-Schnittstelle erwartet.
- „on“ eingeben, um die Kamera zu starten.
- „w:0x12:0x13“ eingeben, damit in Register x“12“ der Wert x“13“ geschrieben wird. Dadurch wird die Kamera in den Bayer-Pattern Modus gesetzt, entsprechend der Bildausgabe in kamera.vhd.
- „w:0x0d:0xc1“ eingeben, um den Multiplikator für den Prescaler auf 8x zu stellen.

  Damit ergibt sich der Interne Takt für die Kamera:

$$f_{internalClock} = \frac{f_{inputClock} \cdot Multiplyer}{(CLKRC + 1) \cdot 2} = 25\text{MHz} \quad (4.1)$$

  Mit

  $f_{inputClock}$ = 6,25MHz

  Multiplyer = 8

  CLKRC = 0

Laut Datenblatt hat die Kamera intern ein Pixelfeld von 664 x 490 Pixel, ähnlich der VGA-Ausgabe von 800 x 520 Pixel. Daraus ergibt sich wiederum folgende Bildwiederholrate:

$$f_{Bildwiederholrate} = \frac{f_{internalClock}}{664 \cdot 490\, Pixel} \approx 77\text{Hz} \quad (4.2)$$

- Über den Switch SW(6) kann zwischen den beiden Kameras umgeschaltet werden.
- SW(1) = 0 und SW(0) = 0 oder SW(1) = 1 und SW(0) = 1 : Kamera Live-Bild
- SW(1) = 0 und SW(0) = 1 : Differenzbild
- SW(1) = 1 und SW(0) = 0 : Screenshot anzeigen. Wenn noch kein Screenshot ausgelöst wurde, wird hier ein Bildausschnitt angezeigt, der gespeichert wurde, bevor die Kamera in

den Bayer-Patern-Modus geschaltet wurde.

- SW(2) : Icon-Darstellung für die Ir-LEDs aktivieren/deaktivieren
- BTN(3) : Screenshot auslösen.

Monitor-Adapter

- Über das von XILINX mitgelieferte Tool IMPACT ist die kompilierte Datai „monatest2.bit“ des Projektes auf das Evaluationsboard zu laden.
- Die Bedienung des jeweiligen Testprogramms ist dem Kapitel 2.1 zu entnehmen. Der Kamera-Adapter hat hierbei keine Funktion.

5. Auswertung

Da die Testprogramme zum Monitor-Adapter übersichtlich und nicht umfangreich sind und bei der Ausführung keine Probleme bereiten, ist die Auswertung auf den Kamera-Adapter fokussiert.

Im Folgenden sind zuerst Probleme und Einschränkungen erläutert.

Problem 1:

In seltenen Fällen schaltet sich der angeschlossene Monitor ab und meldet (modellabhängig) die vierfache horizontale und vertikale VGA-Inputfrequenz zu erhalten (horizontal 125kHz, vertikal 240Hz). Sporadisch tritt auch die dreifache Frequenz auf (horizontal 93,7kHz, vertikal 180Hz), sowie die doppelten Frequenzen (horizontal 62,4kHz und vertikal 120Hz). Gefordert sind horizontal 31,25kHz und vertikal 60Hz. Überprüft man dies mit einem Oszilloskop, so werden tatsächlich im Fehlerfall diese Frequenzen an den Pins der VGA-Schnittstelle angezeigt.

Dieses Problem ist reproduzierbar und tritt vor allem unter diesen Bedingungen auf:

- Die Kamera soll ein Bild in einem nicht gut ausgeleuchteten Raum aufnehmen. Meist ist dann kurz auf dem Monitor zu beobachten, dass die Kamera versucht, einen Helligkeitsausgleich vorzunehmen, d.h. das Bild leuchtet kurz hell auf und passt sich dann insoweit an, dass Gegenstände im Bildbereich zu erkennen sind. Anschließend schaltet sich der Monitor ab.
- Ein Objekt befindet sich sehr dicht vor der Kamera und füllt fast das ganze Bild aus. Bewegt man nun diesen Gegenstand, liegen doppelte, dreifache oder vierfache Frequenzen am Monitor-Eingang an, der sich dann ausschaltet.

Die beiden genannten Punkte haben die Gemeinsamkeit, dass sich in kurzer Zeit bei einer großen Anzahl von Pixeln die Werte ändern. Dieses Problem tritt nämlich nicht auf, wenn die Kamera eine ruhige Szenerie aufnimmt, bei der sich nur kleine Bildausschnitte verändern. In diesem Fall ist ein störungsfreier Betrieb von sechs Stunden getestet worden, der dann durch mich abgebrochen wurde. Wenn die Kamera mit einem Infrarotfilter benutzt wird und sich die LEDs in dem spezifizierten Abstand zur Kamera befinden, ist ebenfalls ein Langzeitbetrieb ohne Fehler möglich. Bewegt man jedoch die LED dicht auf die Kamera zu, so dass ein Großteil des Bildschirms ausgeleuchtet wird, so ergibt sich wieder der bereits genannte Fehler (es ändern sich auch in diesem Fall wieder viele Pixelwerte!).

Die Ursache besteht möglicherweise darin, dass, wie bereits in Kapitel 4 beschrieben, die Bildwiederholrate der Kamera mit 77Hz über den spezifizierten 60Hz liegt. Über den FPGA lässt sich mit einem Taktteiler für den Kamera-Takt XCLK (6,25MHz), der laut dem Hersteller maximal 12MHz betragen darf, eben jener Takt von 12MHz nicht genau erzeugen. Über einen Taktteiler sind 3,125MHz, 6,25MHz oder auch 12,5MHz realisierbar, wobei bei Betrieb mit dem Letzteren sofort die fehlerhaften Frequenzen an der VGA-Schnittstelle anliegen. Über die Formel 4.1 lässt sich somit auch nicht eine vertikale Frequenz von 60Hz erreichen, dazu müsste ein interner Kamera-Takt von

$$60\text{Hz} \cdot 664 \cdot 490\, Pixel = 19521600\text{Hz} \quad (5.1)$$

erzeugt werden. Da dies jedoch nicht erreichbar ist, kann auch nicht überprüft werden, ob dies die Ursache des Problems ist.

Die Kamera und der FPGA scheinen dabei nicht abgestürzt zu sein, denn auf der achstelligen LED-Anzeige, auf der das Kontrollregister für das Frequenzmuster der Ir-LED angezeigt wird, wird immer noch eine Werteveränderung dargestellt. Daher müssen noch Bilddaten von der Kamera vom FPGA verarbeitet werden. Daher ist das Problem vermutlich eher im Umfeld des THS8134 (MonA) zu suchen.

Problem 2:

Das Differenzbild kann mit einem Kamera-Takt (clkXCLK = 6,25MHz) nicht dargestellt werden. Erzwingt man trotzdem die Berechnung, so tritt sofort nach dem Einschalten (Python-Skript Befehl „on“) Problem 1 auf. Die Ursache liegt hierbei darin, dass das Signal writeDataLastFrame nicht mehr korrekt gesetzt wird. Offensichtlich werden die von der Kamera ausgelesenen Bilddaten zu schnell in den Speicher geschrieben, so dass sie nicht über den Befehl writeDataLastFrame <= srio ausgelesen werden können, bzw. writeDataLastFrame ändert sich zu schnell, ehe die Berechnung des Differenzbildes ausgeführt werden kann. Um einen stabiles Verhalten des Programms zu erreichen, wurde die Anweisung readLastRequest <= True auskommentiert. Somit ist writeDataLastFrame immer auf 0 gesetzt, eine Differenzbildberechnung findet zwar noch statt, jedoch werden keine Daten mehr aus dem Speicher gelesen. Der Versuch, writeData zu einem verzögerten Zeitpunkt an writeDataLastFrame (Speicherauslesen wird somit umgangen) zu übergeben, führt zu keinem besseren Ergebnis. Bei einem Takt von clkXCLK = 3,125MHz funktioniert die Differenzbildberechnung.

Problem 3:

Die Frequenz der Infrarot-Dioden ist auf maximal 3Hz einstellbar. Bei höheren Frequenzen schalten die Dioden nicht mehr aus, auf dem Monitor wird dies als ein Auf- und Abdimmen der LEDs sichtbar. Dies liegt nachweislich nicht am Programmcode sondern an der Beschaffenheit der LEDs. Positioniert man eine handelsübliche Infrarotfernbedienung z.B. eines TV-Gerätes vor der Kamera mit Infrarotfilter, ist sichtbar, dass diese mit einer deutlich höheren Frequenz als 3Hz beschaltet wird. Dies wird auch so durch die Kamera registriert und dementsprechend auf dem Monitor ausgegeben. Problematisch sind evtl. die langen Kabel an denen die LEDs angeschlossen sind. Ein weitere Möglichkeit besteht darin, dass die Kathode der LEDs zu lange nachleuchtet, was u.U. durch eine kapazitive Belastung hervorgerufen werden könnte.

6. Zusammenfassung

In dieser Studienarbeit werden zwei Platinen für das Spartan3-Evaluation Board getestet und Anwendungen in VHDL geschrieben. Für den Monitor-Adapter sind fünf einfach und übersichtlich gehaltene Testprogramme erstellt, mit denen die Möglichkeiten zur Farbwiedergabe des Monitor-Adapters aufgezeigt werden kann. Er wird, wie bis auf die fünf zusätzlichen Signale (m1, m2, nSync, nblank und syncT), die nicht weiter genutzt werden und nur einmal auf feste Werte initialisiert werden, genauso angesteuert wie die VGA-Schnittstelle des Evaluations Boardes. Als Erweiterung kommt hierbei hinzu, dass jeder Farbkanal über 8 Bit verfügt und somit 2^{24} Farben dargestellt werden können. Wird dieses Modul allein in Betrieb genommen, treten keine Probleme auf.

Für die Benutzung des Kamera-Adapters muss zuerst der Code dahingehend angepasst werden, dass überhaupt ein Bild auf dem Monitor entsteht. Dazu ist der Kamera-Takt XCLK an der korrekten Stelle mit der richtigen Frequenz zu setzen. Dazu wird der Takt über einen Taktteiler aus dem Hauptprozess mit 50MHz generiert. Der zuvor erzeugte Takt aus dem DCM führt zu instabilem Verhalten. Weiterhin werden Optimierungen vorgenommen - wie die Synchronisation des Monitor-Adapter Taktes mclk mit dem Haupttakt - somit ist das dargestellte Bild nicht mehr verzerrt. Anschließend wird das Projekt vom CamA-Prototypen auf die finale Platine mit zwei Kameras umgestellt, und zwar dahingehend, dass beide Kameras angesprochen werden können.

Als erweiterte Funktion für das Projekt wird die Detektion von zwei Infrarot-Leuchtdioden hinzugefügt, die mit unterschiedlicher Frequenz blinken.

Um für nachfolgende Arbeiten an der MonA- und CamA-Platine eine Basis zu stellen, sind in der Studienarbeit die Module, die zur Kommunikation, Datentransfer und zur schlussendlichen Bilddarstellung dienen, in den obigen Kapiteln erläutert.

Die aufgetretenen Probleme sind nicht alle vollständig gelöst, sie sind, soweit es möglich ist, dargestellt.

7. Ausblick

Im weiteren werden einige Verbesserungsvorschläge zur aktuellen Situation gegeben:

Wie eingangs in der Aufgabenstellung erwähnt, verfügen die Roboter nicht mehr über eine, sondern zwei LEDs. Dies ist zur Richtungserkennung notwendig. Die Studienarbeit beschreibt hiermit nur eine Möglichkeit, wie die Detektion von Leuchtdioden behandelt werden kann. Um mehrere Roboter auf dem Spielfeld zuordnen zu können, müssen mehrere verschiedene Blinkfrequenzen zur Verfügung stehen. Also müssen Frequenzen von über 3Hz möglich sein. Liegt das Problem tatsächlich an den langen Kabeln, die beim Testen verwendet werden, so ist dies später im Anwendungsfall nicht gegeben, da auf einem Roboter keine langen Kabel benutzt werden. Alternativ kann ein anderer Typ von LEDs eingesetzt werden, der kürze Anstiegs- und Abfallzeiten des Stroms aufweist.

Die Detektion von Pixelfehlern ist nicht optimal. Eine bessere Lösung wäre es, eine Art Tiefpassfilter über das infrarot-gefilterte Bild zu legen. Dadurch werden hohe Frequenzen, also abrupte Farbwechsel, geglättet. Somit würden Pixelfehler eliminiert werden. Durchführbar wäre dies beispielsweise, wenn man das gesamte Bild im SRAM speichert und dann blockweise die Werte benachbarter Pixel mittelt.

Die Position einer LED wird nicht exakt ermittelt. Das liegt daran, dass zur Zeit die ersten beiden Pixel eines Leuchtflecks als „LED gefunden“ angenommen werden. Dies ist insofern nicht allzu tragisch, da sich die LEDs später in einem Abstand zur Kamera befinden werden, aus dem ein ohnehin nur kleiner Leuchtfleck entsteht. Eine verbesserte Lösung wäre, die Dimension des Leuchtflecks zu vermerken und in horizontaler und vertikaler Richtung die Mitte als Position der LED anzunehmen. Da sich jedoch später die LEDs in einem bekannten Abstand zur Kamera befinden werden und die LEDs somit nur einen erwartet großen Leuchtfleck liefern, kann dies in die aktuelle Positionsberechnung als Offset mit einbezogen werden.

8. Literaturverzeichnis

Monitor-Adapter

[THS8134B] Datenblatt Texas Instruments THS8134, Texas Instruments, Dallas USA, 2000

[UEBVGA] Aufgabe Bildschirmansteuerung, Institut für Informatik, TU Clausthal, 2007

Kamera Adapter

[OV7725] OV7725 Color CMOS VGA Camera Chip Sensor, Omni Vision Technologies, 2007

[OV7725AN] OV7725 Camera Module Hardware Application Notes, Omni Vision Technologies, 2007

[I2CBUSPH] I²C-Bus Specification And User Manual, Philips 2007

[MATRIXV] Farbverarbeitung mit Bayer-Mosaic Sensoren, Matrix Vision GmbH, 2001

[SCHA] Schaltpläne zum CamA-Adapter, Institut für Informatik, TU Clausthal, 2009

[CODE] Vorgegebene Code-Fragmente in VHDL, Institut für Informatik, TU Clausthal, 2009

Allgemein

[SPARTAN] Spartan-3 Starter Kit Board User Guide, XILINX, 2005

[DCM] Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, XILINX, 2006

Verwendete Software

[XILINX] XILINX ISE 11.2, XILINX

[OPENO] OpenOffice

[PSP7] PaintShop Pro 7, Jasc Software/Corel Inc.

9. Abbildungsverzeichnis