Hochschule Koblenz
Fachbereich Ingenieurwesen
Elektro- und Informationstechnik

Studienarbeit

Bildverarbeitung in einem FPGA

Autor: Christian Oster
Matrikelnummer: 539475
31.August 2021
Studiengang: Informationstechnik

Kurzzusammenfassung

In dieser Studienarbeit wird das Thema Bildverarbeitung in einem FPGA behandelt. Dabei werden Kamerabilder von einem FPGA-Board aufgenommen, gespeichert, weiterverarbeitet und interpretiert, um Informationen aus dem Bild zu extrahieren. Diese Ausarbeitung ist eine Weiterentwicklung auf Basis eines bestehenden Projektes. Aufbauend auf bereits vorhandenen Funktionen, der Speicherung von Kamerabildern und deren Ausgabe über VGA auf einen Monitor.

Ziel war es, die bestehenden Projektstände zu reviewen und mithilfe der vorhandenen Funktionen eine zweidimensionale Objekterkennung zu implementieren. Dabei werden Erkenntnisse aus einer vorherigen Studienarbeit verwendet und auf dieser weiter ausgebaut.

Zusätzlich soll diese Ausarbeitung dem Leser einen guten Überblick darüber bieten, was bereits existierte und wie die Funktionsweise der verschiedenen Teile sind.

Für die Umsetzung wurde die Hardwarebeschreibungssprache „Very High Speed Integrated Circuit (VHDL) „verwendet und neue Module implementiert, mit denen es möglich ist, ein Objekt im zweidimensionalen Raum in seiner Position zu bestimmen.

Abkürzungsverzeichnis

VHDL	Very High Speed Integrated Circuit
FPGA	Field Programmable Gate Array
SDRAM	Synchronous Dynamic Random Access Memory
ROM	Read Only Memory
FPS	Frames per Second/ Bilder pro Sekunde
FIFO	First In First Out
VGA	Video Graphics Array
RGB	Red Green Blue/ Rot Grün Blau
ADDR	Address
REQ	Request
WR	WRITE
RD	READ
EN	ENABLE
CLK	CLOCK

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Einsatzmöglichkeiten der digitalen Bildverarbeitung in der industriellen Anwendung sind vielseitig und stehen daher im Fokus vieler verschiedener Branchen. (Frauenhofer) Bestimmte Anwendungen benötigen eine möglichst schnelle Verarbeitung und so handelt es sich bei dieser Arbeit um eine besondere Form der Bildverarbeitung, und zwar der in Echtzeit.

Unter Echtzeit (real time) versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen. (DIN44300)

Diese Arbeit beschäftigt sich mit der Realisierung einer Kamera basierten Positionserkennung von Objekten. Diese soll in Echtzeit diese Objekte erkennen. Dabei soll die Information eines Farbunterschied zwischen Hintergrund und Objekt, die Erkennung der Position des Objektes ermöglichen.

Farbe ist, in dieser Arbeit durch die Auswahl des Sensors auf eine Kamera der Firma Sentech, durch einen Rot-, Grün- und Blaukanal (RGB) definiert. Diese drei Farbkanäle spiegeln die Grundfarben wider, welcher der Sensor dem natürlichen Licht entnimmt.

Um die Verarbeitungszeit im System gering zu halten, wurde bei der Auswahl der Kamera auf eine hohe Bilderfassungsrate geachtet. Die Auswertung der Bilder für die Erkennung eines Objektes und dessen Position erfolgt in einem Field Programmable Gate Array (FPGA-Board). Das FPGA-Board bietet die Möglichkeit, die Signale von der Kamera in Echtzeit zu verarbeiten und auszuwerten und hat aufgrund seines Aufbaus weniger Limitationen als ein herkömmlicher Mikroprozessor, was die Bearbeitungsgeschwindigkeit angeht.

Zu Bearbeitungsbeginn der Studienarbeit war es möglich, die Daten der Kamerabilder mittels CameraLink-Schnittstelle von der Kamera auf das FPGA-Board zu übertragen und diese dort zwischenzuspeichern. Die zwischengespeicherten Daten konnten über den VGA-Ausgang auf einem angeschlossenen Monitor ausgegeben werden. (Herbst, 2020) Zudem war es möglich, in einer einzigen Zeile, da her eindimensional, anhand von Graustufen (Schwarzweißbild) und dessen Wechsel in der Intensität bis zu drei Objekte zu erkennen. Zusätzlich existierte in einem weiteren Projekt die gleiche Anbindung, jedoch an eine Farbkamera. Dieses Projekt wurde zusätzlich herangezogen, um dieses Projekt im Farbbereich fortzuführen.

Auf Basis der zuvor beschriebenen Funktionen werden neue Funktionen implementiert. Die Funktionsweise der bisherigen Objekterkennung im Graustufenbereich soll für die Erkennung im Farbbereich als Vorlage verwendet werden. So sollte es einfacher sein, ein farbiges Objekt von einem anders gefärbten Hintergrund zu unterscheiden und somit auch dessen Position zu erkennen, als die Erkennung mittels Graustufen zu implementieren.

Wichtig bei der Erfassung ist das Herausfinden der relevanten Parameter, die eine robuste Objekterkennung erst möglich machen. So spielen beispielsweise die Beleuchtungszeit, Umgebungslicht und die Farbwahl von Objekt und Hintergrund eine große Rolle.

Diese Arbeit ist gegliedert in eine kurze Erklärung der benötigten Werkzeuge und Mechanismen die, zum Verständnis der bisher existierenden und neu dazu gekommenen Module. Darauf folgt die Präsentation von dem, was bereits vor dieser Studienarbeit existierte und wichtig für die Schlüssigkeit des ganzen Projektes ist. Anschließend werden die Module und Funktionen, die innerhalb dieser Studienarbeit entstanden, im Detail beleuchtet. Zum Schluss werden die neuen Erkenntnisse präsentiert und diskutiert.

2 Digitale Bildverarbeitung

Diese Studienarbeit stützt sich in vielen Abschnitten aus Erkenntnissen der vorangegangenen Studienarbeit „Bilddatenvorverarbeitung in einem FPGA, 25. Januar 2020“ von Lukas Herbst. Dessen Arbeit bezieht sich in vielen Teilen auf die Literatur von Rafael C. Gonzalez und Richard E. Woods, die zusammen das Buch „Digital Image Processing“ verfasst haben. Eine detaillierte Beschreibung zu diesem Abschnitt finden Sie in der Arbeit von Herrn Herbst. Folgend die Details die für diese Ausarbeitung am wichtigsten waren.

2.1 Funktionsweise Darstellung von Pixeln

Ein digitales Bild kann aus einer endlichen Anzahl aus Pixeln bestehen, so wird meist mit x die horizontale Breite und mit y die vertikale Höhe definiert. Die Gesamtzahl der Pixel in einem Bild ergibt sich aus dem Produkt von x und y.

So ergibt sich ein Raster aus Pixeln. Jedes Pixel kann mit der Funktion f(x,y) einzeln ausgelesen werden.

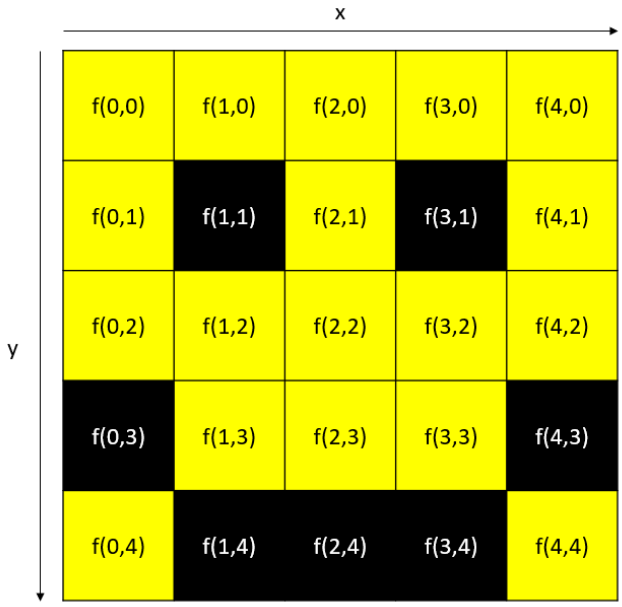


Abbildung 1: Digitales Bild Beispiel

Der Rückgabewert der Funktion f(x,y) ist abhängig von der Definition des Pixels. So wird beispielsweise für ein Bild im Graustufenbereich nur ein einziger Wert definiert oder wie in Abbildung 1 ein RGB Farbwert. Ein RGB Farbwert verfügt über drei Informationen, diese definieren wie stark ein Rot-, Grün- oder Blauanteil ausgeprägt ist.

Beispiel zu Abbildung 1: f(0,0) = {rot = 255, grün = 255, blau = 0}
Das Beispiel ergibt einen starken Gelbton. Die Zahl 255 stellt hier den Maximalwert für die Darstellung mit 8 Bit pro Farbkanal dar und wird auch Farbtiefe genannt. Je mehr Bits für die Farbtiefe zur Verfügung stehen, desto feiner können Farbunterschiede realisiert werden. Für diese Arbeit wird von einer Farbtiefe von 8-Bit ausgegangen.

2.2 Farbunterschiede erkennen

Wie in (Herbst, 2020, S. 2.3) beschrieben, können Unterschiede von Pixeln in einer Reihe wie folgt erfasst werden.

$$1.\,Ableitung\ Definiton\ von\ Gonzalez\colon\ \frac{\partial f}{\partial x} = f(x+1) - f(x)$$

Das Ergebnis der ersten Ableitung beschreibt den Unterschied von einem zum nächsten Pixel. Nachfolgend eine Veranschaulichung zu einer Reihe die aus 5 Pixeln besteht. Jedes Pixel wird durch eine Graustufe repräsentiert.

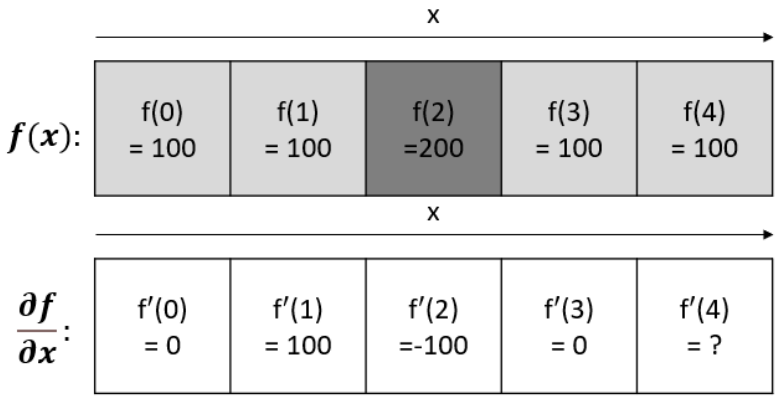
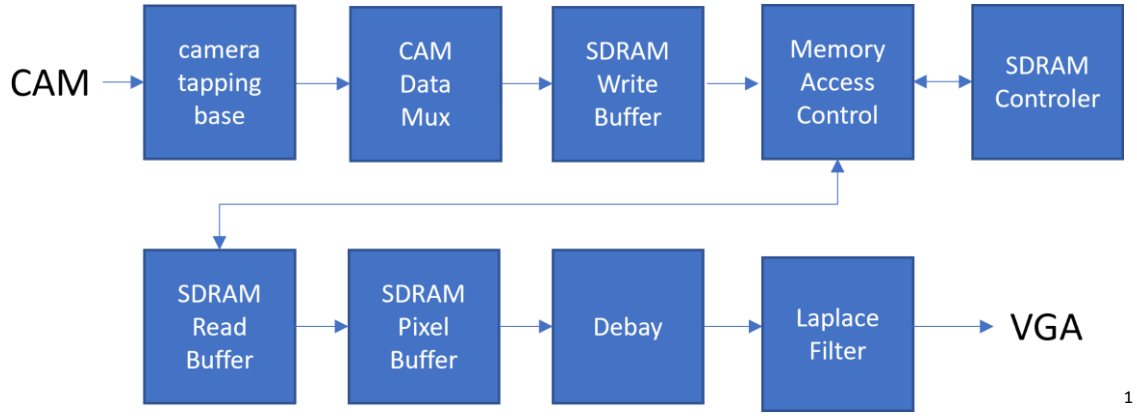


Abbildung 2: Pixelreihe f(x) und 1. Ableitung f(x), Graustufen in 8 Bit Darstellung

An der Stelle f(0) ergibt die erste Ableitung null, da das Nachbarpixel keinen Unterschied aufweist. Ein Sprung ist an einem hohen Betrag der ersten Ableitung zu sehen, so zum Beispiel an f(1) oder f(2). Hier ändert sich der Wert sehr stark im Vergleich zum Nachbarn. Eine Besonderheit stellen die Ränder da, hier kann man mit der Formel der ersten Ableitung keinen Wert berechnen, daher werden diese vernachlässigt.

3 Übersicht Zustand vorher

Eine kurze Übersicht bietet folgende Grafik und beschreibt den vorgefundenen Zustand, der bei Beginn des Projektes übernommen wurde.



[1]

Abbildung 3: Übersicht der Module innerhalb des bisherigen Quartus Projekt

In der Ausarbeitung der Studienarbeit „Bilddatenvorverarbeitung in einem FPGA“ von Herrn Herbst, sind die Module im Detail beschrieben. Daher werden im Folgenden die Module in Ihrer groben Funktion beschrieben, um sich ein Gesamtbild des Projektes machen zu können.

Abbildung 3 erfasst die Aufteilung der Module innerhalb des FPGA Board und beschreibt den Zustand, wie er im Quartusprojekt von Herrn Perske vorgefunden wurde. Die bisherigen Module der Linienerkennung ist auf dieser Übersicht nicht zu finden, da der ursprüngliche Stand von Herrn Perske als Beginn dieser Arbeit verwendet wurde.

Der Aufbau der vorgefundenen Hardware ist in Abbildung 4 zu sehen.

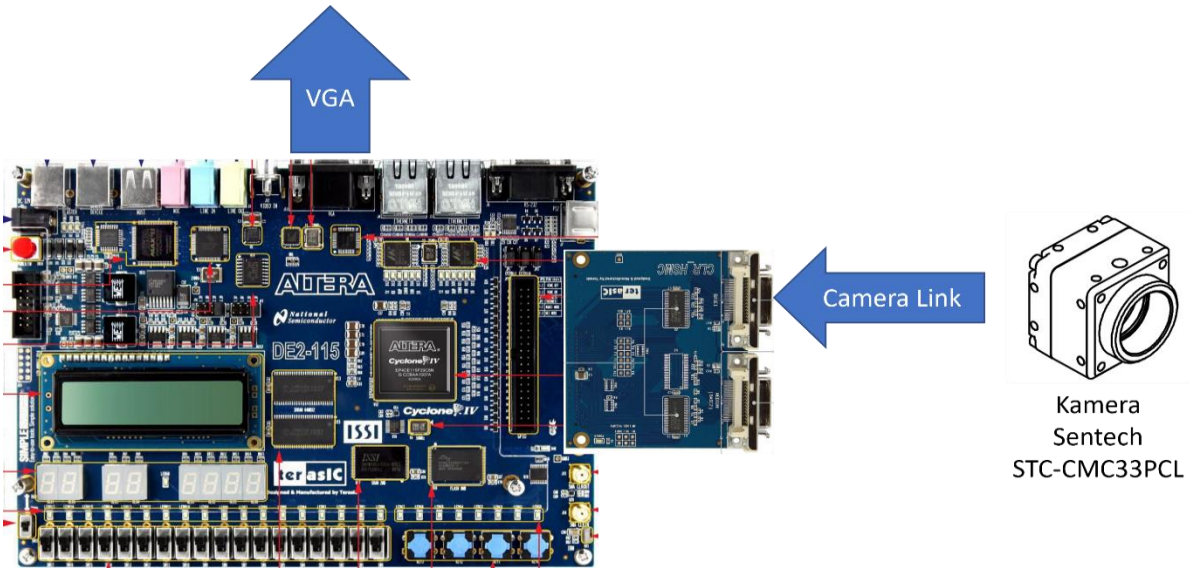


Abbildung 4: Übersicht verwendete Hardware, Bild aus folgenden Quellen zusammengestellt:
(Quelle Bild FPGA: DE2-115 User Manual)
(Quelle Bild Kamera: SENTECH STC-CMC33PCL Product Spec)
(Quelle Bild Zusatzboard: CLR-HSMC User Manual)

[1] Modul „LaplaceFilter“ hieß vorher „Convolution“. Zum besseren Verständnis wurde das Modul umbenannt, ist jedoch in alten Projektständen von Herrn Perske noch als „Convolution“ Modul anzufinden. Die Funktionsweise bleibt gleich.

3.1 Modul: cameralink_tapping_base

Die Kamera „STC-CMC33PCL“ verfügt über verschiedene Modi für die Übertragungsweise der Kameradaten. Der erste Kontaktpunkt zwischen FPGA-Modul und Kamera stellt das Modul „cameralink_tapping_base“ dar. In dem Modul werden die Daten der CameraLink-Schnittstelle nach der Spezifikation der Kameramodi vorverarbeitet.

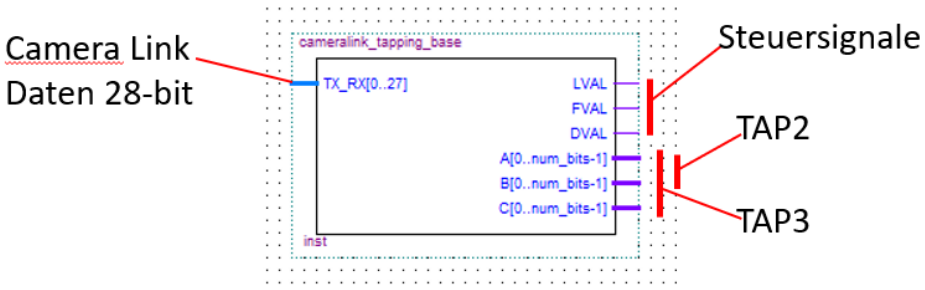


Abbildung 5: Blockschaltbild

Die Eingangsgröße „CLRRX_BASE[0..27]“ stellt die Daten dar, die von der Kamera an das FPGA Board übermittelt werden. Die Definition vom Eingang „TX_RX“ kann in der Spezifikation (CameraLink Spec, 2000) zu CameraLink nachgelesen werden und wird an dieser Stelle nicht genauer erläutert. Um die Ausgangsgrößen aus Abbildung 5 zu verstehen, benötigen wir ein Verständnis dafür welche Modi in der Kamera zur Verfügung stehen.

3.1.1 Kameramodi: Taps, horizontales und vertikales Timing

Ein Bild wird von der Kamera zeilenweise übermittelt, für die Übertragung werden die Steuersignale LVAL, FVAL und DVAL verwendet.

LVAL = Line Valid, HIGH definiert gültige Pixel
FVAL = Frame Valid, HIGH definiert gültige Zeile
DVAL = Data Valid, HIGH definiert Daten gültig

Jeder Pixel kann dabei aus einer Bittiefe von 8, 10 oder 12 Bits bestehen, dies ist eine weitere Einstellung der Kamera.

3.1.2 Horizontales Timing 2 Tap

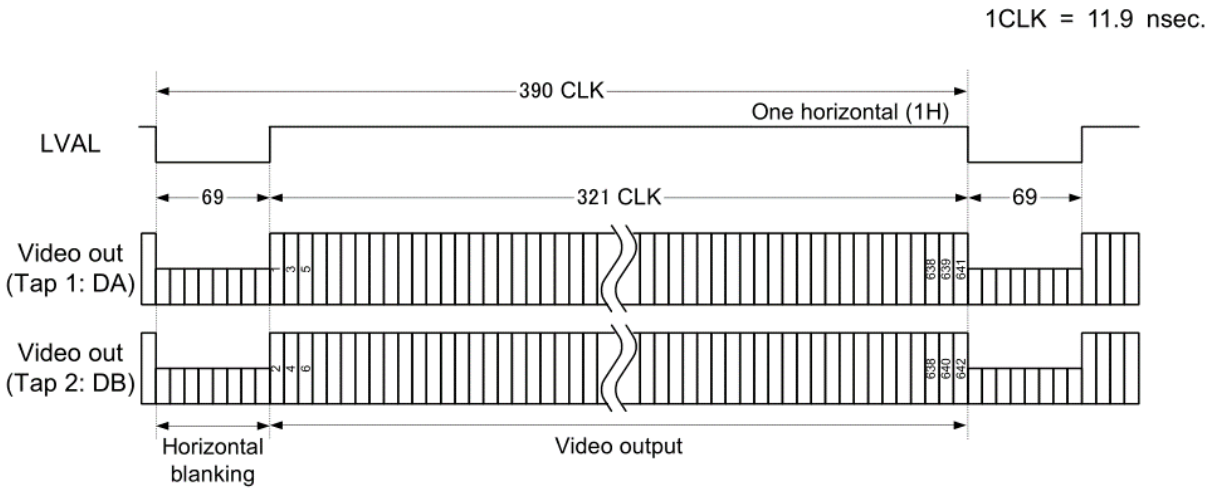


Abbildung 6: Übersicht Timing horizontal, Modi 2Tap, 642 Pixel horizontal (Quelle: SENTECH Product Spec)

„2 Tap“ bedeutet, dass eine Bildzeile aufgeteilt über zwei Kanäle übertragen wird. In Abbildung 6 werden über das Signal „Tap 1: DA“ alle ungeraden Pixel aus den gesamten 642 übertragen und in „Tap 2: DB“ alle geraden Pixel. Durch die parallele Übertragung wird der Datendurchsatz verdoppelt im Vergleich zur einfachen Datenübertragung bei gleichem Takt.

Diese Einstellung wird in der Kamera als „TAP Count“ eingestellt, oben beschrieben die Einstellung „TAP Count = 2Tap“. Darüber hinaus gibt es die Einstellung „3Tap“, welche die Übertragung auf drei Kanäle aufteilt. In diesem Projekt wird die Einstellung „2Tap“ verwendet.

Das Signal „LVAL“ zeigt mit einer steigenden Flanke den Beginn der Pixel an.

3.1.3 Vertikales Timing

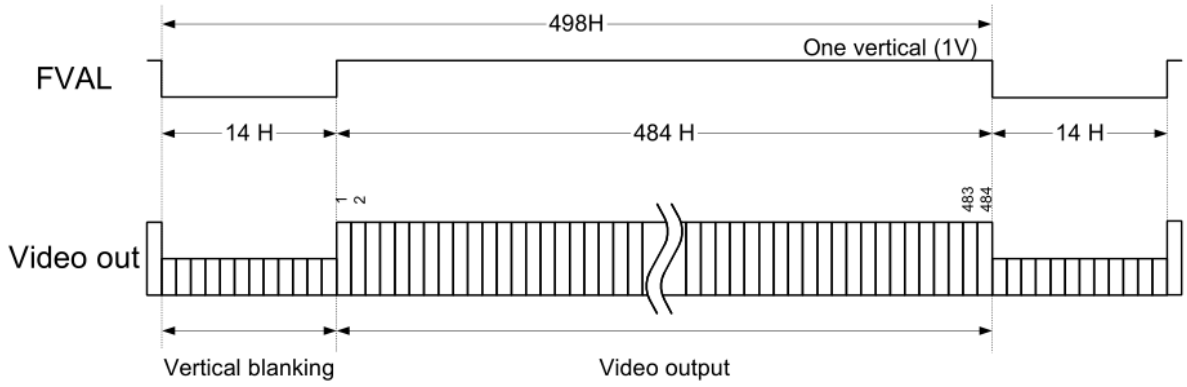


Abbildung 7: Übersicht Timing vertikal, 484 Zeilen vertikal (Quelle: SENTECH Product Spec)

In Abbildung 7 wird das Timing der vertikalen Zeilen erläutert. Wichtig beim Lesen der Abbildung ist die Einheit der Zeit; anders als bei Abbildung 6 ist hier nicht der Takt (CLK) Ausschlag gebend, sondern die Anzahl der vergangenen Zeilen. So steht „498H“ für 498 Zeilen, die vergangen sind. (H = horizontals).

3.2 Modul: camera_data_mux_gen

Das Modul vereint die Signale „TAP1“ und „TAP2“ aus Abbildung 5: Blockschaltbild in einem 4 Byte Shift Register. Eine detaillierte Erklärung des Moduls kann in der Ausarbeit (Herbst, 2020, S. 19) gefunden werden.

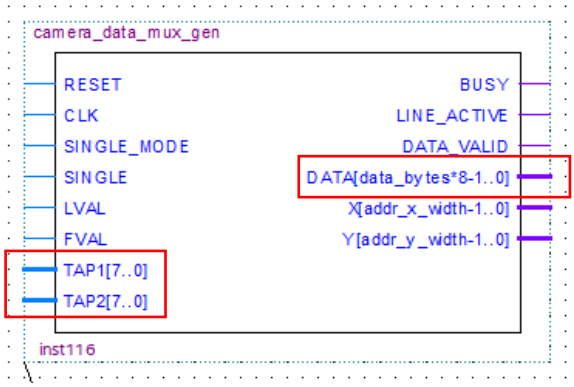


Abbildung 8: Blockschaltbild, (Quelle: Quartus Prime Lite)

Aufbau Intern: camera_data_mux_gen (Vereinfachte Version)

In der Abbildung 9 zu sehen, die interne Struktur des Moduls „camera_data_mux_gen“ für die Einstellung „TAP2“. Dies ist an der Verwendung von den Eingängen „TAP1“ und „TAP2“ zu erkennen. Die Einstellung „TAP3“ würde einen weiteren Eingang im Modul benötigen.

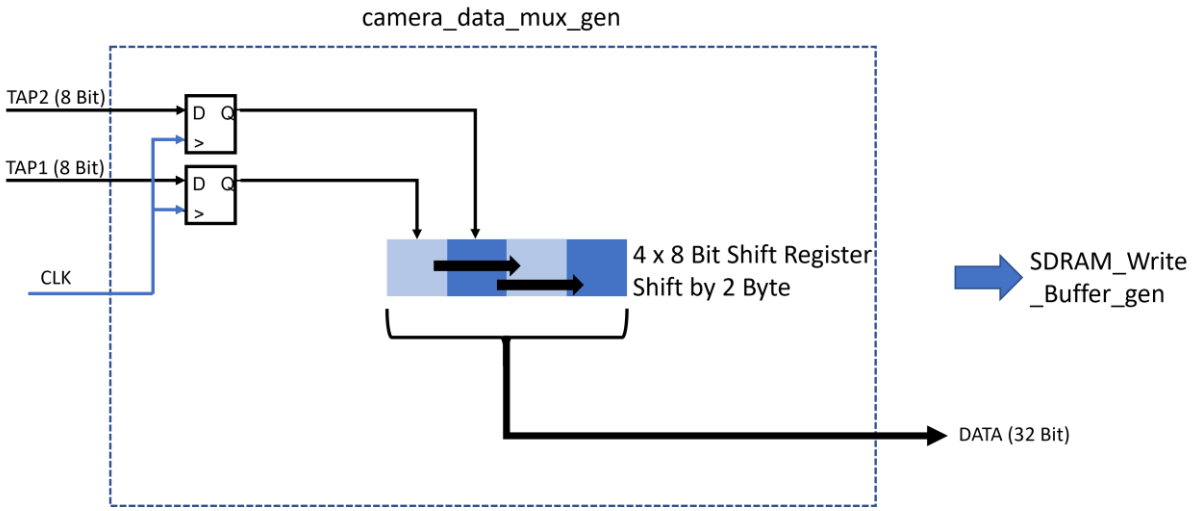


Abbildung 9: Vereinfachte Funktionsweise Modul „camera_data_mux_gen“

3.3 Modul: SDRAM_Write_Buffer_gen

Das Modul nimmt die vorverarbeiteten Kameradaten aus dem Modul „camera_data_mux_gen“ entgegen und schreibt diese mithilfe des Moduls „Memory Access Control“ in den SDRAM der sich auf dem FPGA Board befindet. Es ist die Schnittstelle zwischen dem Kameradatenbereich und den SDRAM-Bereich. Beide Bereiche sind unterschiedlich getaktet, daher benötigt das Modul aus beiden Bereichen den Takt. So werden die Eingangsdaten der Kamera mit dem Takt „cam_clk“ verarbeitet und die Synchronisation mit dem Modul „Memory Access Control“ über den Takt „sdram_clk“ realisiert.

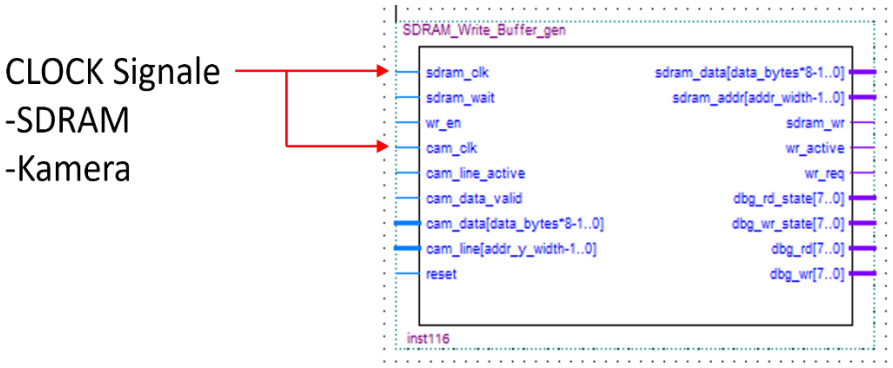


Abbildung 10: Blockschaltbild, (Quelle: Quartus Prime Lite)

Die Kameradaten werden in einem FIFO gepuffert, um dann bei freiem Schreibzugriff auf den SDRAM diese dort zu speichern. Der Ausgang „sdram_data“ sind die Kameradaten, die in den SDRAM geschrieben werden und „sdram_addr“ gibt die Position im SDRAM vor. In der aktuellen Form werden jeweils zwei Byte pro steigender Taktflange aus dem FIFO in den SDRAM geschrieben.

Aufbau Intern: SDRAM_Write_Buffer_gen (Vereinfachte Version)

Abbildung 11 zeigt den internen Aufbau von dem Modul „SDRAM_Write_Buffer_gen“ und beschreibt die wichtigsten Signale für die Kommunikation mit dem Modul „Memory_Access_Control“.

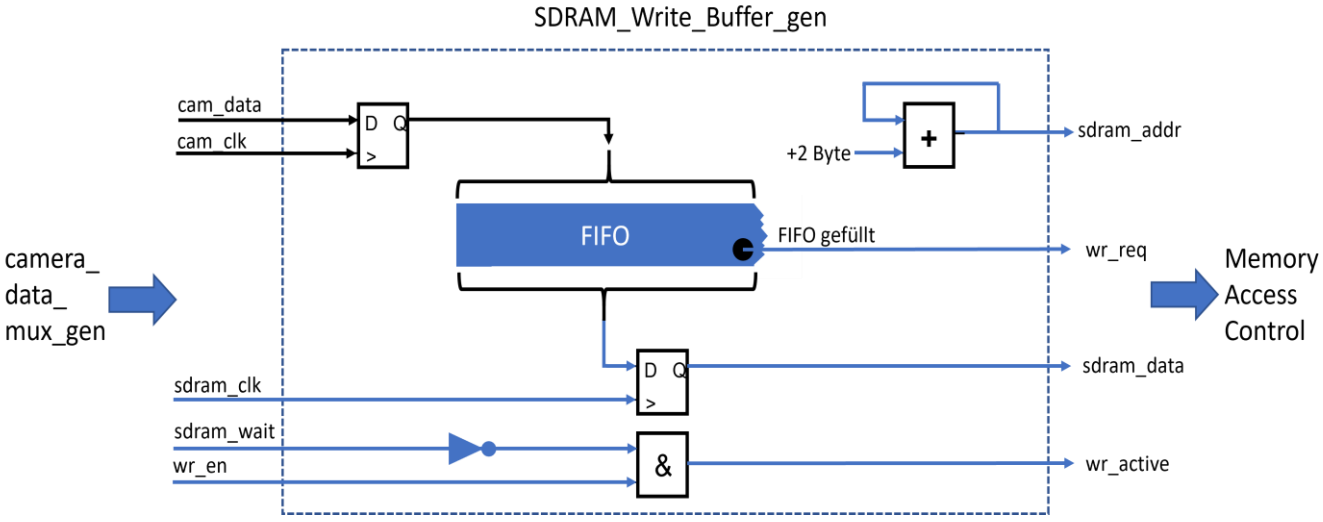
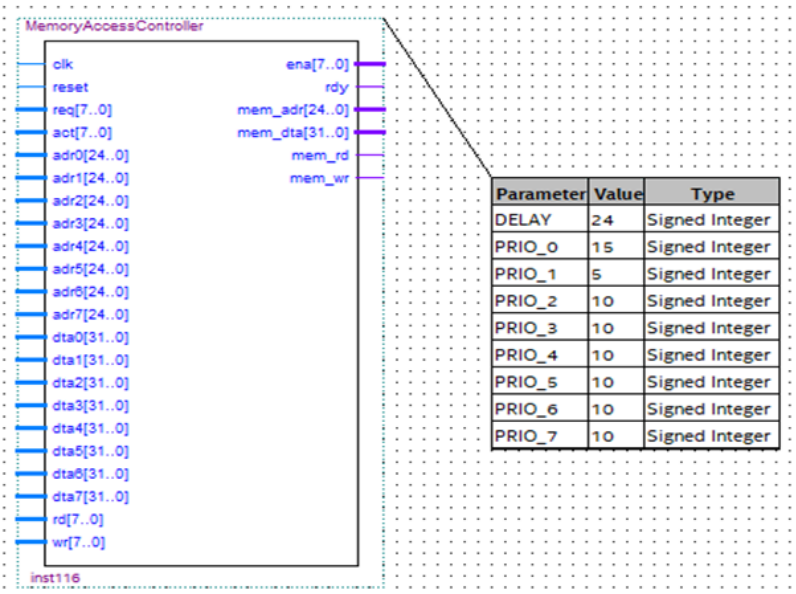


Abbildung 11: Vereinfachte Funktionsweise Modul „SDRAM_Write_Buffer_gen“

3.4 Modul: Memory Access Control

Das Modul regelt die Lese- und Schreibzugriffe von mehreren Quellen auf den gleichen SDRAM. Das Modul bietet acht verschiedene Plätze an seinem Interface an. In Abbildung 12 ist die Beschriftung der Eingangsgrößen dargestellt, die von einer Quelle gesteuert werden.



Parameter	Value	Type
DELAY	24	Signed Integer
PRIO_0	15	Signed Integer
PRIO_1	5	Signed Integer
PRIO_2	10	Signed Integer
PRIO_3	10	Signed Integer
PRIO_4	10	Signed Integer
PRIO_5	10	Signed Integer
PRIO_6	10	Signed Integer
PRIO_7	10	Signed Integer

Abbildung 12: Blockschaltbild „MemoryAccessController“ mit Parametern, (Quelle: Quartus Prime Lite)

Das Interface selbst besteht aus den folgenden Signalen:

- req[7..0]: steht für „request“, Zugriffsanfrage auf den Speicher
- act[7..0]: Zusage des Zugriffs auf den Speicher
- adrx[24..00]: Adresse die gelesen oder geschrieben werden soll
- datx[31..00]: Data Input, 4 Byte zum Schreiben in den SDRAM
- rd[7..0]: steht für „read“, Lesezugriff signalisieren
- wr[7..0]: steht für „write“, Schreibzugriff signalisieren
- ena[7..0]: steht für „enable“, Zugriff erteilt signalisieren

Funktionsweise Interface

Eine Zugriffsanfragen wird mittels „req“ angefordert. Bleibt die Anforderung „DELAY“ Takte stehen, ohne dass eine Anfoderung mit höherer Priorität kommt wird „ena“ gesetzt. Darauf muss der anfordernde Baustein „act“ setzten, solange ein Zugriff erfolgt. Dieser Ablauf wird in Abbildung 13 zur Veranschaulichung dargestellt.

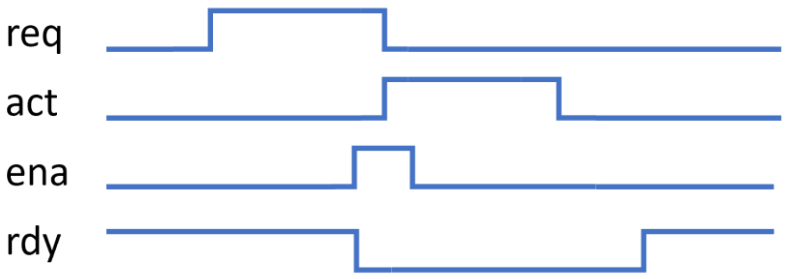


Abbildung 13: Funktionsweise Interface MemoryAccessController

3.5 Modul: SDRAM Controller

Ursprünglich entsprang das Modul aus einem SDRAM Controller Modul von Altera/Terasic und war in Verilog verfasst. Im Jahr 2018 wurde das Modul von Herrn Prof. Dr. Gick an der Hochschule Koblenz in VHDL verfasst. Das Modul ermöglich den direkten Lese- und Schreibzugriff auf die Hardware des SDRAM. Die interne Funktionsweise wird nicht weiter durchleuchtet, da es sich hier um sehr zeitkritische Vorgänge handelt würde dies den Ramen dieser kleinen Zusammenfassung übersteigen. Eine detaillierte Erklärung zur Funktionsweise eines anderen SDRAM Controller erhalten Sie unter http://www.geocities.ws/mikael262/sdram.

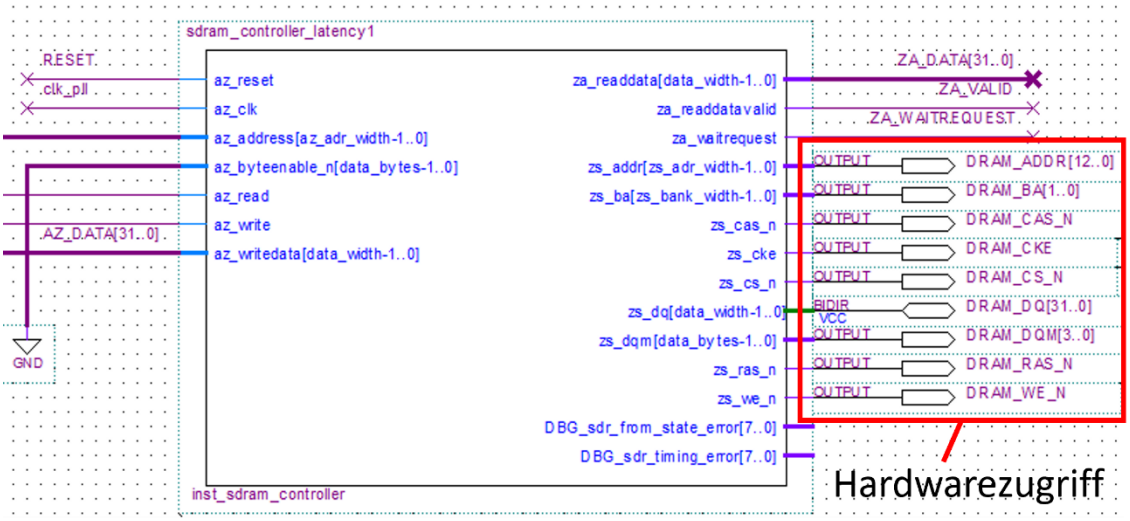


Abbildung 14: Blockschaltbild + Beschaltung, (Quelle: Quartus Prime Lite)

Um als Hardwareentwickler von dem Modul im Abbildung 14 Gebrauch zu machen ohne die Zugriffsmechanismen zur SDRAM Hardware zu verstehen, benötigen wir die folgenden Signale:

- az_clk: Clock Signal für die Ansteuerung des Moduls, nicht Takt für den SDRAM!
- az_address: Adresse für Schreibe- oder Lesezugriff
- az_read: Vorgang ist ein Lesevorgang bei logischen High-Pegel
- az_write: Vorgang ist Schreibvorgang bei logischen High-Pegel
- az_writedata: Daten die in den SDRAM zu schreiben sind
- za_readdata: Daten die aus dem SDRAM gelesen wurden
- za_readdatavalid: Daten in „za_readdata“ sind gültig bei einem logischen High-Pegel
- za_waitrequest: Bei einem logischen High-Pegel werden keine Lese- und Schreibanforderungen bearbeitet, SDRAM ist beschäftigt

3.6 Modul: SDRAM_Read_Buffer_gen

Das Modul „SDRAM_Read_Buffer_gen“ stellt das Gegenstück des Moduls „SDRAM_Write_Buffer_gen“ zum Schreiben der Bilddaten in dem SDRAM dar. Mit diesem Modul können die Bilddaten wieder aus dem SDRAM gelesen und für die weitere Verarbeitung bereitgestellt werden.

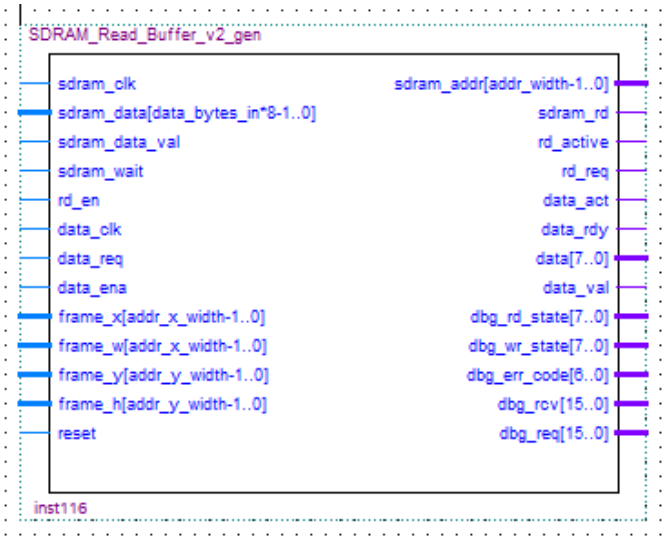


Abbildung 15: Blockschaltbild, (Quelle: Quartus Prime Lite)

Interne Buffer Struktur

Die SDRAM Daten werden in FIFO Buffern gespeichert. Um den Datendurchsatz zu erhöhen, wird ein voll beschriebener FIFO dem Ausgang bereitgestellt und ein weiterer FIFO mit Daten aus dem SDRAM beschrieben. Dieser Vorgang ist in Abbildung 16 dargestellt.

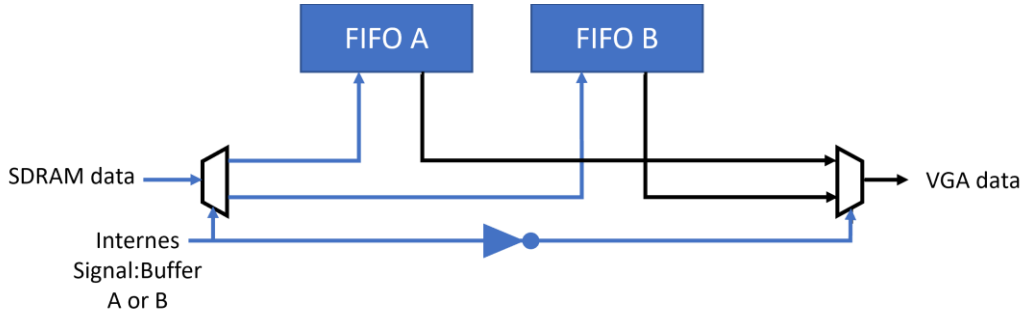


Abbildung 16: Interne Buffer Struktur des Moduls „SDRAM_Read_buffer_gen“

3.7 Modul: SDRAM_Pixelbuffer

Durch vorherige Schritte liegen Bilddaten im SDRAM vor. Die einzelnen Pixel werden jeweils als Byte dargestellt und sind einzeln aus dem SDRAM lesbar. Für die weitere Bildverarbeitung und die Ausgabe auf einer VGA Schnittstelle werden immer 5x5 Pixelblöcke im Modul „SDRAM_Pixelbuffer“ zwischengespeichert.

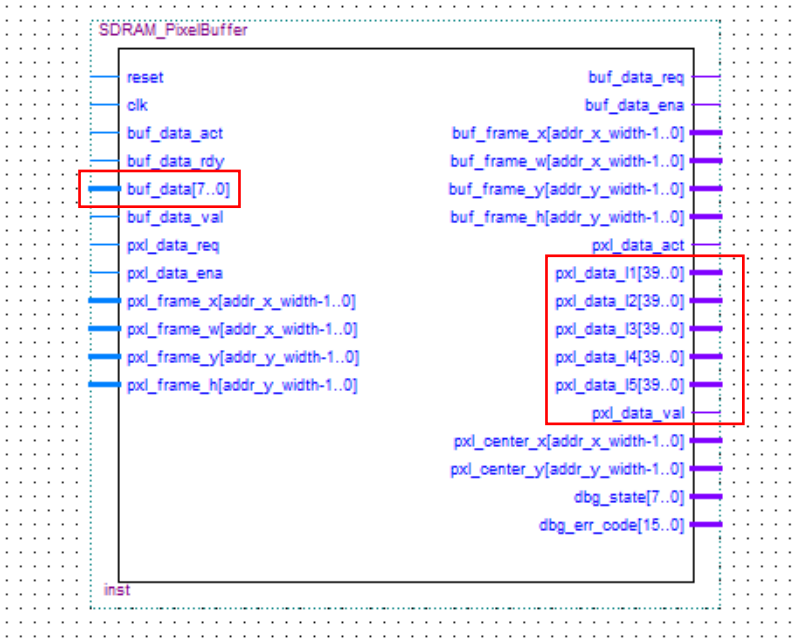


Abbildung 17: Blockschaltbild, rot markiert die wichtigsten Ios, (Quelle: Quartus Prime Lite)

Der 5x5 Byte Block ist zeilenweise auf den Ausgängen „pxl_data_l1[39 .. 0]“ bis „pxl_data_l5[39 .. 0]“ lesbar. Die Ausgänge dürfen erst gelesen werden, wenn der Ausgang „pxl_data_val“ einen logischen High-Pegel aufweist.

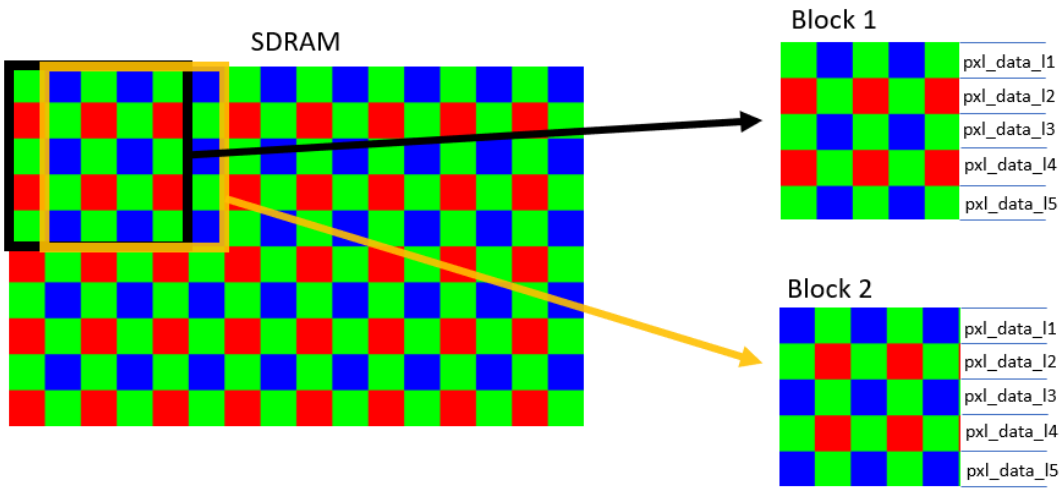


Abbildung 18: Schaubild Funktionsweise Pixelbuffer

Die fünf Ausgänge, für jeweils eine Zeile, sind auf den Eingang „pxl_datal1 ... pxl_datal5“ des Modul „debay“ gelegt für die Auswertung des Bayer Pattern.

Wenn ein Block komplett erfasst und bearbeitet wurde, wird der Block um eine Pixelspalte im Bild verschoben und der nächste Block wird gepuffert. Ist das Ende der aktuellen Spalte erreicht, dann wird der Buffer mit dem nächsten Block, am Spaltenbeginn um eine Zeile versetzt, befüllt. Siehe dafür Abbildung 19.

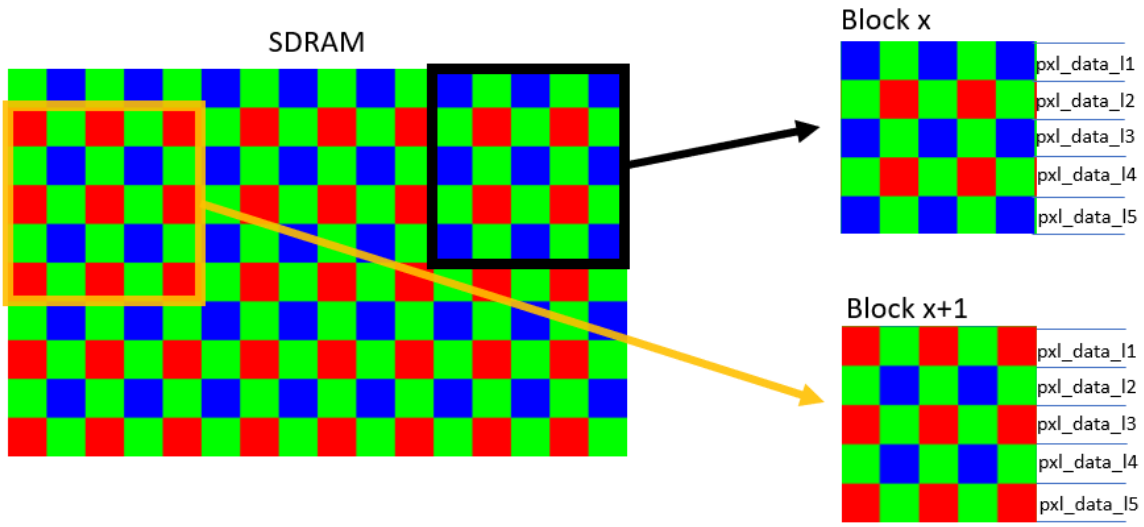


Abbildung 19 Schaubild Pixelbuffer Zeilenende

Aufbau Intern: SDRAM_Pixelbuffer(Vereinfachte Version)

Abbildung 20 zeigt die interne Verarbeitung der rot markierten Signale in Abbildung 17, da diese für die weitere Verarbeitung in den folgenden Modulen am wichtigsten waren.

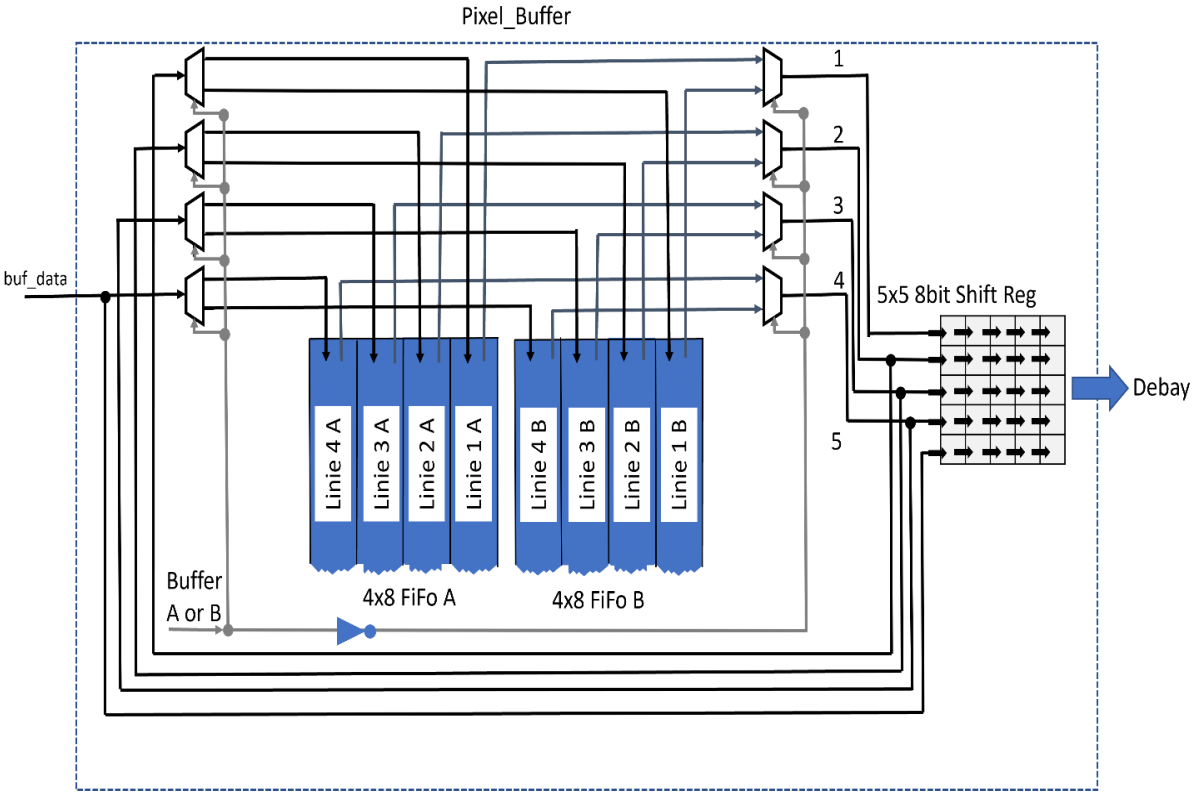


Abbildung 20 Schaubild interner Aufbau SDRAM_Pixelbuffer

3.8 Modul: Debay

Die Pixeldaten (5x5 Byte Block) des Modul „sdram_pixelbuffer“ liegen an den Eingängen „pxl_data_l1[39 .. 0]“ bis „pxl_data_l5[39 .. 0]“ an. Jeder Eingang steht für eine Zeile im Block.

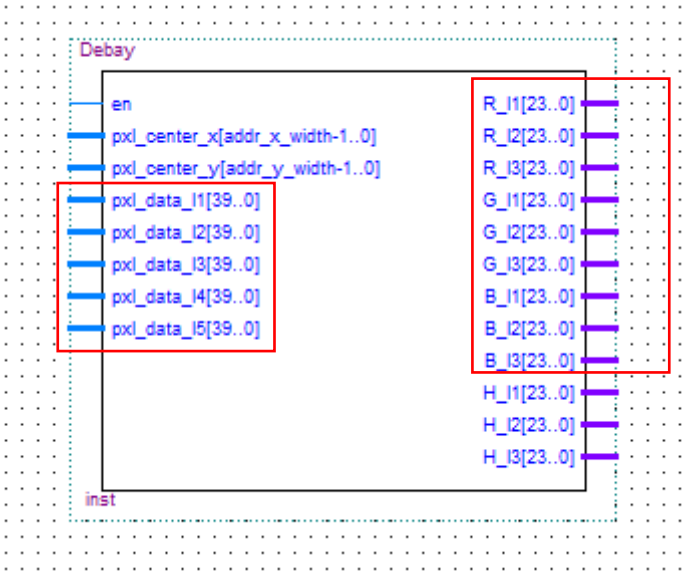


Abbildung 21 Blockschaltbild, (Quelle: Quartus Prime Lite)

Der 5x5 Byte Block ist nach dem Bayer Muster formatiert. Das Bayer Muster reduziert die Anzahl der Farbkanäle pro Pixel auf einen einzigen Farbkanal. Für die Ausgabe des Bildes über eine VGA-Schnittstelle sollen drei Kanäle pro Pixel zur Verfügung stehen. Im Modul „Debay“ wird aus dem 5x5 Byte Block für jeden der drei Farbkanäle (R, G, B), ein 3x3 Byte Block gewonnen. Um jeweils 3x3 Bytes pro Kanal zu bekommen, wird der 5x5 Byte Block wie in Abbildung 22 aufgeteilt.

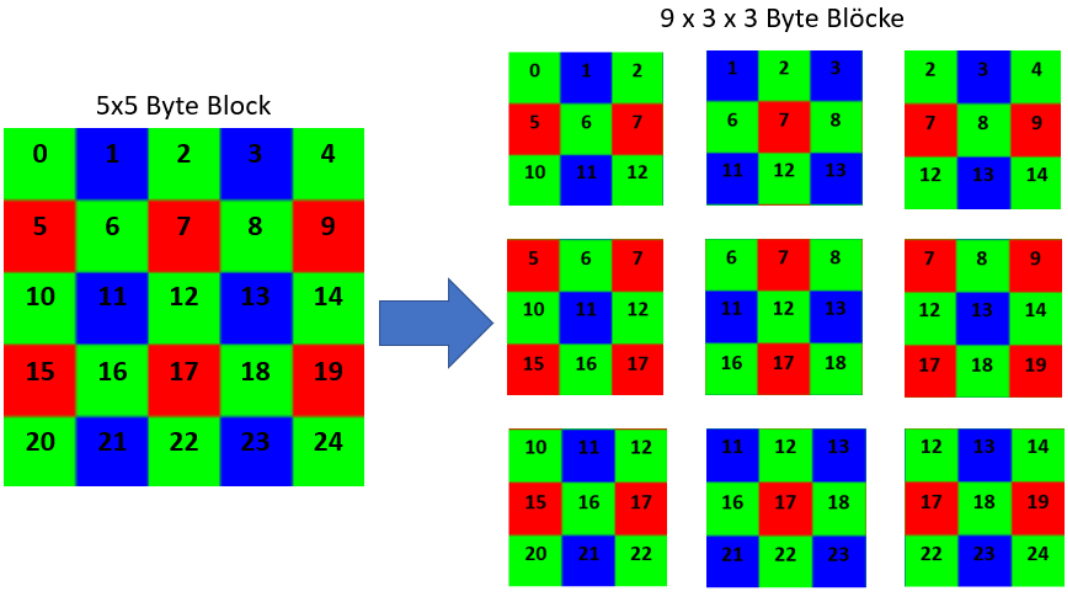


Abbildung 22 Aufteilung 5x5 Byte Block in kleinere 3x3 Blöcke

Durch die Aufteilung erhalten wir neun 3x3 Byte Blöcke. Aus jedem Block wird für alle drei Farbkanäle ein Byte extrahiert. Wie in Abbildung 22 zu sehen, beinhaltet ein 3x3 Block mehr als nur eine Farbinformation für einen Kanal, so ist beispielsweise im ersten Block an Stelle 5 und 7 die Farbinformation für rot doppelt. Um aus zwei Byte eines zu bekommen, nehmen wir das arithmetische Mittel der zwei Farbinformationen aus diesem Block. Zum besseren Verständnis dieser Vorgehensweise siehe Abbildung 23.

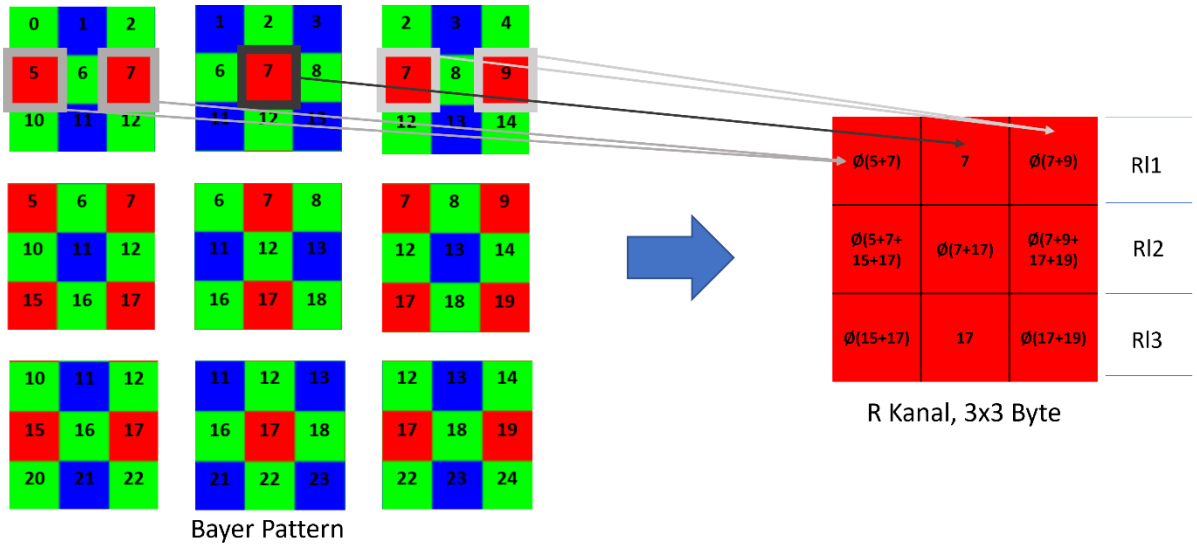


Abbildung 23: Zusammensetzung 3x3, Beispiel am Rotkanal, Rl1 = Output der ersten Zeile des Rotkanals

Nach dem Auswerten der kleineren Blöcke, werden die Informationen der 3x3 Blöcke pro Farbkanal, auf den, in Abbildung 21 rot markierten, Ausgängen gelegt.

Für die reine Ausgabe der Pixel über eine VGA Schnittstelle, würde ein Overhead bei der Anzahl der gelesenen Pixel entstehen, da 3x3 große Blöcke als Eingabe für das Decodieren des Bayer Pattern reichen würde. Jedoch ist die Größe von 5x5 als Eingabe gewünscht, da die zusätzlichen Pixel ein Maß an extra Information über Farbänderung und Farbbewegung liefern.

3.10 Modul: LaplaceFilter

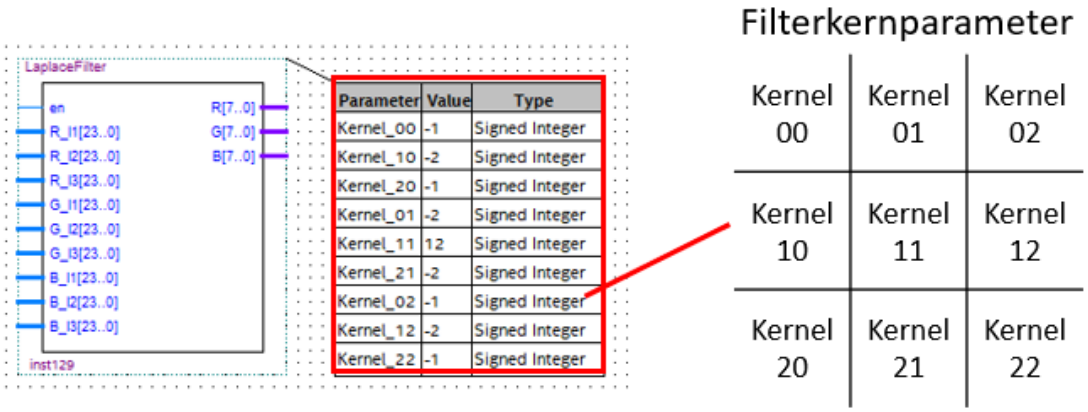


Abbildung 24: Blockschaltbild

Die Eingangsgröße des Blocks ist die Ausgangsgröße zu sehen in Abbildung 23, es handelt sich um jeweils 3x3 Byte Blöcke für jeden Farbkanal (RGB). Aus diesem Byte Blöcken wird jeweils nur ein Farbwert gewonnen und auf die Ausgänge „R[7..0], G[7..0] und B[7..0]“ gelegt.

Das Ziel des Blocks ist es, die in Abbildung 24 zu sehenden Filterkernparamter auf die am Eingang liegenden Farbkanäle anzuwenden. Die Anwendung von Filterkernen ist eine Technik aus der Bildverarbeitung, um gewünschte Details aus einem Bild zu extrahieren oder andere Bildeffekte herbeizurufen. In unserem Fall sollen alle Kanten im Bild extrahiert werden. Dafür wurde an dieser Stelle ein Laplace-Filter verwendet. Die Anwendung des Filters ist optional und kann über den Eingang „en“ ein- und ausgeschaltet werden.

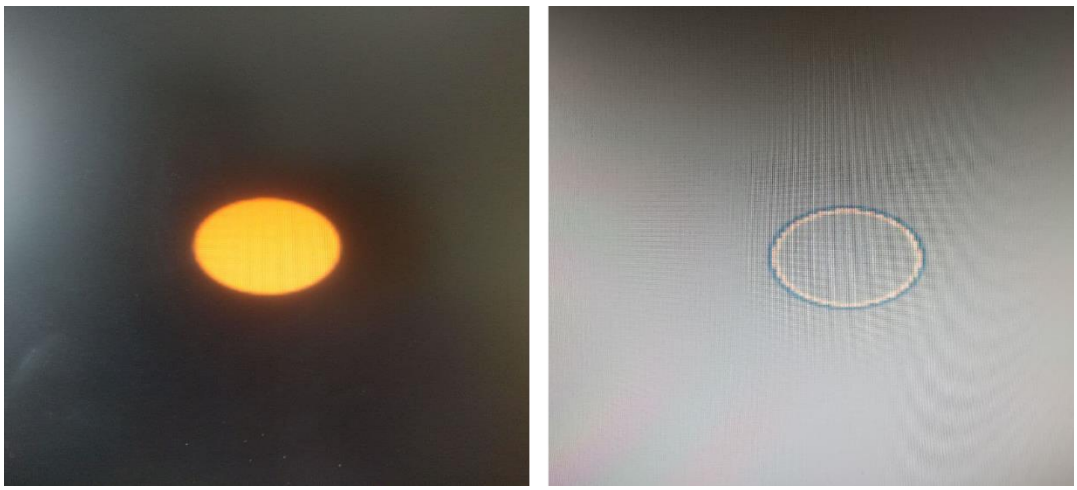


Abbildung 25: links roter Kreis ohne Filter, rechts gleicher Kreis mit Laplace-Filter

Abbildung 25 zeigt die Auswirkung des Laplace-Filter auf einem roten Testobjekt. Im Bild zu sehen ist das nur noch die Ränder des roten Kreises einen erheblichen Unterschied zum Rest des Laplace-gefilterten Bildes aufweisen. Mit dieser Information können die Umrisse eines farbigen Objektes erkannt werden.

Eine gute und detaillierte Erklärung zu Filter allgemein und speziell zum Laplace-Filter kann in einer Präsentation zum Thema „Bildverarbeitung: Filter“ von (Thormählen, 2020) nachgelesen werden. Im Folgenden wird der verwendete Laplacefilter anhand eines Beispiels erläutert.

Wird auf den Block in Abbildung 24 auf den Eingang „en“ ein logischer High-Pegel gelegt, so verhält der Ausgang „R[7..0]“ sich wie folgt. Das Prinzip wird gleichermaßen auf die beiden anderen Kanäle angewendet, jedoch zur Vereinfachung dient an dieser Stelle nur der Rotkanal als Beispiel.

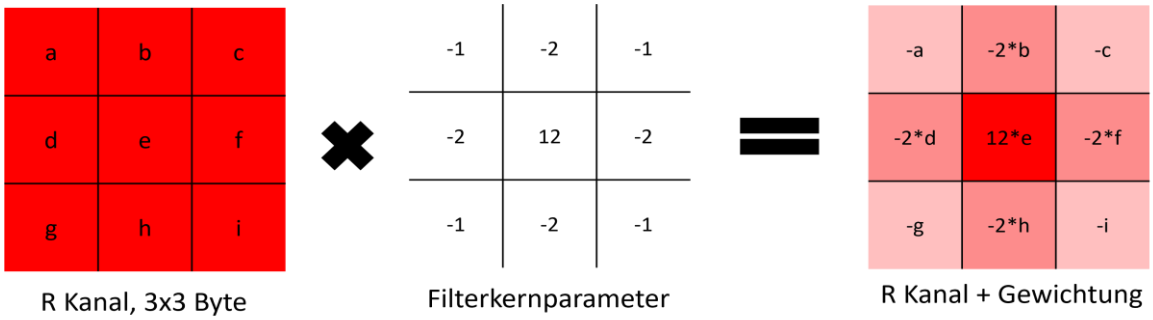


Abbildung 26: Gewichtung des 3x3 Rot Kanal für Berechnung Rot Kanal

In Abbildung 26 sieht man wie die Filterkernparameter auf den 3x3 Byte Block aufgeteilt und mit den einzelnen Werten verrechnet werden. Der Wert eines Filterkernparameters kann als eine Gewichtung für den jeweiligen Farbwert gesehen werden.

Die Gewichtung führt dazu, dass in Abbildung 26 der Wert „e“ am stärksten betont wird, wobei die Nachbarwerte weniger gewichtet werden. Abbildung 27 zeigt, wie aus dem gewichteten R Kanal Block ein einziger Wert wird.

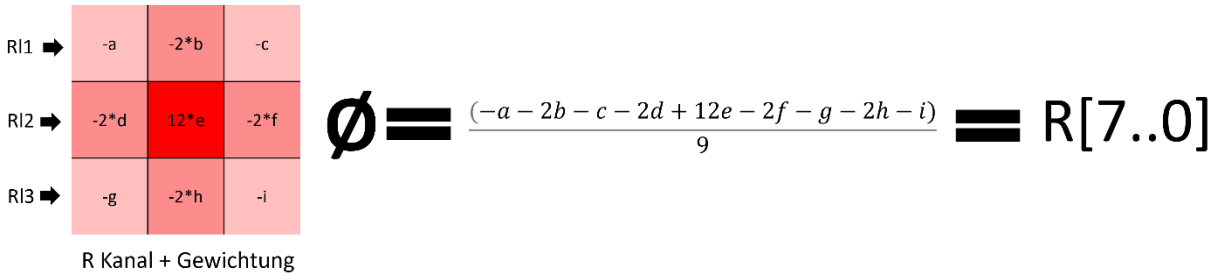


Abbildung 27: Berechnung Rotkanal, en=High-Pegel

Bei der Berechnung von „R[7..0] ist zu erkennen, dass wenn alle Nachbarwerte von „e“ den gleichen Wert aufweisen wie „e“ selbst, so ist das Ergebnis 0. Dies würde eine gleichmäßig rote Fläche darstellen, wie beispielsweise das Innere des roten Kreises aus Abbildung 25. Im Umkehrschluss bedeutet dies, dass Abbildung 27 nur einen hohen Wert aufweist, wenn die Nachbarwerte von „e“ sich groß unterscheiden. Dies zum Beispiel an den Kanten des roten Kreises aus Abbildung 25, der Wechsel von Schwarz auf Rot, erzeugt einen großen Unterschied. Wird auf den Block in Abbildung 24 auf den Eingang „en“ ein logischer Low-Pegel gelegt, so verhält der Ausgang „R[7..0]“ sich wie in Abbildung 28 und gibt die originalen Bilddaten ohne Veränderung weiter.

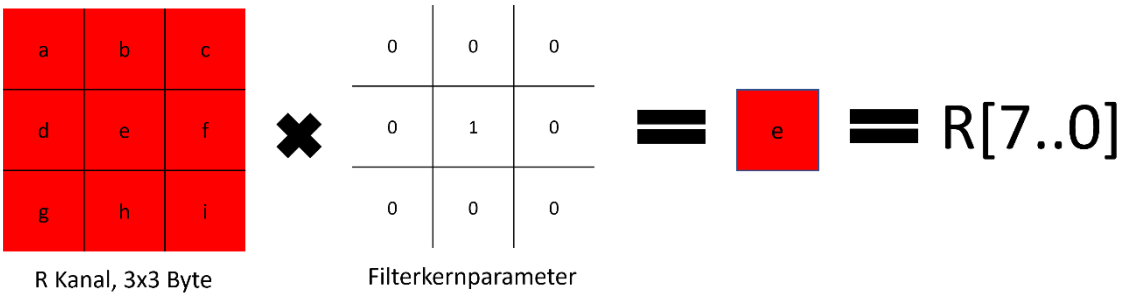


Abbildung 28: Berechnung Rotkanal, en = Low-Pegel

3.11 VGA Ausgabe

Für die Ausgabe der Bilddaten ist das Modul „vga_controller“ zuständig. Eine gute Übersicht über die VGA Spezifikation erhält man im Tech Forum der Firma Digi-Key zum Thema „VGA Controller (VHDL)“ (DigiKey, 2021), daher wird dieser Block hier nicht weiter erläutert.

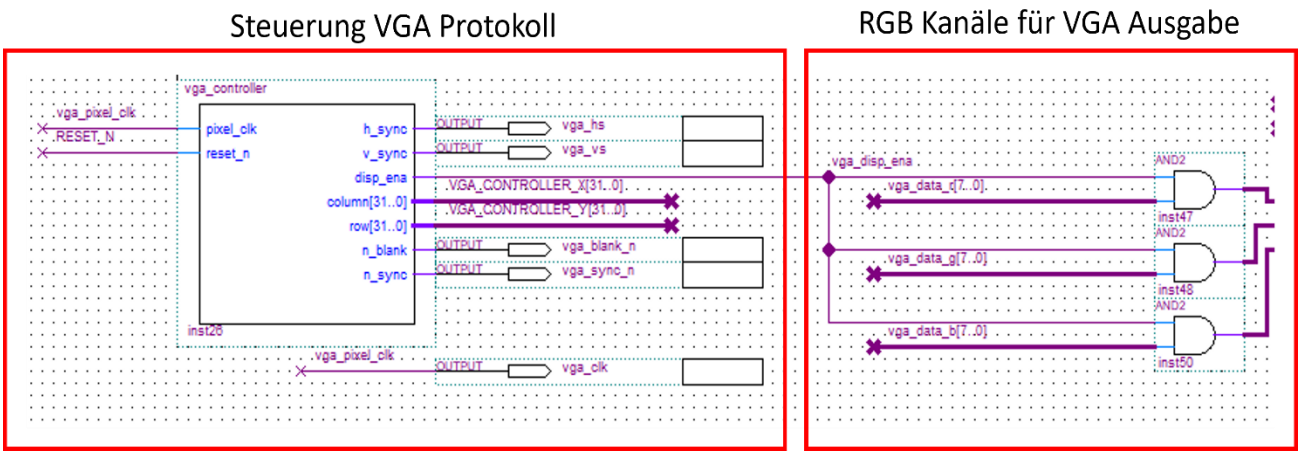


Abbildung 29: Schaltbild VGA Ausgabe, (Quelle: Quartus Prime Lite)

In Abbildung 29 sind alle Steuersignale für die VGA Schnittstelle zu sehen. Im rechten Bereich finden wir die Farbkanäle „vga_data_r“, „vga_data_g“ und „vga_data_b“, die unsere Bilddaten darstellen. Diese Signale kommen aus dem „LaplaceFilter“ Block.

4 Übersicht Zustand (nach Studienarbeit Oster)

Während der Bearbeitung wurde die Erkennung eines farbigen Objektes in seiner X- und Y-Achse angestrebt. Dies wurde mit gewissen Einschränkungen erreicht, diese Einschränkungen werden im Kapitel „Beinflussende Parameter“ erläutert. Als Beispiel für die Erkennung diente der Aufbau in Abbildung 30.

Abbildung 30: Testaufbau Erkennung rotes Rechteck auf Bildschirm

Die Kamera ist auf das rote Rechteck gerichtet. Innerhalb des Kamerablickwinkel soll die Position des Pixels gefunden werden, welches im Zentrum des Rechtecks sitzt. Die Position wird in Abhängigkeit der horizontalen und vertikalen Pixel dargestellt die maximal zur Verfügung stehen. Wenn das Objekt genau mittig im Sichtfeld der Kamera steht, so wird für die X-Achse eine Position von 320 und für die Y-Achse 240 ermittelt. Da Bilder in einer maximalen Gesamtauflösung von 640x480 Pixeln aufgenommen werden können.

Die bisherige Ausgabe auf einem VGA-Monitor wurde um ein Zielkreuz (in Grün) erweitert, um die bisher erkannte Position auszugeben, dies ist zu sehen in Abbildung 31.

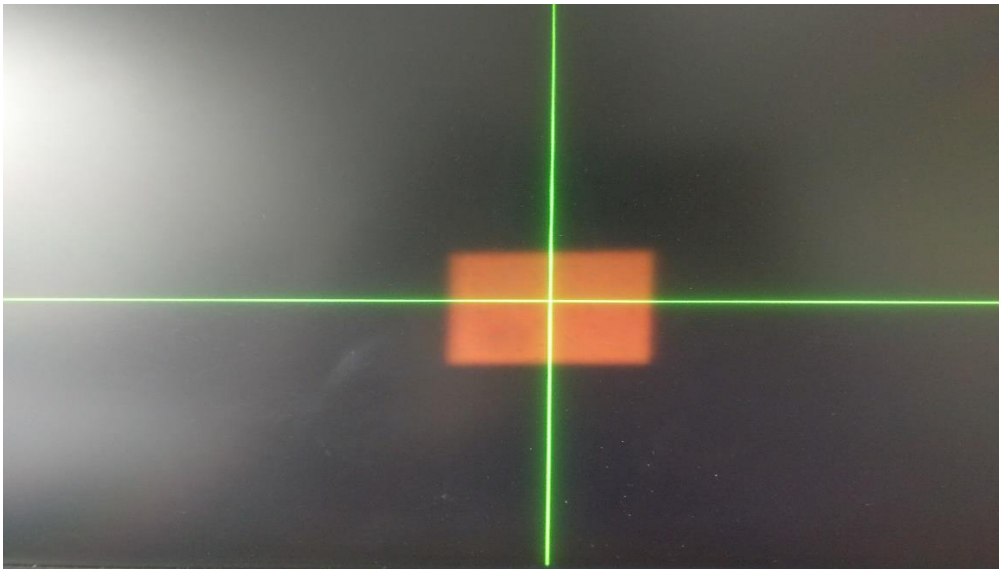


Abbildung 31: Zielkreuz auf erkannter Position

4.1 Funktionsweise/Anknüpfung an Vorheriges

Aus der vorhergegangenen Studienarbeit von (Herbst, 2020) ist können wir die Erkenntnis ziehen, dass wir aus dem Unterschied der Pixelintensität innerhalb einer Bildzeile, einfache Formen erkennen können. Diese Formen können ein einzelner Punkt oder gar eine ganze Linie darstellen. Mit dem Wissen, welche Formen sich in den Bildzeilen befinden, lassen sich komplexe Formen innerhalb eines ganzen Bildes wiederfinden.

Aus dem Abschnitt „Digitale Bildverarbeitung, Farbunterschiede erkennen“ ist bereits das Werkzeug für die Erkennung vorhanden. Mit diesem Werkzeug können, am Beispiel folgender Abbildung gezeigt, ein farbiges Objekt innerhalb eines ganzen Bildes erkannt werden.

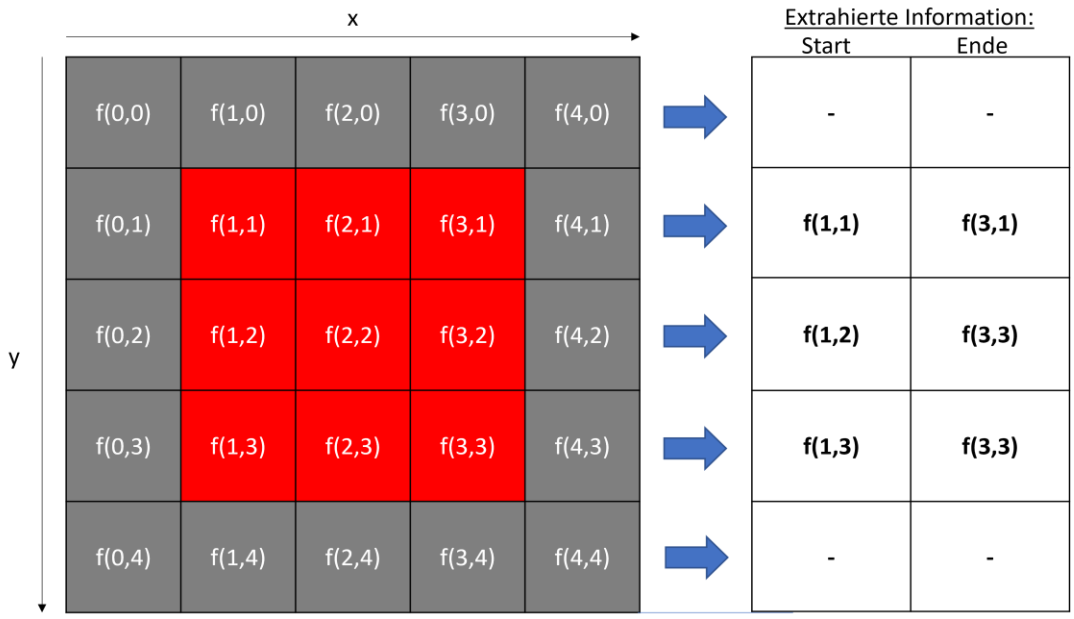


Abbildung 32: Beispiel Erkennung rotes Rechteck

Zum besseren Verständnis erklären wir die Funktionsweise der Erkennung von Start und Ende anhand der zweiten Zeile die mit f(0,1) beginnt aus Abbildung 32. Für die Erkennung stehen die RGB-Farbinformationen zur Verfügung, um starke Änderung zwischen Pixelnachbarn ausfindig zu machen. So können wir wie in Abbildung 32 die Änderung des Rotkanals betrachten und definieren ein Delta (THRESHOLD) beidem bestimmt werden kann, dass wir einen bestimmten Wechsel von wenig auf viel Rotanteil haben. Das Ganze in folgender Abbildung nochmal verdeutlicht.

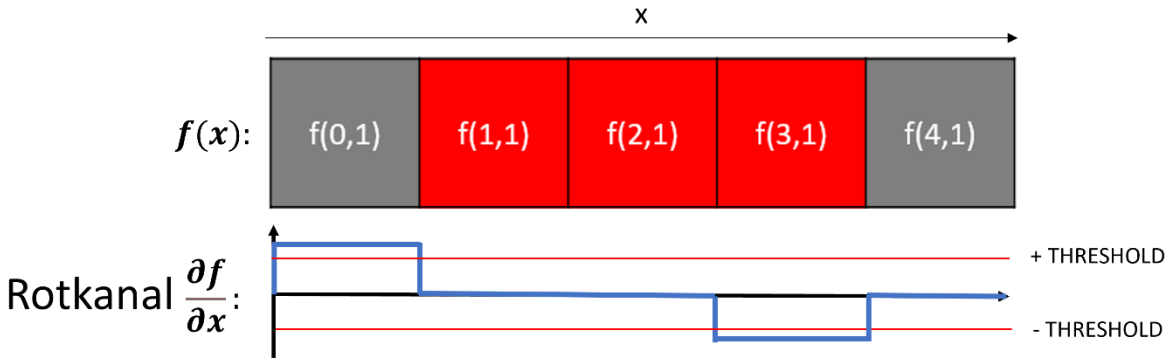


Abbildung 33: Änderung der Intensität des Rotkanals, 1.Ableitung nach Gonsalez

Abbildung 33 zeigt, dass Anfang und das Ende eines Objektes, über die Änderung der Farbintensität herauszufinden sind. Diese Daten beziehen sich immer nur auf eine Zeile und damit kann ein Objekt in erster Linie nur in seiner Breite identifiziert werden. Um die Höhe des Objektes auszumachen, können die Linien gezählt werden. Solange diese in der vertikalen Ebene, Nachbar sind, gehören Sie einem Objekt an. Mit der Information über Breite und Höhe des Objektes lässt sich der Mittelpunkt erfassen.

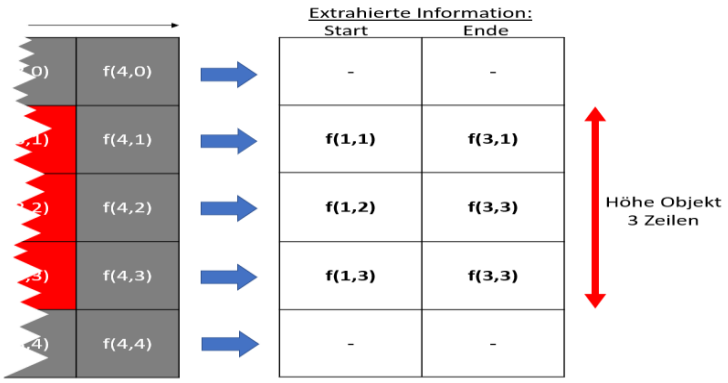


Abbildung 34: Zählen der Zeilen in dem ein Objekt erkannt wurde

Diese Funktion der zeilenweisen Erkennung von Start und Ende ist in dem Modul „LINE_DETECTION_CONV“ implementiert. Die Interpretation des gefundenen Starts und Ende geschieht im Modul „OBJ_DETECTION“. Für die Überprüfung des Ergebnisses wird die gefundene Position wie in Abbildung 31 auf der VGA Ausgabe angezeigt, dies ist im Modul „DEB_OBJ_DETECTION“ umgesetzt.

Verwendung Modul „LaplaceFilter“ aus Stand (Perske, 2019)

Die vorher beschriebene Erkennung, beruht auf den Erkenntnissen aus der Arbeit von (Herbst, 2020). In dem vorgefundenen Projekt von (Perske, 2019) wurden bereits die Grundsteine für eine zukünftige Kantenerkennung gelegt. So kann mit dem im Abschnitt „Modul: LaplaceFilter“ beschriebenen, ähnlich wie in Abbildung 32 ein Objekterkennung implementieren.

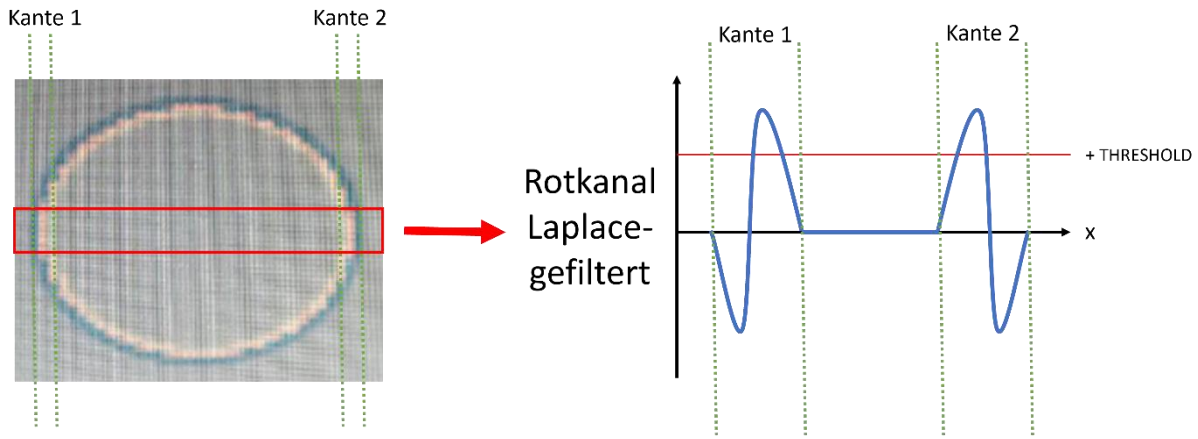


Abbildung 35: Beispiel Funktionsweise Kantenerkennung mit Hilfe von Laplace-Filter

In Abbildung 35 sieht man das Bild des Laplace-gefilterten roten Kreises aus Abbildung 25. In der Abbildung wird eine Zeile betrachtet und auf ihren Werteverlauf analysiert. Zu sehen, dass auch hier eine Überschreitung eines Threshold die Erkennung von Start und Beginn des Objektes möglich macht.

4.2 Übersicht aktueller Zustand

Mit den neu implementieren Features ergibt sich folgende Übersicht über die Projektmodule:

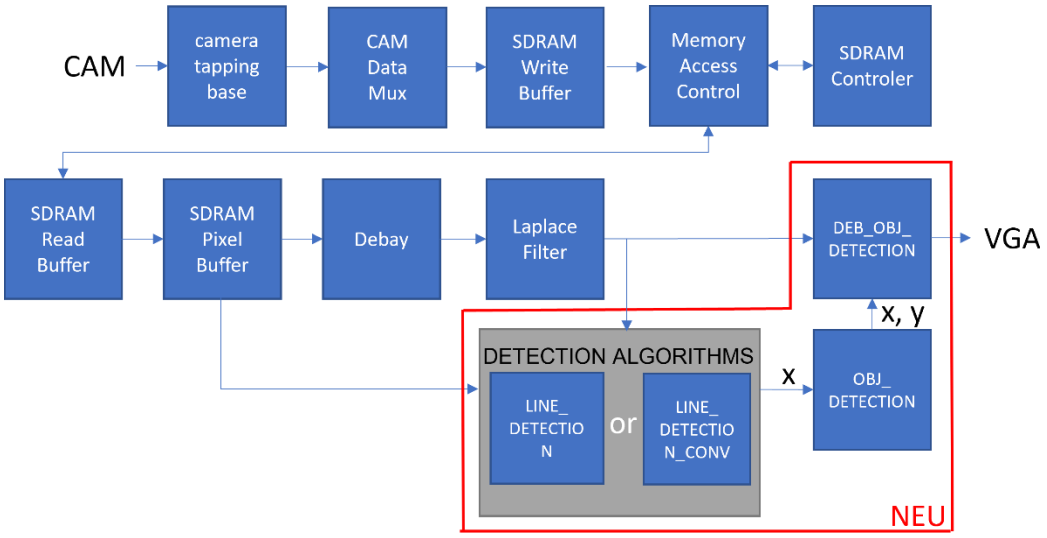


Abbildung 36: Übersicht Zustand Module nach Studienarbeit Oster,2021

Die neuen Module aus Abbildung 36, rot markiert, arbeiten auf den Ausgangssignalen des Moduls „LaplaceFilter“. In den folgenden Kapiteln werden die Module im Detail beschrieben. Im grau hinterlegten Kasten befinden sich zwei Implementierungen der zeilenweisen Objekterkennung. Das Modul „LINE_DETECTION“ beruht auf den Erkenntnissen von (Herbst, 2020) und das Modul „LINE_DETECTION_CONV“ auf den Erkenntnissen von (Perske, 2019). Beide Ansätze werden im Folgenden erläutert.

4.3 Modul: LINE_DETECTION(Herbst/Oster) und LINE_DETECTION_CONV(Perske/Oster)

Wie in Abbildung 33 dargestellt, übernimmt das Modul „LINE_DETECTION“ die Aufgabe, innerhalb einer Zeile Übergänge zwischen wenig und viel Rotanteil zu finden.[2] Daran werden der Start und Ende eines Objektes innerhalb der Zeile fest gemacht. Die Erkenntnisse, die für die Implementierung dieses Modul nötig waren kamen aus der Studienarbeit von (Herbst, 2020).

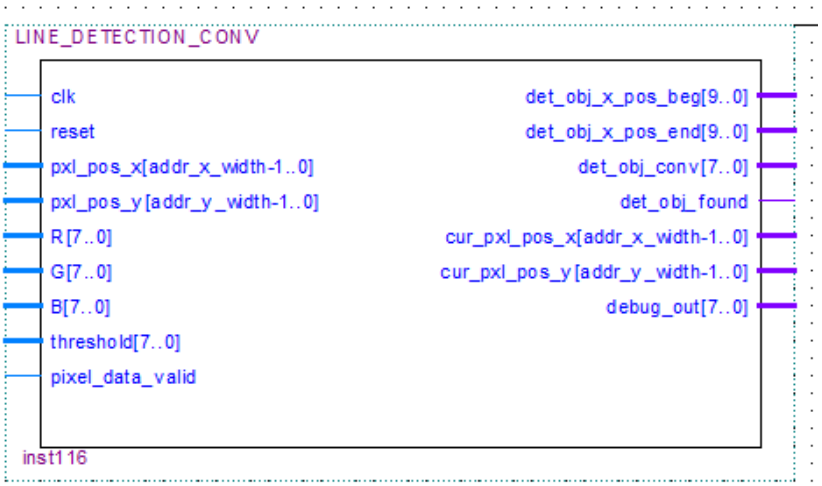


Abbildung 37: Blockschaltbild

Eingänge Modul:

- clk: Taktsignal, Takt mit dem das Modul „SDRAM_Pixelbuffer“ arbeitet
- reset: Zustandsreset innerhalb des Moduls, HIGH-Pegel für einen Takt
- pxl_pos_x: X Position des Pixels aus **Fehler! Verweisquelle konnte nicht gefunden werden.**, innerhalb eines Kamerabildes
- pxl_pos_y: Y Position des Pixels aus **Fehler! Verweisquelle konnte nicht gefunden werden.**, innerhalb eines Kamerabildes
- R,G,B: Pixel aus dem Modul „Convolution“, Rot-,Grün- und Blaufarbkanal, Farbtiefe 8 Bit
- threshold: Mindestunterschied der nötig ist für die Erkennung, siehe Abbildung 33
- pixel_data_valid: Pixeldaten die von Modul „SDRAM_Pixelbuffer“ bereit gestellt werden sind valide. HIGH-Pegel = valide

Ausgänge Modul:

- det_obj_x_pos_beg: Erkannter Start eines Objekts, valide wenn „det_obj_found“ = HIGH_PEGEL
- det_obj_x_pos_end: Erkanntes Ende eines Objekts, valide wenn „det_obj_found“ = HIGH_PEGEL
- det_obj_conv: Debug Signal, Ausgabe der berechneten 1.Ableitung nach Gonzalez
- det_obj_found: Objekt erkannt, „det_obj_x_pos_beg“ und „det_obj_x_pos_end“ sind valide
- cur_pxl_pos_x: Eingang „pxl_pos_x“ verzögert um einen Takt
- cur_pxl_pos_y: Eingang „pxl_pos_y“ verzögert um einen Takt
- debug_out: Debug Ausgabe für die Testbench oder 7-Segmentanzeige

[2] In der vorliegenden Fassung werden lediglich Änderungen der Rot-Anteile berechnet.

4.3.1 Interner Aufbau Modul „LINE_DETECTION“

Ein grober Einblick in die Funktionsweise bietet Abbildung 38. In der Abbildung fehlen ausgewählte Ein- und Ausgänge, damit die Funktionsweise einfach zu verstehen ist.

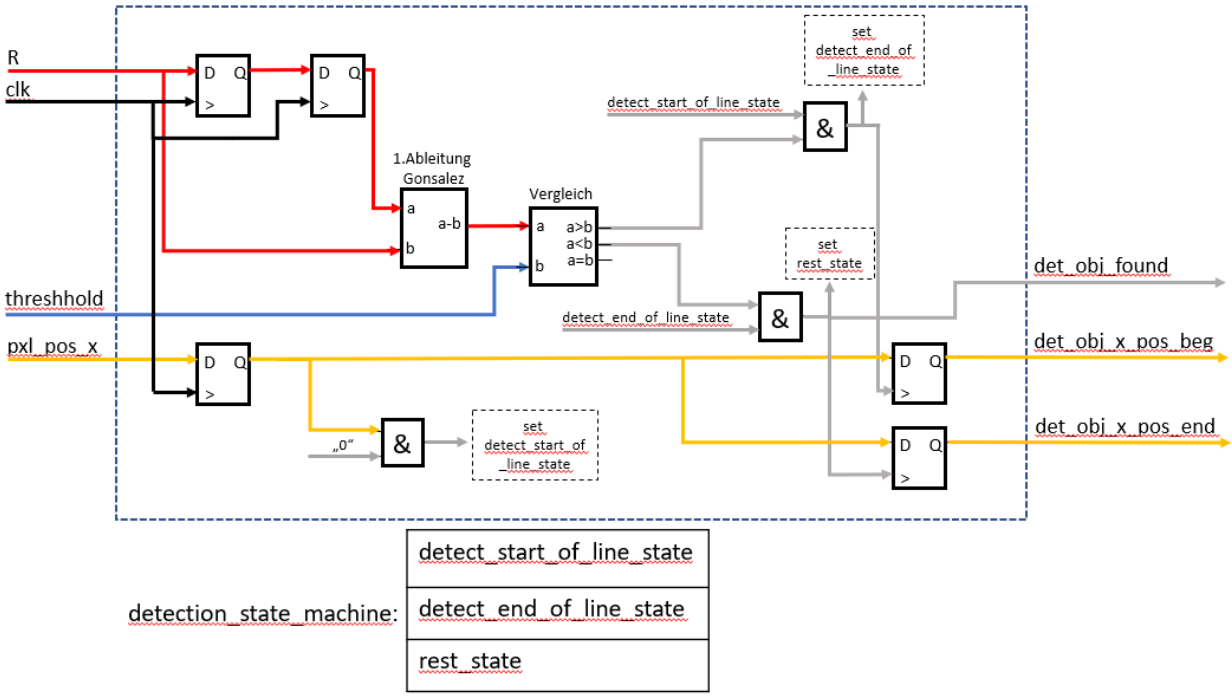


Abbildung 38: Vereinfachte Funktionsweise Modul „LINE_DETECTION_CONV“

Wie in Abbildung 38 zu sehen, wird derzeit nur der Rotkanal der Bilddaten für die Erkennung betrachtet. Im Modul stehen die Grün- und Blaukanalinformation zur Verfügung, da diese in einer Weiterentwicklung des Moduls hilfreich sind. So ließen sich anders farbige Objekte mit der gleichen Methode erkennen.

Der Threshold bezieht sich, in der Implementierung bei Veröffentlichung dieser Ausarbeitung, nur auf die Veränderung des Rotkanals. Das hat gewisse Nebeneffekte, die sich beim Testen herausgestellt haben. So wird eine starke Reflektion einer weißfarbigen Lichtquelle als Objekt erkannt, da im weißen Licht der Rotanteil sehr hoch ist im Vergleich zu einem dunklen Hintergrund. Die Lösung für dieses Problem wird, aufgrund des zeitlichen Rahmens, nicht weiter in dieser Arbeit thematisiert.

4.3.2 Testbench „LINE_DETECTION_CONV_TB“

Das Modul „LINE_DETECTION_CONV“ wurde mittels einer Testbench getestet und mit ModelSim simuliert. Die Testbench initialisiert innerhalb des eigenen Gerüsts das zu testende Modul und generiert die Eingangssignale. Damit können verschiedene Testszenarien erstellt werden, damit wird die Fehlersuche bei der Entwicklung erleichtert. Abbildung 39 zeigt die Ergebnisse der Simulation der Testbench.

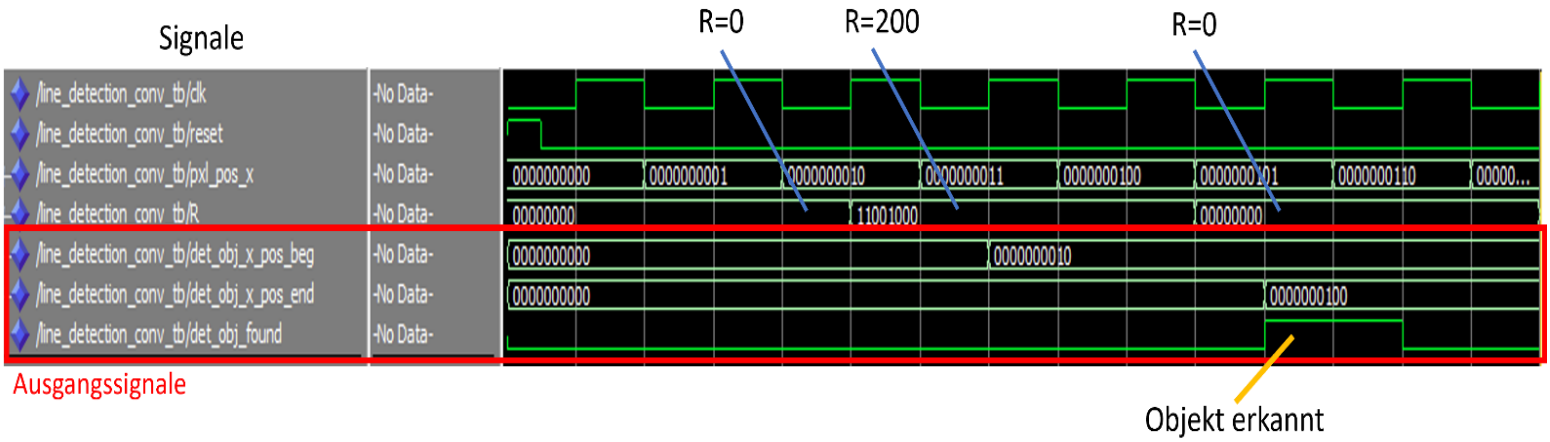


Abbildung 39: Auszug ModelSim, Simulation „LINE_DETECTION_CONV_TB“

Wir betrachten Abbildung 39, dort sind links die simulierten Signale verzeichnet und auf der rechten Seite der Verlauf der Simulation. Die Testbench inkrementiert das Signal „pxl_pos_x“ jeden Taktzyklus um 1. Nach zwei Taktzyklen ändert sich der Wert des Rotkanals, in Abbildung 39 als Signal „R“, von 0 auf 200. Dies stellt den Start eines roten Objektes dar. Es braucht drei weitere Taktzyklen bis der Rotkanal seinen ursprünglichen Wert von 0 erhält und damit das Ende eines roten Objektes darstellt. Das Signal „det_obj_found“ gibt an, dass die Signale „det_obj_x_pos_beg“ und „det_obj_x_pos_end“ valide sind, es steht für eine Taktperiode an.

4.3.3 Modul „LINE_DETECTION_CONV“

Im Gegensatz zum Modul „LINE_DETECTION“, arbeitet das Modul „LINE_DETECTION_**CONV**“ nicht mit den originalen Farbinformationen des Bildes, sondern sind diese vorher bereites Laplace-gefiltert. Diese Vorgehensweise beruht auf den Erkenntnissen aus der Arbeit von (Perske, 2019). Intern funktionieren die beiden Module sehr ähnlich und so sind auch die Ein- und Ausgänge des Moduls identisch zu Abbildung 37. Jedoch anstelle der Berechnung der ersten Ableitung nach Gonsalez, kann der Wert der auf Eingang „R“ liegt direkt mit dem Threshold verglichen werden.

Da die Unterschiede zum Modul „LINE_DETECTION“ gering ausfallen, wird das Modul an dieser Stelle nicht detaillierter beschrieben. Für die genaue Funktionsweise der Erkennung, siehe Abbildung 35.

Die Erkennung mittels der Laplace-gefilterten Informationen kann im Projekt über den Schalter „SW[13]“ ein- und ausgeschaltet werden.

4.4 : OBJ_DETECTION

Das Modul „OBJECT_DETECTION“ interpretiert die vom Modul „LINE_DETECTION“ oder Modul „LINE_DETECTION_CONV“ stammenden Signale. Diese Signale beinhalten den Start und Ende eines erkannten Objektes innerhalb einer Bildzeile. Aus diesen Informationen lässt sich die Höhe eines Objektes bestimmten und letztendlich auch damit dessen Position in seiner X- und Y-Koordinate innerhalb des Kamerabildes, die Veranschaulichung dazu in Abbildung 34.

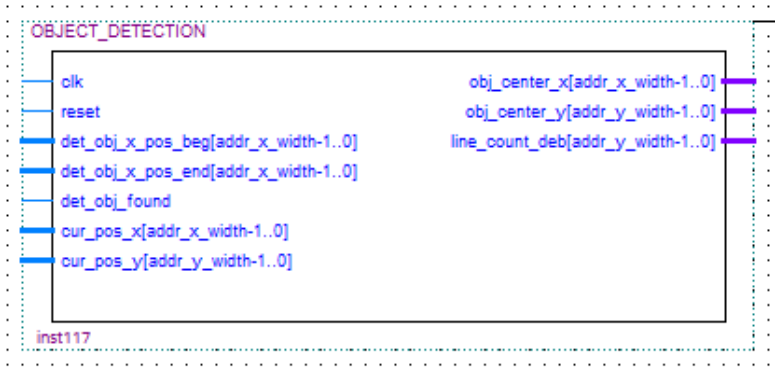


Abbildung 40: Blockschaltbild, (Quelle: Quartus Prime Lite)

Eingänge Modul:

- clk: Taktsignal, Takt mit dem das Modul „SDRAM_Pixelbuffer“ arbeitet
- reset: Zustandsreset innerhalb des Moduls, HIGH-Pegel für einen Takt
- det_obj_x_pos_beg: Signal „pxl_pos_x“ verzögert um einen Takt
- det_obj_x_pos_end: Signal „pxl_pos_y“ verzögert um einen Takt
- det_obj_found: Objekt erkannt, „det_obj_x_pos_beg“ und „det_obj_x_pos_end“ sind valide
- cur_pos_x: X Position des Pixels aus **Fehler! Verweisquelle konnte nicht gefunden werden.**, innerhalb eines Kamerabildes
- cur_pos_y: Y Position des Pixels aus **Fehler! Verweisquelle konnte nicht gefunden werden.**, innerhalb eines Kamerabildes

Ausgänge Modul:

- obj_center_x: X Position des erkannten Objekts, Mitte des Objektes
- obj_center_y: Y Position des erkannten Objekts, Mitte des Objektes
- line_count_deb: Debug-Ausgabe, Anzahl der Zeilen aus denen ein erkanntes Objekt besteht

4.4.1 Interner Aufbau Modul „OBJECT_DETECTION“

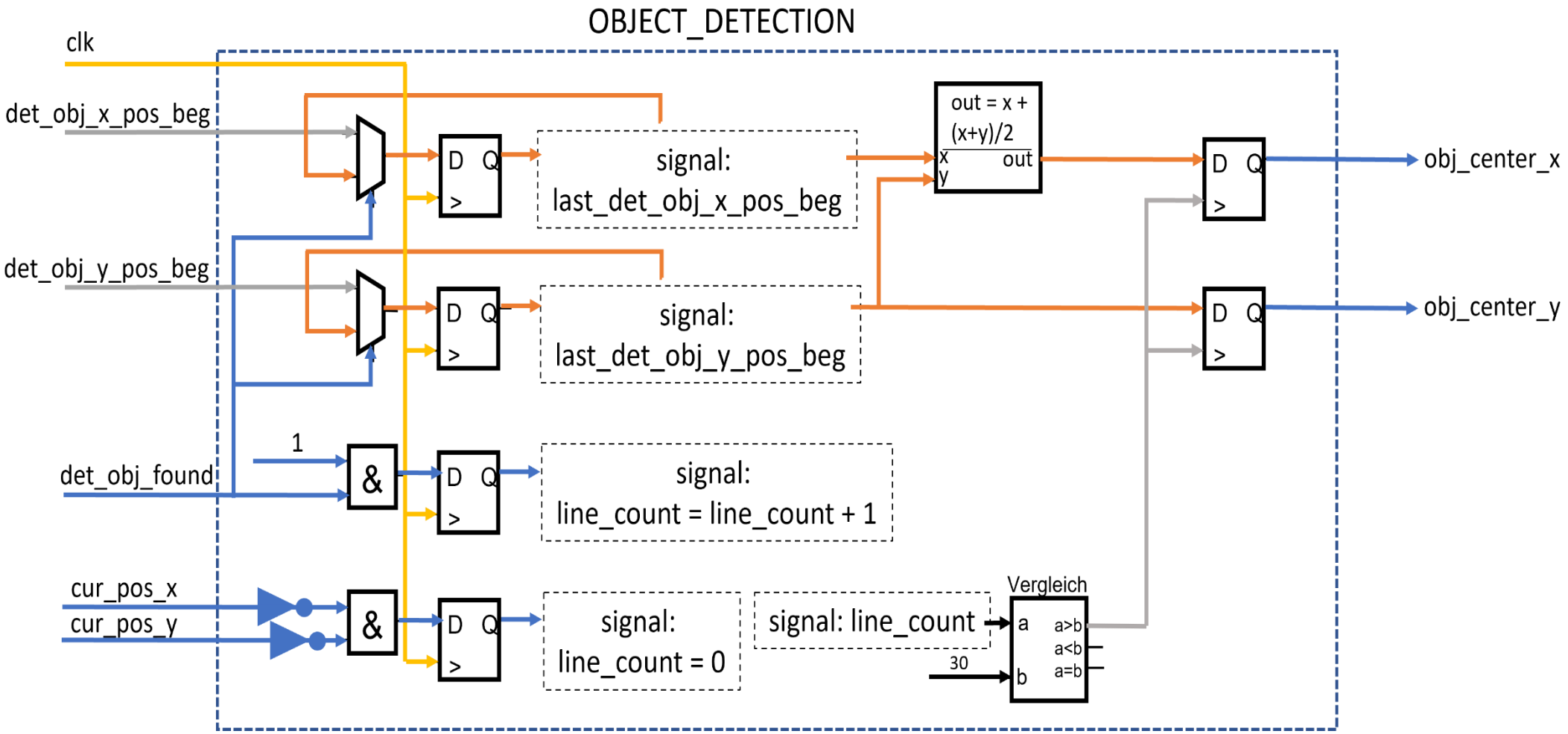


Abbildung 41: Vereinfachte Funktionsweise Modul „OBJECT_DETECTION“

4.5 Modul: DEBUG_OBJ_DETECTION

Das Modul „DEBUG_OBJ_DETECTION“ erzeugt ein Zielkreuz, um die erkannte Position des Moduls „OBJECT_DETECTION“ einfacher zu visualisieren. Das Zielkreuz ist zu sehen in Abbildung 31.

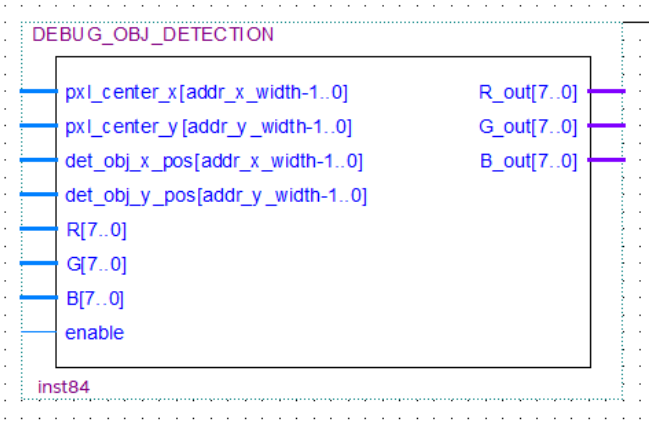


Abbildung 42: Blockschaltbild, (Quelle: Quartus Prime Lite)

Eingänge Modul:

- pxl_center_x: X Position des gelesenen Pixels aus Modul „SDRAM_Pixelbuffer“
- pxl_center_y: Y Position des gelesenen Pixels aus Modul „SDRAM_Pixelbuffer“
- det_obj_x_pos: X Position des erkannten Objekts, Mitte des Objektes
- det_obj_y_pos: Y Position des erkannten Objekts, Mitte des Objektes
- R,G,B: Pixel aus dem Modul „Convolution“, Rot-,Grün- und Blaufarbkanal, Farbtiefe 8 Bit
- enable: Zielkreuz aktiviert = log. HIGH-Pegel, deaktiviert = log. LOW-Pegel

Ausgänge Modul:

- R_out, G_out, B_out: Manipulierte Pixel für die VGA Ausgabe

Das Modul sitzt vor der Ausgabe auf der VGA Schnittstelle und nimmt Einfluss auf die Farbkanäle, um das Zielkreuz zu erzeugen.

Interner Aufbau Modul „DEBUG_OBJECT_DETECTION“

Abbildung 43 zeigt die grobe Funktionsweise des Moduls „DEBUG_OBJECT_DETECTION“. Sobald das Signal „pxl_center_x“ und „det_obj_x_pos“, repräsentativ für die X-Achse, übereinstimmen, werden die Farbsignale am Eingang ignoriert. Am Ausgang wird der Rot- und Blaukanal auf den Farbwert null gesetzt, jedoch der Grünkanal wird auf den Maximalwert von 255 gesetzt. Dies resultiert in einem stechenden Grün für den betroffenen Farbwert.

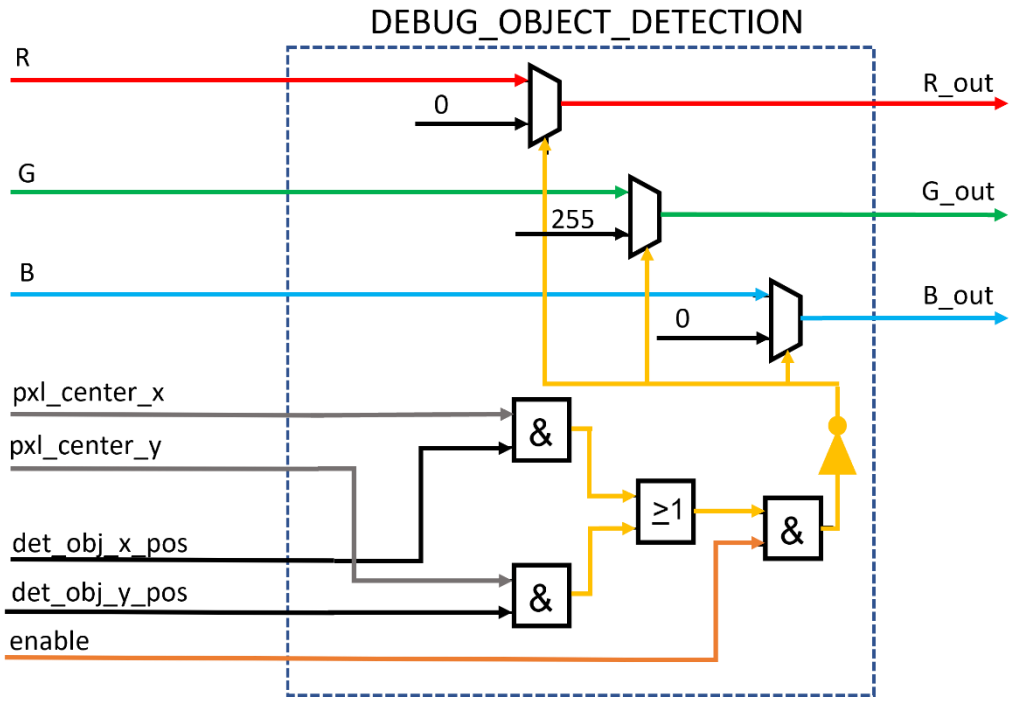


Abbildung 43: Vereinfachte Funktionsweise Modul „DEBUG_OBJECT_DETECTION“

5 Beeinflussende Parameter

5.1 Beleuchtung

Der Punkt „Beleuchtung“ ist ein stark beeinflussender Faktor für die Erkennung eines farbigen Objektes. Die Erkennung basiert auf dem Erfassen von Farbunterschieden. Diese Unterschiede werden durch die Menge an Licht die in den Sensor fallen und die Belichtungszeit, hauptsächlich definiert.

Wenn sehr wenig Licht in die Kamera fällt und zugleich eine sehr kleine Belichtungszeit gewählt wird, so wird der Unterschied zwischen verschiedenen Farben auch sehr gering ausfallen. Und damit eine Erkennung kaum möglich sein.

Die Menge an Licht ist durch die Lichtquellen in der Umgebung definiert. Es ist hilfreich die Szene, auf der sich das zu erkennende Objekt befindet, gut auszuleuchten. Es gibt in beide Richtungen ein Extrem, dies sind Unter- und Überbelichtung, siehe Abbildung 44, welche beide vermieden werden sollten.

Abbildung 44: Beispiel zu Beleuchtung, (Quelle: www.flocutus.de)

Die Belichtungszeit gibt die Zeit an, die der Sensor mit Licht in Kontakt kommt, um ein Bild zu entwickeln, bis das nächste Bild aufgenommen wird. Grundsätzlich gilt, je länger belichtet wird, umso heller wird die Aufnahme. Eine Erhöhung der Belichtungszeit führt jedoch zu einer verringerten Anzahl pro Bilder, die innerhalb einer Sekunde aufgenommen werden, dies ist kritisch für eine schnelle Erkennung.

5.2 Einstellung Treshold

Die Einstellung „threshold“ gibt an, wie groß der Unterschied zwischen zwei Nachbarpixeln sein muss, damit eine Kante eines Objekt erkannt wird. Der Unterschied berechnet sich wie in Abbildung 2 gezeigt. Für die Einstellung des „threshold“ stehen 8 Bit zur Verfügung, daher 256 Werte. Da die Bestimmung des Wertes stark von dem Umgebungslicht abhängig ist und daher oft geändert werden muss, ist es sinnvoll den Wert nicht als statisches Signal einzubinden, sondern einfach änderbar in das Projekt einzubinden. In dieser Arbeit hat man sich auf die Verwendung eines einfachen ROM (Read Only Memory) festgelegt.

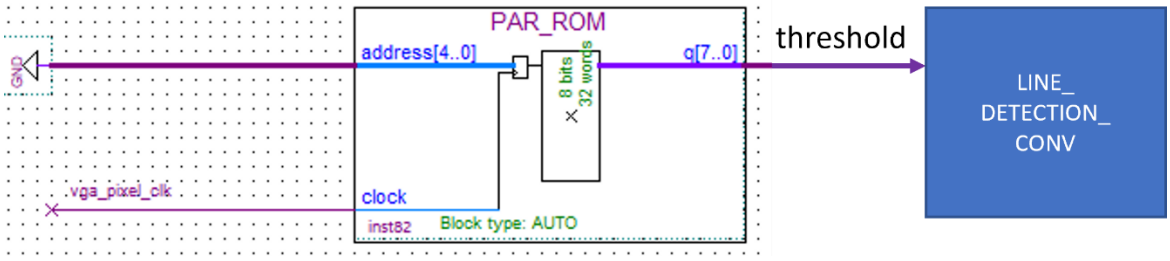


Abbildung 45: Read Only Memory Quartus Projekt, (Quelle: Quartus Prime Lite)

Am Eingang des ROMs wird die Adresse angegeben, die gelesen werden soll. In diesem Fall liegt der „threshold“ Wert an Adresse null. Jede Adresse beinhaltet 8 Bit an Daten, diese werden um einen Takt verzögert auf den Ausgang „q“ gelegt. Der Inhalt des Speichers kann über die Datei „PAR_ROM.mif“, welche dem Quartus Projekt angehört, bestimmt werden.

6 Ergebnis und Diskussion

6.1 Zusammenführen verschiedener Projektstränge

Zu Beginn dieser Arbeit existieren bereits zwei Projektstränge, die von zwei unterschiedlichen Personen bearbeitet wurden. Projektstrang A entstand unter Ausarbeitung einer Studienarbeit, durchgeführt von Herrn Herbst, Informationstechnik Student Hochschule Koblenz, zu Jahresbeginn 2020. Projektstrang B entstand unter der Arbeit von Herrn Perske als wissenschaftliche Hilfskraft, eingestellt von Herrn Prof. Gick an der HS Koblenz, im Frühjahr 2019. In beiden Strängen existiert die Schnittstelle über CameraLink an eine Hochgeschwindigkeitskamera und die Speicherung der Bilddaten. Die Gemeinsamkeit liegt in der Verwendung von gleichen VHDL Modulen, um die vorher genannten Aufgaben zu erledigen. Die Unterschiede zu den Projekten sah folgendermaßen aus:

Projektstrang A, Ausarbeitung Studienarbeit Herbst, Frühjahr 2020:

- Verwendete Kamera liefert Graustufenbilder
- Ausgabe einer Bildzeile auf VGA
- Erkennung isolierter Punkte in einem eindimensionalen Bild
- Erkennung von Lininenobjekten in einem eindimensionalen Bild

Projektstrang B, Ausarbeitung wissenschaftliche Hilfskraft Perske, Frühjahr 2019

- Verwendete Kamera liefert Farbbilder im Bayer Pattern
- Auswertung Bayer Pattern
- Berechnung Faltung

Aus diesen beiden Strängen entstand dieses Projekt den Einfluss aus beiden Projekten hatte. Aus der Studienarbeit von Herrn Herbst ging hervor, dass in einer Fortsetzung die Verarbeitung mit Farbbildern folgen könnte. *„So kann als nächste Schritte die Positions- und Objekterkennung in zweidimensionalen Bildern sowie die Verarbeitung von RGB-Farbbildern implementiert werden.“* (Herbst, 2020). Aufgrund dessen wurde als Grundlage der Projektteil von Herrn Perske als Ursprung verwendet und bei der Implementierung neuer Module, die Erkenntnisse aus der Studienarbeit von Herrn Herbst in Betracht gezogen.

6.2 Kontentmanagement

Die bisherigen Projektstände wurden nicht einheitlich gepflegt und ohne Richtlinien was den Speicherort und die Revisionierung angeht. Daher wurden im ersten Schritt die Projektstände refaktored. Das Refaktoring beinhaltete folgende Schritte:

- Entfernen unbenutzter und alter Revisionen im Quartus Projekt
- Entfernen von nicht verwendeten Ein- und Ausgängen in den bestehenden Modulen
- Entfernen nicht verwendeter interner Signale in den bestehenden Modulen, die zu einem Warning geführt haben
- Beseitigung von Warnings innerhalb der Quartus Projekte

Um das Problem des Speicherort zu beseitigen, wurde eine Ordnerstruktur für alle Projektrelevanten Dateien entworfen die wie folgt aussieht:

- Dokumentation (Beinhaltet alle Dokumente zu den Projekten)
    - Stand Oster
    - Stand Herbst
    - Stand Perske
    - Datenblätter

- QuartusProjekt
    - Stand Oster
    - Stand Herbst
        - Projektstand sauber
        - Projektstand unsauber
    - Stand Perske
        - Projektstand sauber
        - Projektstand unsauber

Alle Ordner unter Dokumentation enthalten jeden Datenfluss und Aufzeichnung, die nicht im direkten Zusammenhang mit Quartus stehen, so wird sich auch dieses Schriftstück unter dem „Dokumentation“ Ordner wieder finden.

Die „Projektstand unsauber“ Ordner beinhalten die exportierten Quartusarchive aus den Originalprojektständen, wie sie vorgefunden wurden. Die Ordner mit „sauber“ beinhalten die refactored Projektstände.

Zugänglich wird diese Struktur unter GitHub [3]unter der Adresse „https://github.com/Chriss95De/Studienarbeit.git“. Dort kann das ganze als ZIP Archive heruntergeladen oder wie ursprünglich gedacht, als Git[4]-Rep[5]ository weitergeführt werden.

[3] GitHub ist eine kommerzielle Plattform um Git Repositories zu hosten
[4] Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien (GIT SCM)
[5] Ein Repository beschreibt die Gesamtheit aller verwalteten Dateien und deren Änderungen

6.3 Erkennung von Linienobjekten innerhalb einer Bildzeile

Für die Umsetzung der Erkennung von Linienobjekten innerhalb einer Bildzeile wurden das Modul „LINE_DETECTION“ und das Modul „LINE_DETECTION_CONV“ entworfen. Die Module erkennen beliebig viele Objekte innerhalb einer Zeile, solange die Voraussetzung des Mindestunterschied (Threshold) erfüllt sind. So wird jeder Start und Ende eines Linienobjekt erkannt, sobald der Threshold erreicht wurde, siehe Abbildung 33 und Abbildung 35 zum Verständnis.

Die Erkennung unter der aktuellen Implementierung funktioniert am besten unter einer gut ausgeleuchteten Szene mit einem sehr dunklen Hintergrund, ohne großen Rotanteil und einem Objekt mit einem sehr hellen Rotfarbton. Innerhalb der Szene sollte eine zu punktuelle Beleuchtung vermieden werden, da es sonst zu Reflektionen und damit zu weißen Punkten in der Bildaufnahme kommt. Dies würde die Linienerkennung stören.

Eine ausführliche Erklärung kann in Kapitel 4.3 gefunden werden.

Problematiken

In den Überlegungen aus der Studienarbeit von (Herbst, 2020) und unter der Verwendung der ersten Ableitung nach Gonzalez, siehe Abbildung 2, wird immer von einem harten Übergang ausgegangen. So wäre ein Wechsel von einem schwarzen Hintergrund auf ein rotes Stück Papier ohne große Vermischung an den Rändern möglich, welches zu einer erhöhten Änderung der Intensität an diesen Übergang führen würde. In der Realität sind diese Übergänge oft ein Farbverlauf anstatt eines harten Übergangs. Dies wird deutlich, wenn wir uns Abbildung 46 ansehen.

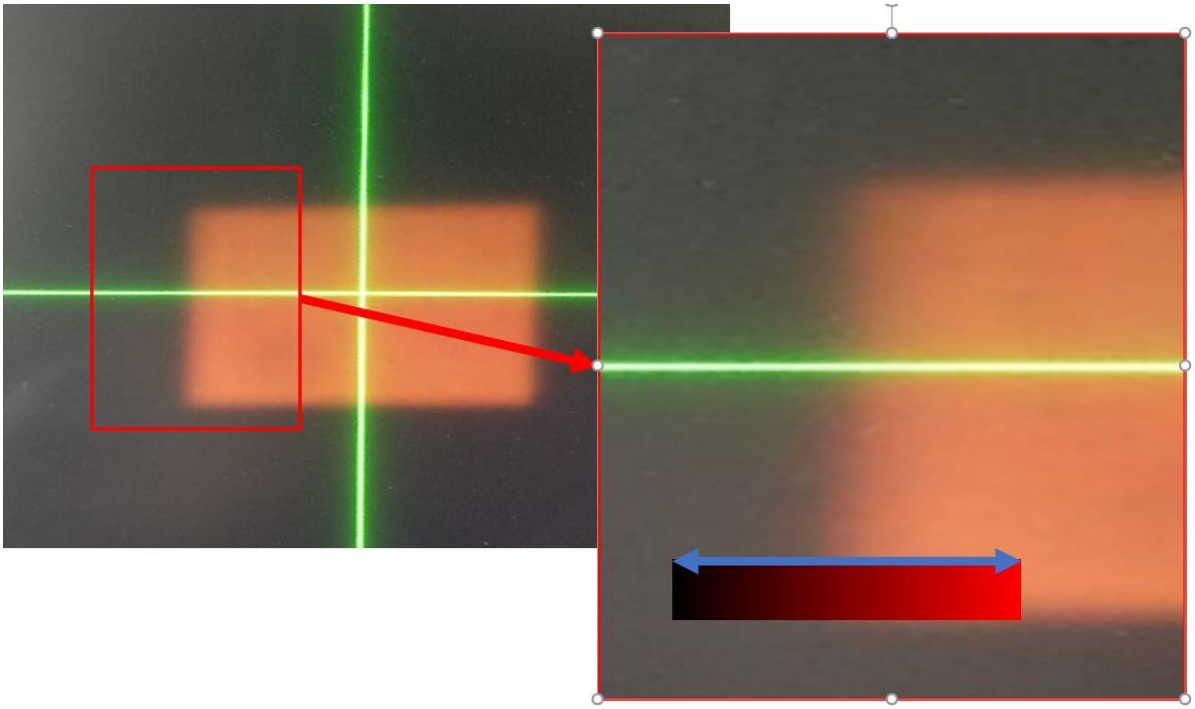


Abbildung 46: Ausschnitt rotes Objekt auf schwarzem Hintergrund, Kante mit Farbverlauf.

Der gezeigte Farbverlauf, resultiert in einen kleinen Unterschied zwischen den Pixeln, die kann mittels eines niedrigeren Threshold kompensiert werden. Dies führt jedoch zu einem weiteren Nebeneffekt, denn ist der Threshold gering, reichen bereits kleinste Änderung, um ein Start oder Ende eines Objektes zu erkennen. So reicht beispielweise ein Staubkrümel oder eine Veränderung des Winkels zwischen Szene und Kamera aus, damit die Erkennung gestört wird. Dieses Problem existiert auch bei der Verwendung der Laplace-gefilterten Bilddaten.

6.4 Erkennung von zweidimensionalen Objekten

Für die Erkennung von zweidimensionalen Objekten werden die Daten aus der Erkennung der Linienobjekten herangezogen und daraus auf ein Objekt geschlossen. Die Implementierung findet man im Modul „OBJ_DETECTION“ (Kapitel 4.4).

Die Erkennung in diesem Modul basiert auf dem Zählen der gefunden Linienobjekte und dessen Position. Sollten diese zusammenhängend übereinander liegen, so werden diese als Objekt interpretiert. Aus der Anzahl der gezählten Linien und den Beginn der ersten Linie kann auf den Mittelpunkt geschlossen werden.

Derzeit wird bei der Erkennung des Objektes immer nur von einem Objekt innerhalb des Bildes ausgegangen.

6.4.1 Problematiken

Das Modul „OBJ_DETECTION“ vertraut den Daten die es vom Modul „LINE_DETECTION“ oder Modul „LINE_DETECTION_CONV“ erhält und funktioniert auch nur wie geplant, wenn diese Daten korrekt sind. Wenn aber Probleme wie in Kapitel 0 auftreten, gibt es keine Plausibilitäts-Prüfung und somit kann es zu falsch voraus gesagten Positionen des Objekts kommen.

Des Weiteren wird nicht auf das Ende des Objektes bei der Zählung der Zeilen gewartet, sondern nach erreichen der Mindesthöhe von 30 Zeilen wird die aktuelle Position ermittelt und an den Ausgang gegeben.

6.5 Ausgabe der erkannten Position

In der Studienarbeit von (Herbst, 2020) wurden 7-Segmentanzeigen für jegliche Ausgaben an Informationen gegeben. So beispielsweise auch die Ausgabe der Position von isolierten Punkten und Linienobjekte in einem dimensionalen Bild aus der 7-Segmentanzeige als hexadezimal Zahl ausgegeben. Dies funktioniert ohne Einwände, ist jedoch mühselig in der Handhabung und verlangsamt das Prüfen der Daten auf Richtigkeit. Aufgrund dessen wurde sich in dieser Ausarbeitung dafür entschieden, die Positionsinformation zusätzlich auf der VGA Schnittstelle auszugeben. Die Visualisierung erfolgt über ein grünes Fadenkreuz, zu sehen in Abbildung 31.

Die Implementierung erfolgt in dem Modul „DEBUG_OBJ_DETECTION“ und setzt die oben genannten Funktionen um.

Eine ausführliche Erklärung kann in Kapitel 4.5 gefunden werden.

7 Zusammenfassung und Aussicht

Ziel dieser Studienarbeit war es, die bestehenden Projektstände zu reviewen und mithilfe der vorhandenen Funktionen eine zweidimensionale Objekterkennung zu implementieren. Dabei wurden Erkenntnisse aus einer vorherigen Studienarbeit verwendet und auf diesen weiter aufgebaut.

Zusätzlich soll diese Ausarbeitung dem Leser einen guten Überblick darüber bieten, was bereits existierte und wie die Funktionsweise der verschiedenen Teile sind. Dafür wurde Bestehendes um einige, für das Verständnis wichtige, Details und Grafiken ergänzt.

Um das Ziel zu erreichen, wurden unterschiedliche Projektstränge reviewed und zusammengeführt. So wurde als Basis für das Projekt ein vorheriger Stand aufgebaut, der bereits farbige Kamerabilder einlesen, speichern und auf der VGA Schnittstelle ausgeben konnte. Der Projektstrang, der aus der Studienarbeit von Herrn Herbst entstand, inspirierte vor allem bei der Erkennung von Linienobjekten. Damals noch im Graustufenbereich und nach dieser Ausarbeitung im Farbbereich.

Um die eindimensionale Linienobjekterkennung zu implementieren, wurde ein Funktionsblock entwickelt, der Gebrauch von den bereits ausgelesenen Pixeldaten, aus dem SDRAM, macht. Dieser Block erhält die Pixeldaten zeilenweise und wertet diese Anhand der Intensität aus, damit die Kanten eines roten Objektes gefunden werden. Dafür wurde ein Vergleichswert erstellt, dieser wird Threshold genannt. Der Threshold kann über einen ROM auf dem FPGA konfiguriert werden. Mit diesen Methoden lässt sich ein Linienobjekt definieren. Die Erkenntnis, anhand der Intensität eine Linienobjekterkennung zu implementieren beruht auf der Arbeit von (Herbst, 2020).

Zusätzlich wurde ein Block entwickelt, der auf den Erkenntnissen von (Perske, 2019) und dessen bereits implementierten Laplace-Filter basiert. Dieser lässt sich identisch zu dem vorher beschriebenen Funktionsblock für die eindimensionale Linienobjekterkennung verwenden. Ein Vergleich der beiden Methoden findet innerhalb dieser Studienarbeit nicht statt.

Für die Erkennung von zweidimensionalen Objekten wurde ein Funktionsblock erschaffen, der auf der Information der Linienobjekterkennung arbeitet. Durch Zählen der gefunden Linienobjekte, kann auf die Position des zweidimensionalen Objektes geschlossen werden.

Die aktuellen Positionserkennung bietet eine gute Grundlage für eine Fortführung. In der horizontalen Ebene ist die Erkennung schon sehr genau bei normalen Lichtverhältnissen. Die Erkennung in der vertikalen Ebene, benötigt einen gut eingestellten Fokus der Kamera und gute Beleuchtung. Die Qualität kann daher als ausbaufähig bezeichnet werden, wobei bei bestimmten Umgebungsparametern die Erkennung bereits robust funktioniert.

Um zukünftige Entwicklungen innerhalb der Positionserkennung zu erleichtern und den Debugging Prozess zu beschleunigen, wurde ein Modul implementiert, das eine zweidimensionale Position mithilfe eins grünen Fadenkreuzes auf der VGA Schnittstelle ausgeben kann. Dafür wurde, die bereits vorhandene Farbausgabe auf der VGA Schnittstelle manipuliert.

Zu der Implementierung der Module gehörten jeweils eine Testbench um das Debugging zu erleichtern und der eigentliche Test. Der bei Ausführung der beschriebenen Hardware das Ergebnis der Testbench bestätigen soll.

In einer Fortführung dieser Arbeit sollte die gewählte Art der Erkennung, wissenschaftlich geprüft werden. Dabei sollten die angesprochenen Problematiken und damit kommende Einschränkungen, der aktuellen Implementierungen in Betracht gezogen werden und mit Alternativen verglichen werden. Die implementierten Erkennungen funktionieren in ihren Grundzügen, jedoch existierten keine wissenschaftlichen Beweise ob diese von Relevanz sind.

Zusätzlich sollten Testszenarien definiert werden, bei den eine gewählte Erkennung ein definiertes Objekt in seiner Position bestimmen muss. Damit die Qualität der Objekterkennung bestimmt werden kann und unter anderem Probleme vorzeitig erkannt werden. Dabei könnten folgende Parameter in Betracht gezogen werden: Helligkeit (Auswahl Lichtquelle, Lichtfarbe), Abstand Kamera und Objekt, Winkel von Kamera zu Objekt, Fokus der Kamera, Objektfarbe und Hintergrundfarbe, Geschwindigkeit des Objekt und Größe des Objekt.

Alle Ergebnisse aus diesem Projekt, sowie vorherige Projektstände und alle dazu relevanten Informationen wurden in einem Repository geordnet und zusammengefasst. Damit sind alle nötigen Grundsteine gelegt für die Fortsetzung des Projektes.

8 Literaturverzeichnis

CameraLink Spec, C. (Oktober 2000). *imagelabs CameraLink Spec*. Von https://www.imagelabs.com/wp-content/uploads/2010/10/CameraLink5.pdf abgerufen

DigiKey. (17. 03 2021). *forum.digikey vga-controller-vhdl*. Von forum.digikey vga-controller-vhdl: https://forum.digikey.com/t/vga-controller-vhdl/12794 abgerufen

DIN19226. (kein Datum). *DIN 19226.* DIN. Von https://www.ingenieurkurse.de/regelungstechnik/einfuehrung-in-die-regelungstechnik/regelung/definition-der-regelung.html abgerufen

floctus.de. (31. 12 2017). Von floctus.de: https://www.flocutus.de/besser-fotografieren-belichtungbetter-photography-exposure/ abgerufen

GIT SCM. (kein Datum). Von https://git-scm.com/ abgerufen

Herbst, L. (2020). *Bilddatenvorverarbeitung in einem FPGA.* Hochschule Koblenz.

Mikael262, M. (kein Datum). *geocities*. Von http://www.geocities.ws/mikael262/sdram abgerufen

OMRON, S. (2017). *Color Camera Link Camera STC-CMC33PCL (VGA, Color), Product Specifications.*

Oster, C. (2021). *GitHub Chriss95De*. Von https://github.com/Chriss95De/Studienarbeit.git abgerufen

Perske. (2019). Projektarbeit FPGA RGB, Projektstand Quartus 2019. Koblenz.

Rafael C. Gonzalez, R. E. (1987). *Digital Image Processing, Third Edition, 978-0131687288.* Prentice Hall.

Technologies, T. (2013). *DE2-115 User Manuel.*

Terasic Technologies. (2003-2010). *CLR-HSMC User Manual.*

Thormählen, T. (19. Juni 2020). Multimediale Signalverarbeitung Bildverarbeitung: Filter. Marburg.