

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 Access Memory
ROM	Read Only Memory
FPS	Frames per Second/ Bilder die 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

Kurzzusammenfassung.....	2
Abkürzungsverzeichnis .....	3
Abbildungsverzeichnis.....	6
1    Einleitung.....	7
2    Digitale Bildverarbeitung.....	8
2.1    Funktionsweise Darstellung von Pixeln .....	8
2.2    Farbunterschiede erkennen .....	9
3    Übersicht Zustand vorher.....	10
3.1    Modul: cameralink_tapping_base.....	11
3.1.1    Kameramodi: Taps, horizontales und vertikales Timing .....	11
3.1.2    Horizontales Timing 2 Tap.....	12
3.1.3    Vertikales Timing .....	12
3.2    Modul: camera_data_mux_gen .....	13
3.2.1    Aufbau Intern: camera_data_mux_gen (Vereinfachte Version).....	13
3.3    Modul: SDRAM_Write_Buffer_gen .....	14
3.3.1    Aufbau Intern: SDRAM_Write_Buffer_gen (Vereinfachte Version).....	14
3.4    Modul: Memory Access Control.....	15
3.4.1    Funktionsweise Interface .....	15
3.5    Modul: SDRAM Controller .....	16
3.6    Modul: SDRAM_Read_Buffer_gen .....	17
3.6.1    Interne Buffer Struktur .....	17
3.7    Modul: SDRAM_Pixelbuffer.....	18
3.7.1    Aufbau Intern: SDRAM_Pixelbuffer(Vereinfachte Version) .....	19
3.8    Modul: Debay .....	20
3.10    Modul: Convolution.....	22
3.11    VGA Ausgabe .....	23
4    Übersicht Zustand (nach Studienarbeit Oster).....	24
4.1    Funktionsweise/Anknüpfung an Vorheriges .....	25
4.2    Übersicht aktueller Zustand .....	26
4.3    Modul: LINE_DETECTION_CONV .....	27
4.3.1    Interner Aufbau Modul „LINE_DETECTION_CONV“ .....	28
4.3.2    Testbench „LINE_DETECTION_CONV_TB“.....	29
4.4    Modul: OBJ_DETECTION.....	30
4.4.1    Interner Aufbau Modul „OBJECT_DETECTION“ .....	31
4.5    Modul: DEBUG_OBJ_DETECTION .....	32

4.5.1	Interner Aufbau Modul „DEBUG_OBJECT_DETECTION“ .....	33
5	Beeinflussende Parameter .....	34
5.1	Beleuchtung.....	34
5.2	Einstellung Treshold .....	35
6	Ergebnis und Diskussion .....	36
6.1	Zusammenführen verschiedener Projektstränge.....	36
6.2	Kontentmanagement .....	36
6.3	Erkennung von Linienobjekten innerhalb einer Bildzeile.....	38
6.3.1	Problematiken .....	38
6.4	Erkennung von zweidimensionalen Objekten.....	39
6.4.1	Problematiken .....	39
6.5	Ausgabe der erkannten Position .....	39
7	Zusammenfassung und Aussicht .....	40
8	Literaturverzeichnis.....	41

## Abbildungsverzeichnis

ABBILDUNG 1: DIGITALES BILD BEISPIEL .....	8
ABBILDUNG 2: PIXELREIHE $f(x)$ UND 1. ABLEITUNG $f'(x)$ , GRAUSTUFEN IN 8 BIT DARSTELLUNG .....	9
ABBILDUNG 3: ÜBERSICHT DER MODULE INNERHALB DES BISHERIGEN QUARTUS PROJEKT .....	10
ABBILDUNG 4: ÜBERSICHT VERWENDETE HARDWARE, (QUELLE BILD FPGA: DE2-115 USER MANUAL) (QUELLE BILD KAMERA: SENTECH STC-CMC33PCL PRODUCT SPEC) .....	10
ABBILDUNG 5: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	11
ABBILDUNG 6: ÜBERSICHT TIMING HORIZONTAL, MODI 2TAP, 642 PIXEL HORIZONTAL (QUELLE: SENTECH PRODUCT SPEC) .....	12
ABBILDUNG 7: ÜBERSICHT TIMING VERTIKAL, 484 ZEILEN VERTIKAL (QUELLE: SENTECH PRODUCT SPEC) .....	12
ABBILDUNG 8: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	13
ABBILDUNG 9: VEREINFACHTE FUNKTIONSWEISE MODUL „CAMERA_DATA_MUX_GEN“ .....	13
ABBILDUNG 10: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	14
ABBILDUNG 11: VEREINFACHTE FUNKTIONSWEISE MODUL „SDRAM_WRITE_BUFFER_GEN“ .....	14
ABBILDUNG 12: BLOCKSCHALTBILD „MEMORYACCESSCONTROLLER“ MIT PARAMETERN, (QUELLE: QUARTUS PRIME LITE) .....	15
ABBILDUNG 13: FUNKTIONSWEISE INTERFACE MEMORYACCESSCONTROLLER .....	15
ABBILDUNG 14: BLOCKSCHALTBILD + BESCHALTUNG, (QUELLE: QUARTUS PRIME LITE) .....	16
ABBILDUNG 15: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	17
ABBILDUNG 16: INTERNE BUFFER STRUKTUR DES MODULS „SDRAM_READ_BUFFER_GEN“ .....	17
ABBILDUNG 17: BLOCKSCHALTBILD, ROT MARKIERT DIE WICHTIGSTEN IOS, (QUELLE: QUARTUS PRIME LITE) .....	18
ABBILDUNG 18: SCHAUBILD FUNKTIONSWEISE PIXELBUFFER .....	18
ABBILDUNG 19 SCHAUBILD PIXELBUFFER ZEILENENDE .....	19
ABBILDUNG 20 SCHAUBILD INTERNER AUFBAU SDRAM_PIXELBUFFER .....	19
ABBILDUNG 21 BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	20
ABBILDUNG 22 AUFTeilUNG 5x5 BYTE BLOCK IN KLEINERE 3x3 BLÖCKE .....	20
ABBILDUNG 23: ZUSAMMENSETZUNG 3x3, BEISPIEL AM ROTKANAL, $R_{l1}$ = OUTPUT DER ERSTEN ZEILE DES ROTKANALS .....	21
ABBILDUNG 24: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	22
ABBILDUNG 25: GEWICHTUNG DES 3x3 ROT KANAL FÜR BERECHNUNG ROT KANAL .....	22
ABBILDUNG 26: BERECHNUNG FARBWERT R .....	22
ABBILDUNG 27: SCHALTBILD VGA AUSGABE, (QUELLE: QUARTUS PRIME LITE) .....	23
ABBILDUNG 28: TESTAUFBAU ERKENNUNG ROTES RECHTECK AUF BILDSCHIRM .....	24
ABBILDUNG 29: ZIELKREUZ AUF ERKANNTER POSITION .....	24
ABBILDUNG 30: BEISPIEL ERKENNUNG ROTES RECHTECK .....	25
ABBILDUNG 31: ÄNDERUNG DER INTENSITÄT DES ROTKANALS, 1.ABLEITUNG NACH GONSALEZ .....	25
ABBILDUNG 32: ZÄHLEN DER ZEILEN IN DEM EIN OBJEKT ERKANNT WURDE .....	26
ABBILDUNG 33: ÜBERSICHT ZUSTAND MODULE NACH STUDIENARBEIT OSTER, 2021 .....	26
ABBILDUNG 34: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	27
ABBILDUNG 35: VEREINFACHTE FUNKTIONSWEISE MODUL „LINE_DETECTION_CONV“ .....	28
ABBILDUNG 36: AUSZUG MODEL SIM, SIMULATION „LINE_DETECTION_CONV_TB“ .....	29
ABBILDUNG 37: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	30
ABBILDUNG 38: VEREINFACHTE FUNKTIONSWEISE MODUL „OBJECT_DETECTION“ .....	31
ABBILDUNG 39: BLOCKSCHALTBILD, (QUELLE: QUARTUS PRIME LITE) .....	32
ABBILDUNG 40: VEREINFACHTE FUNKTIONSWEISE MODUL „DEBUG_OBJECT_DETECTION“ .....	33
ABBILDUNG 41: BEISPIEL ZU BELEUCHTUNG, (QUELLE: WWW.FLOCUTUS.DE) .....	34
ABBILDUNG 42: READ ONLY MEMORY QUARTUS PROJEKT, (QUELLE: QUARTUS PRIME LITE) .....	35
ABBILDUNG 43: AUSSCHNITT ROTES OBJEKT AUF SCHWARZEM HINTERGRUND, KANTE MIT FARBVERLAUF. ....	38

# 1 Einleitung

Das Regeln/die Regelung ist ein Vorgang bei dem eine Größe, die zu regelnde Größe (Regelgröße), fortlaufend erfasst, mit einer anderen Größe, der Führungsgröße verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst (DIN19226)

Diese Arbeit beschäftigt sich mit der Realisierung eines Messglied, bei denen geringe Totzeiten im Fokus stehen. Totzeiten können allgemein dafür sorgen, dass ein Regelkreis instabil oder träge wird und somit einer definierten Anforderung nicht mehr entsprechen könnte.

Die Messgröße soll die horizontale Position eines Objektes darstellen, die anhand seines Farbunterschied zum Rest einer Szene (Hintergrund) erkannt wird. Für die Erkennung eines solchen Objektes bietet sich eine Sensorlösung an, die Teile des elektromagnetischen Spektrums aufnimmt und diese in ein elektrisches Signal umwandelt.

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

Um die Totzeiten im Messglied gering zu halten, wurde bei der Auswahl der Kamera auf eine hohe Anzahl an Bildern die Sekunde 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 Camera Link von der Kamera auf das FPGA-Board zu übertragen und diese dort zwischenspeichern. Die zwischengespeicherten Daten konnten über den VGA-Ausgang auf einem angeschlossenen Monitor ausgegeben werden. [Lukas Herbst, Bilddatenvorverarbeitung in einem FPGA] Zudem war es möglich 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 beschreibenden 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.

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 des bisher Existierenden und Neu dazu kommenden Modulen. 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 auf Erkenntnisse der vorangegangenen Studienarbeit „Bildenvorverarbeitung 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 von 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. Jeder Pixel kann mit der Funktion  $f(x,y)$  einzeln ausgelesen werden.

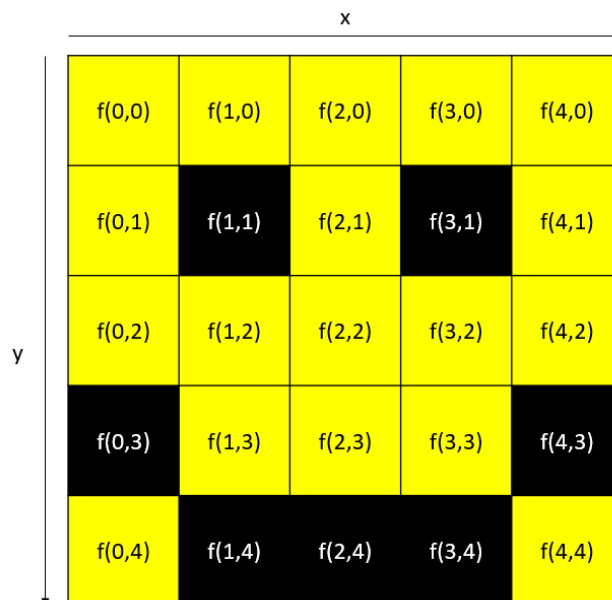


Abbildung 1: Digitales Bild Beispiel

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 Bilderverarbeitung 1 ein RGB Farbwert. Ein RGB Farbwert verfügt über drei Informationen, diese definieren wie stark ein rot, grün oder blau Anteil ausgeprägt ist.

Beispiel zu Abbildung 1: {rot = 255, grün = 255, blau = 0} =  $f(0,0)$

Das Beispiel ergibt einen starken Gelbton. Die Zahl 255 stellt hier den maximalen Wert 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

In [2.3 Mathematische Operationen zur Bilddatenverarbeitung, Bilddatenvorverarbeitung in einem FPGA, Lukas Herbst] beschrieben, können Unterschiede von Pixeln in eine Reihe wie folgt erfasst werden.

$$1. \text{Ableitung Definition von Gonzalles: } \frac{\partial f}{\partial x} = f(x+1) - f(x)$$

Das Ergebnis der ersten Ableitung beschreibt den Unterschied von einem zum nächsten Pixel.

Folgend eine Veranschaulichung zu einer Reihe die aus 5 Pixeln besteht. Jeder Pixel wird durch eine Graustufe repräsentiert.



	$x$				
					
$f(x):$	$f(0)$ = 100	$f(1)$ = 100	$f(2)$ = 200	$f(3)$ = 100	$f(4)$ = 100
	$x$				
					
$\frac{\partial f}{\partial x}:$	$f'(0)$ = 0	$f'(1)$ = 100	$f'(2)$ = -100	$f'(3)$ = 0	$f'(4)$ = ?

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 der Nachbar Pixel 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 und kann daher vernachlässigt werden.

### 3 Übersicht Zustand vorher

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

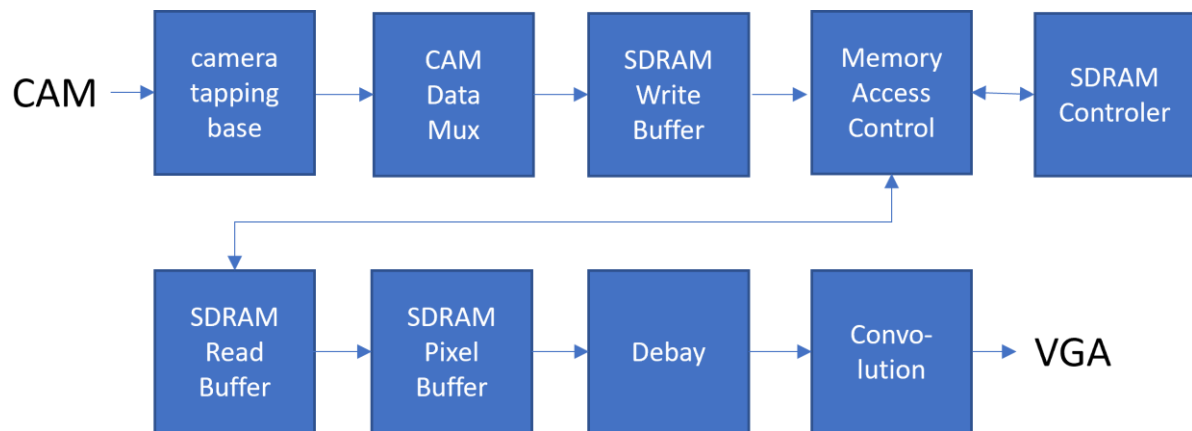


Abbildung 3: Übersicht der Module innerhalb des bisherigen Quartus Projekt

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

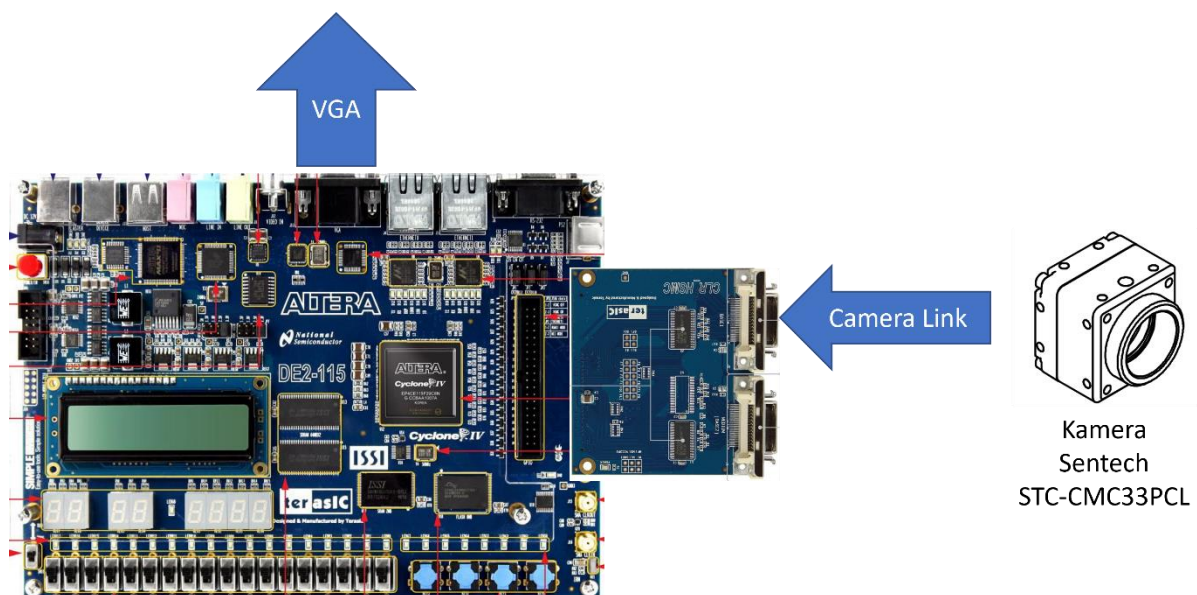


Abbildung 4: Übersicht verwendete Hardware,  
(Quelle Bild FPGA: DE2-115 User Manual)  
(Quelle Bild Kamera: SENTECH STC-CMC33PCL Product Spec)

### 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.

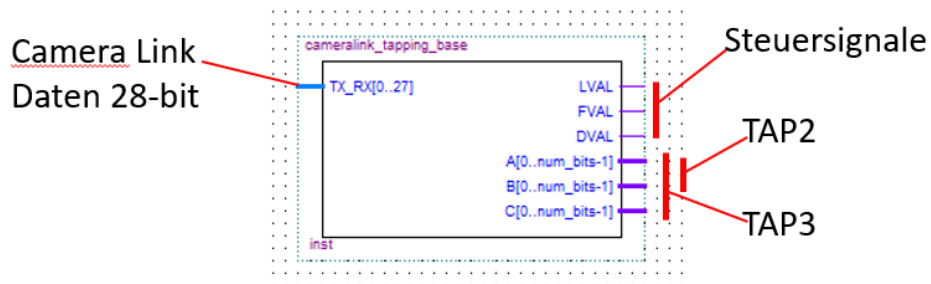


Abbildung 5: Blockschaltbild, (Quelle: Quartus Prime Lite)

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

1CLK = 11.9 nsec.

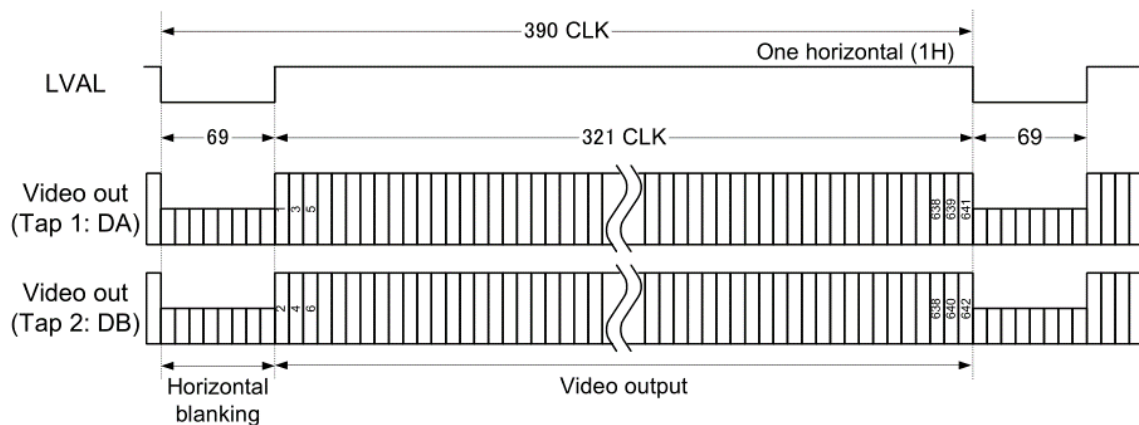


Abbildung 6: Übersicht Timing horizontal, Modi 2Tap, 642 Pixel horizontal (Quelle: SENTECH Product Spec)

„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.

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

### 3.1.3 Vertikales Timing

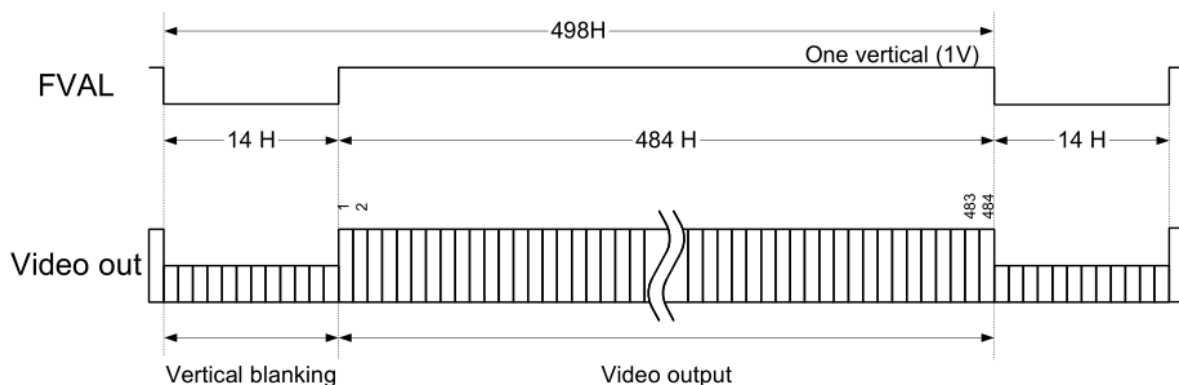


Abbildung 7: Übersicht Timing vertikal, 484 Zeilen vertikal (Quelle: SENTECH Product Spec)

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 „Seite 19 camera\_data\_mux\_gen, Bilddatenvorverarbeitung in einem FPGA, Lukas Herbst“ gefunden werden.

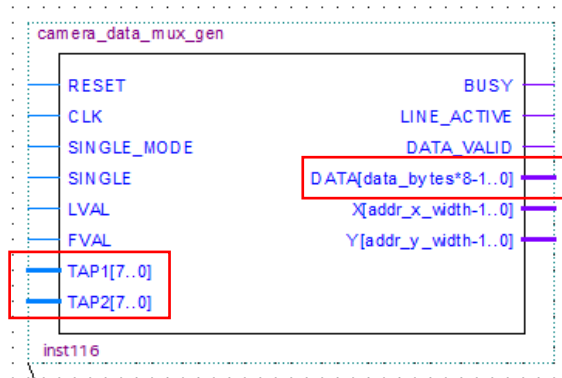


Abbildung 8: Blockschaltbild, (Quelle: Quartus Prime Lite)

#### 3.2.1 Aufbau Intern: camera\_data\_mux\_gen (Vereinfachte Version)

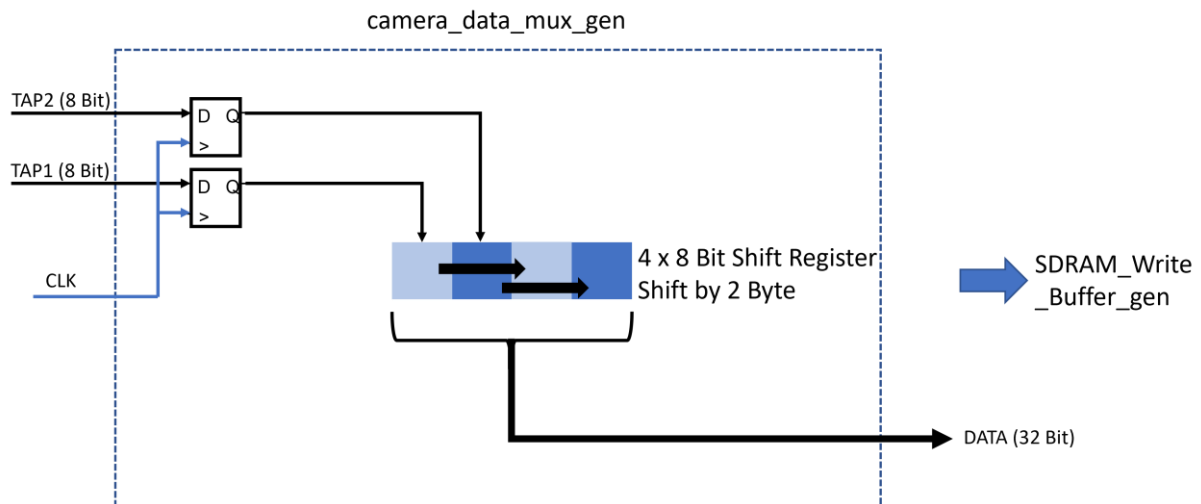


Abbildung 9: Vereinfachte Funktionsweise Modul „camera\_data\_mux\_gen“

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 „sdr\_clk“ realisiert.

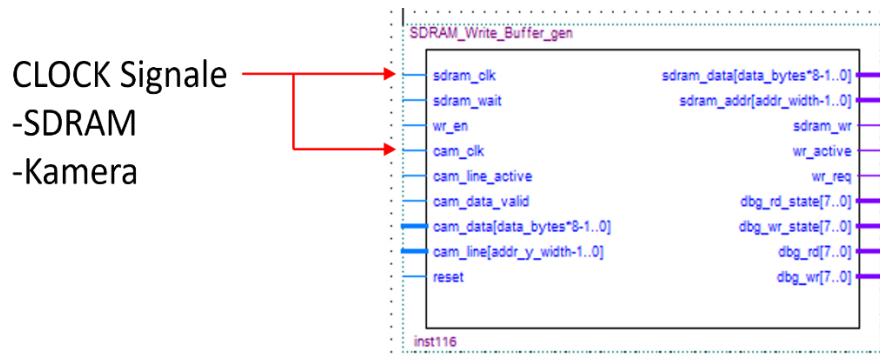


Abbildung 10: Blockschaltbild, (Quelle: Quartus Prime Lite)

Die Kameradaten werden in einem FIFO gepuffert, um dann bei freiem Schreibzugriff auf den SDRAM diese dort zu speichern. Der Ausgang „sdr\_data“ sind die Kameradaten, die in den SDRAM geschrieben werden und „sdr\_addr“ gibt die Position im SDRAM vor. In der aktuellen Form werden jeweils zwei Byte per Taktänderung aus dem FIFO in den SDRAM geschrieben.

### 3.3.1 Aufbau Intern: SDRAM\_Write\_Buffer\_gen (Vereinfachte Version)

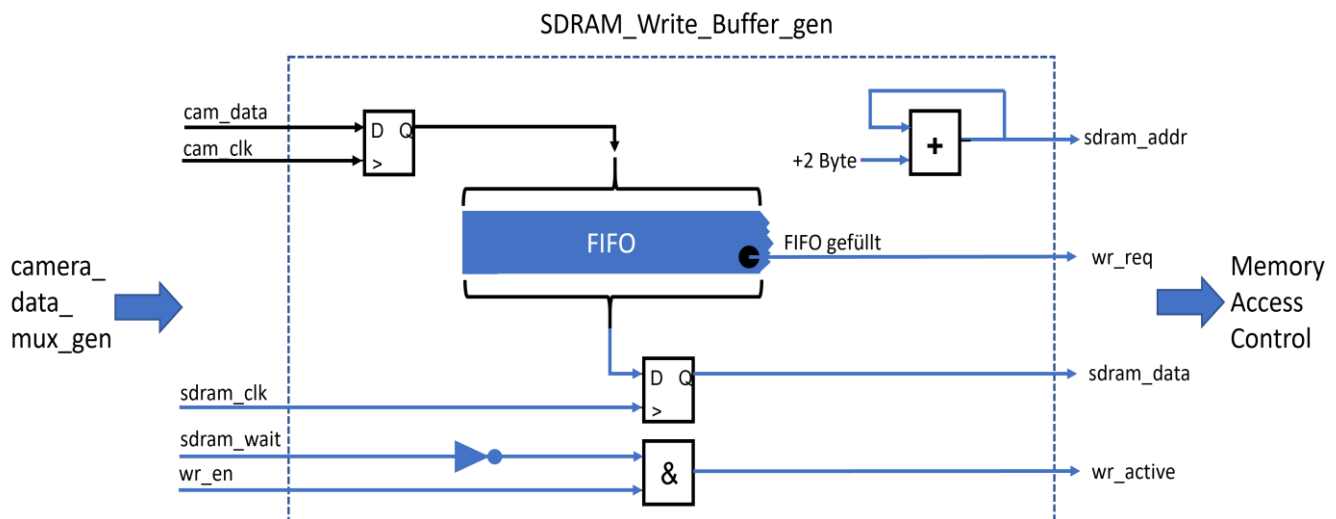


Abbildung 11: Vereinfachte Funktionsweise Modul „SDRAM Write Buffer gen“

### 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 die Beschriftung der Eingangsgröße die von einer Quelle gesteuert werden.

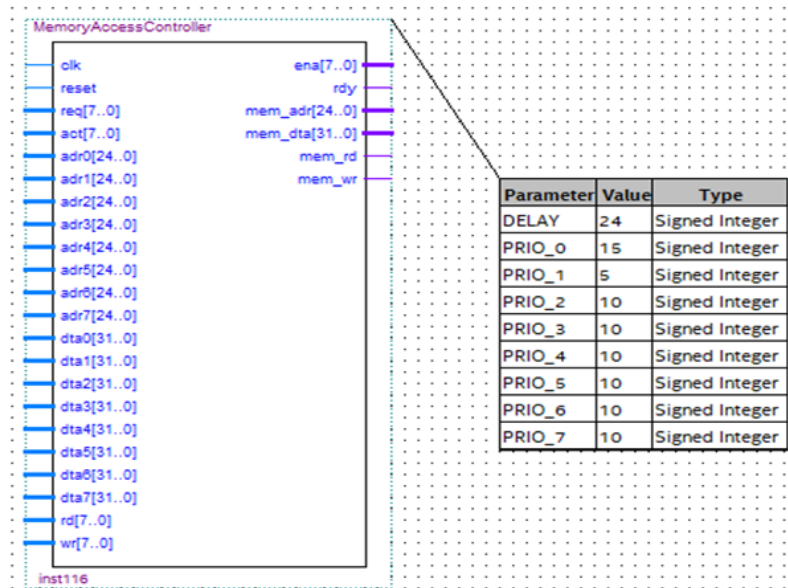


Abbildung 12: Blockschaltbild „MemoryAccessController“ mit Parametern, (Quelle: Quartus Prime Lite)

Das Interface selbst besteht aus den folgenden Signalen:

- req[7..0]: steht für „request“, Zugriffsanfrage auf den Speicher
- act[7..0]: Zusage des Zugriff 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

#### 3.4.1 Funktionsweise Interface

Eine Zugriffsanfragen wird mittels „req“ angefordert. Bleibt die Anforderung „DELAY“ Takte stehen, ohne dass eine Anforderung mit höherer Priorität kommt wird „ena“ gesetzt. Darauf muss der anfordernde Baustein „act“ setzen, solange ein Zugriff erfolgt.

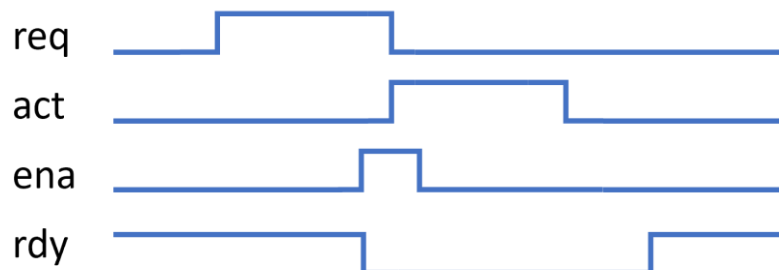


Abbildung 13: Funktionsweise Interface MemoryAccessController

### 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 ermöglicht 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 Rahmen dieser kleinen Zusammenfassung übersteigen. Eine detaillierte Erklärung zur Funktionsweise eines anderen SDRAM Controller erhalten Sie unter <http://www.geocities.ws/mikael262/sdram>.

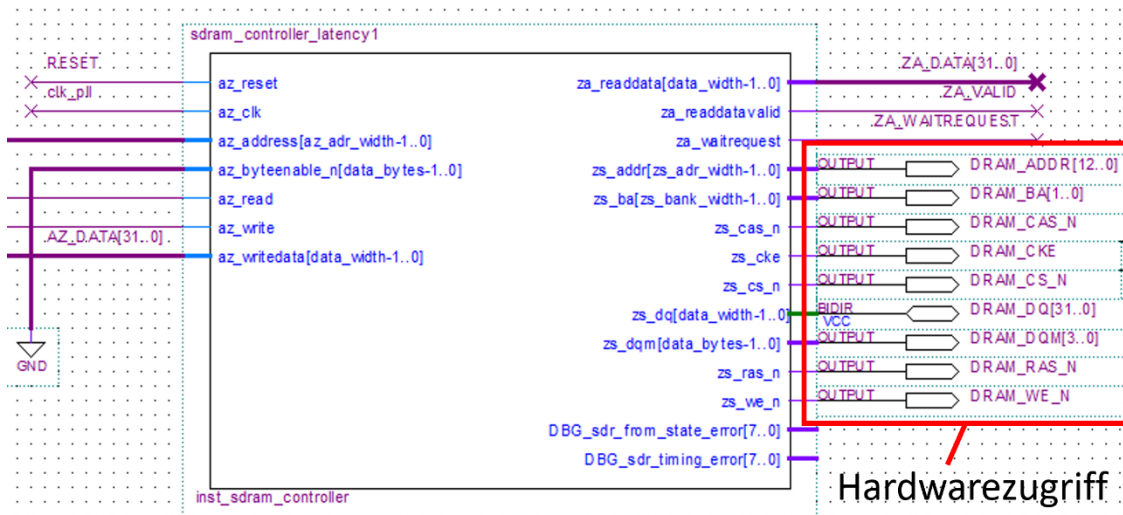


Abbildung 14: Blockschaltbild + Beschaltung, (Quelle: Quartus Prime Lite)

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 logischen High-Pegel
- `za_waitrequest`: Bei 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.

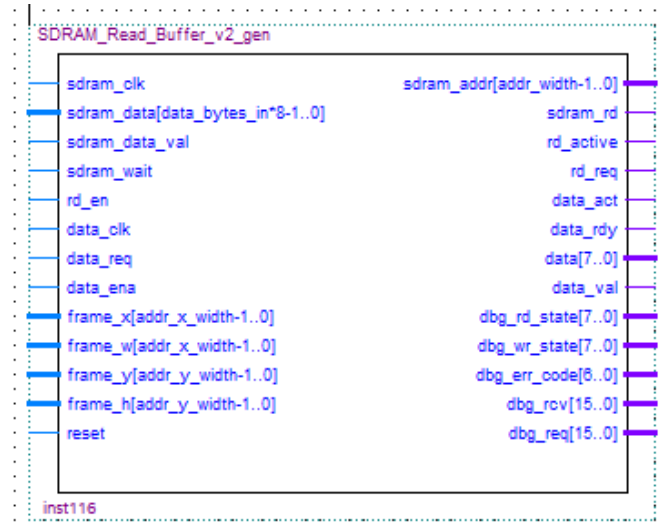


Abbildung 15: Blockschaltbild, (Quelle: Quartus Prime Lite)

#### 3.6.1 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.

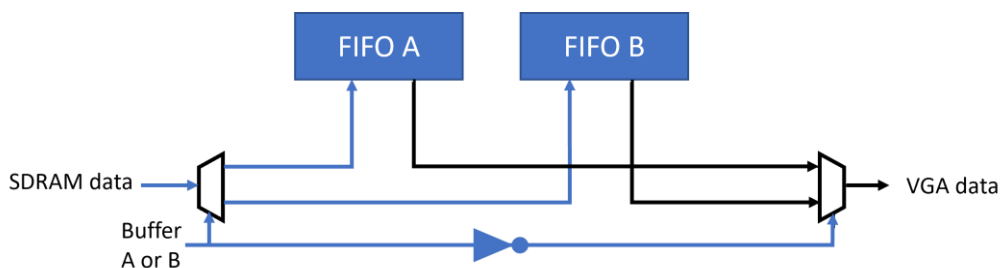


Abbildung 16: Interne Buffer Struktur des Moduls „SDRAM\_Read\_buffer\_gen“

### 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.

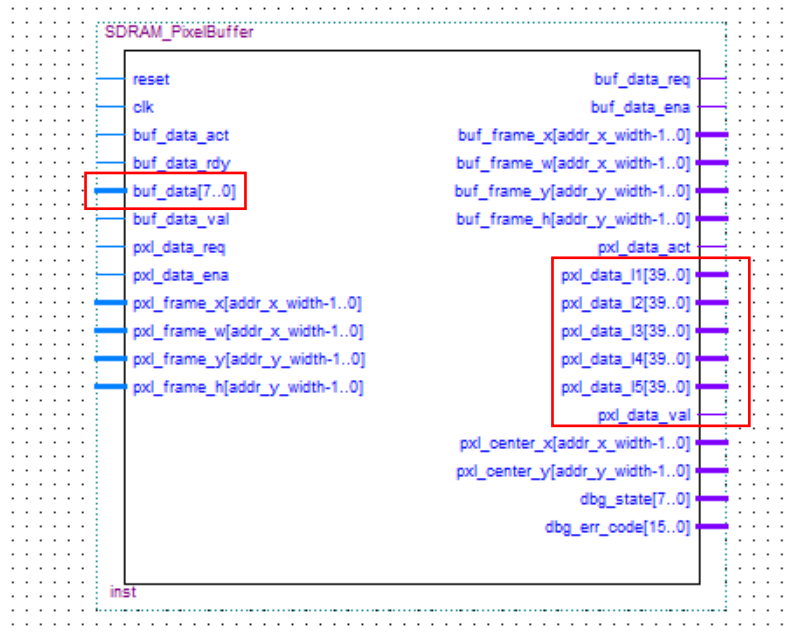


Abbildung 17: Blockschaltbild, rot markiert die wichtigsten I/Os, (Quelle: Quartus Prime Lite)

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.

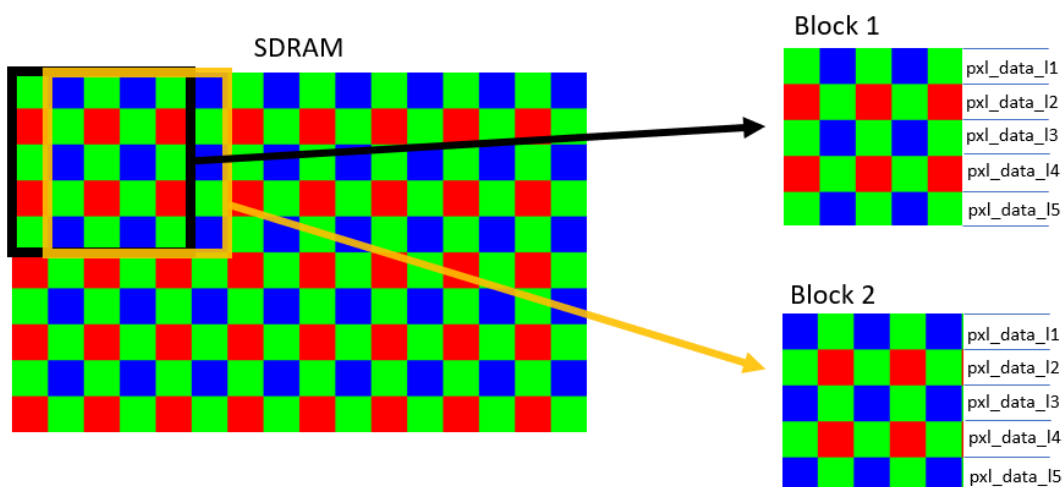


Abbildung 18: Schaubild Funktionsweise Pixelbuffer

Die fünf Ausgänge, für jeweils eine Zeile, sind auf den Eingang „pxl\_data1 ... pxl\_data5“ 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.

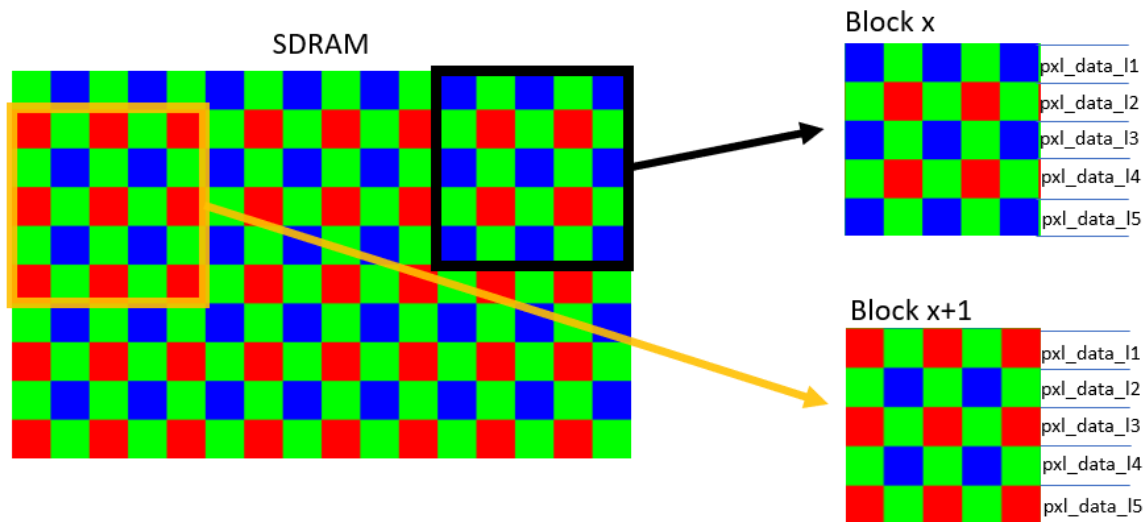


Abbildung 19 Schaubild Pixelbuffer Zeilenende

### 3.7.1 Aufbau Intern: SDRAM\_Pixelbuffer(Vereinfachte Version)

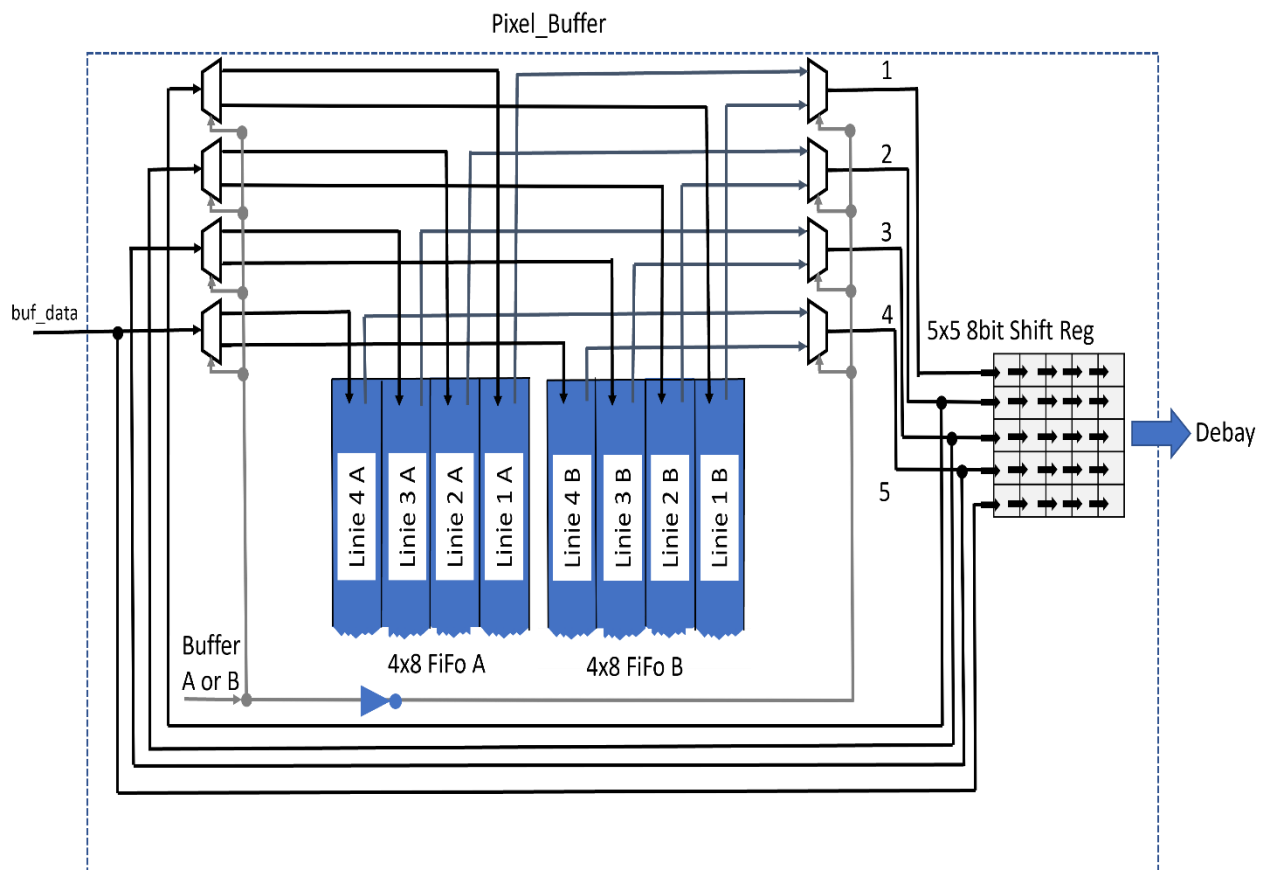


Abbildung 20 Schaubild interner Aufbau SDRAM\_Pixelbuffer

### 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.

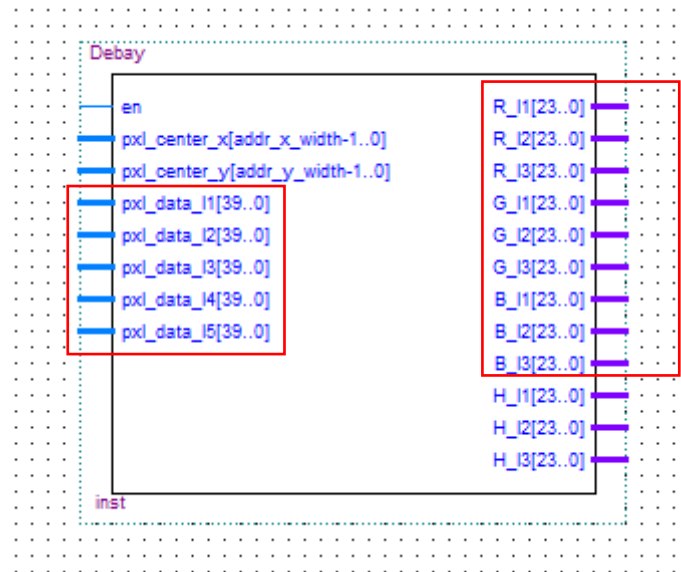


Abbildung 21 Blockschaltbild, (Quelle: Quartus Prime Lite)

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.

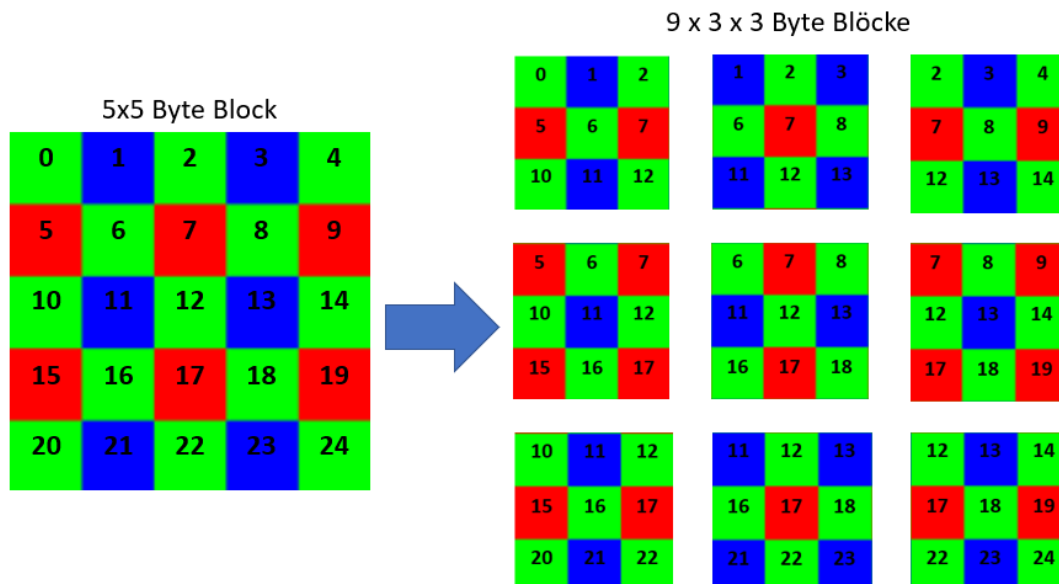


Abbildung 22 Aufteilung 5x5 Byte Block in kleinere 3x3 Blöcke

Durch die Aufteilung erhalten wir neun 3x3 Byte Blöcke. Aus jedem Block wird für alle drei Farbkanaäle 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.

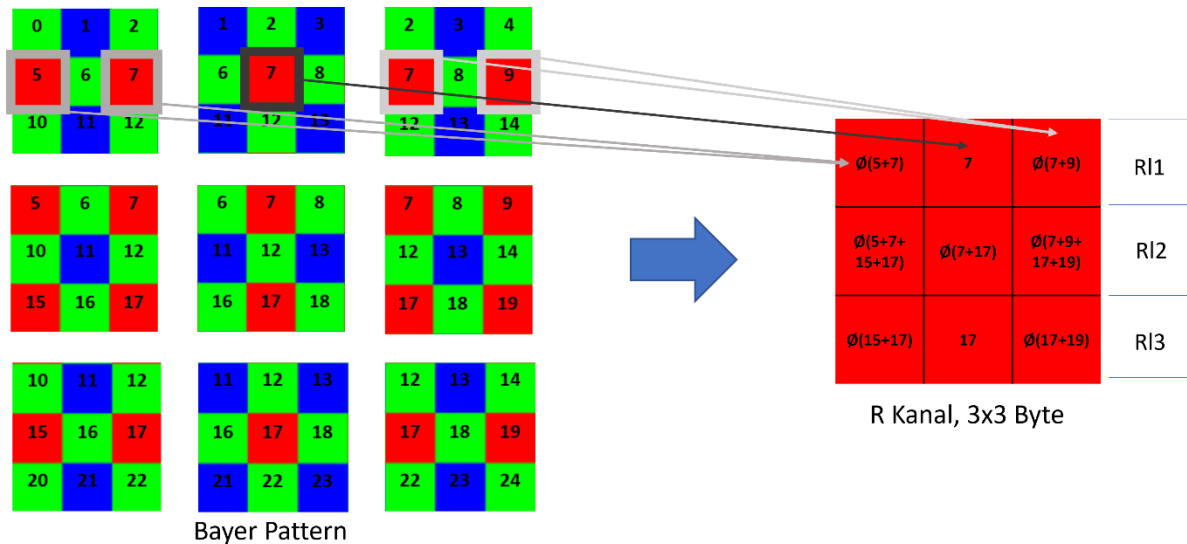


Abbildung 23: Zusammensetzung 3x3, Beispiel am Rotkanal, R1 = Output der ersten Zeile des Rotkanals

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: Convolution

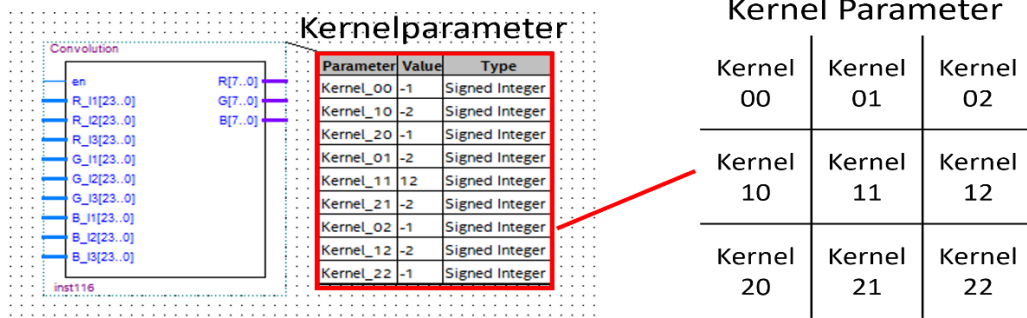


Abbildung 24: Blockschaltbild, (Quelle: Quartus Prime Lite)

„Convolution“ das bedeutet Faltung und ist der Hauptbestandteil der inneren Funktionsweise des Moduls in Abbildung 24. 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 für jeden neuen Block einen RGB Pixelwert zu berechnen. Die Blöcke stammen aus dem ursprünglichen Bild und werden jeweils um **eine** Zeile in Ihrer Auswahl verschoben. Damit schlussendlich ein Bild mit der gleichen Anzahl an Pixel in Höhe und Breite zur Verfügung steht.

Für die Gewichtung der Farbanteile aus dem Byte Block stehen die Kernel Parameter, siehe Abbildung 24, zur Verfügung.

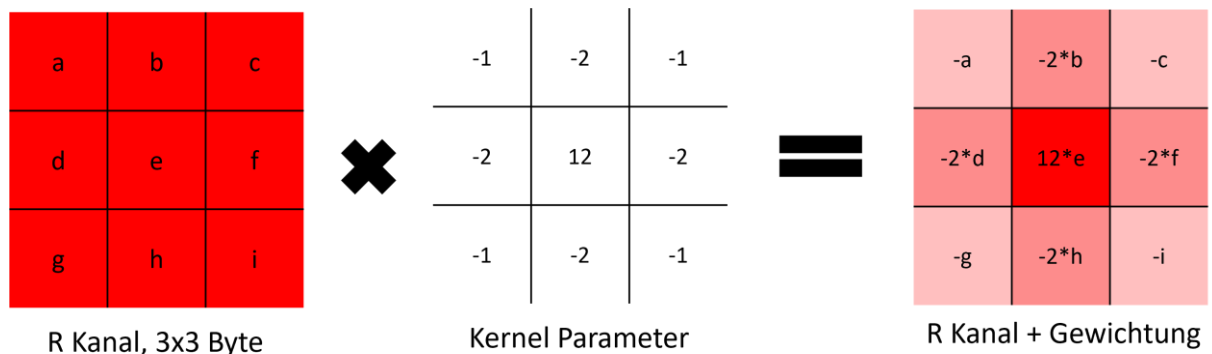


Abbildung 25: Gewichtung des 3x3 Rot Kanal für Berechnung Rot Kanal

Die Gewichtung führt dazu, dass in Abbildung 25 der Wert „e“ am stärksten betont wird, wobei die Nachbarwerte weniger gewichtet werden. **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt, wie aus dem gewichteten R Kanal Block ein einziger Farbwert wird.

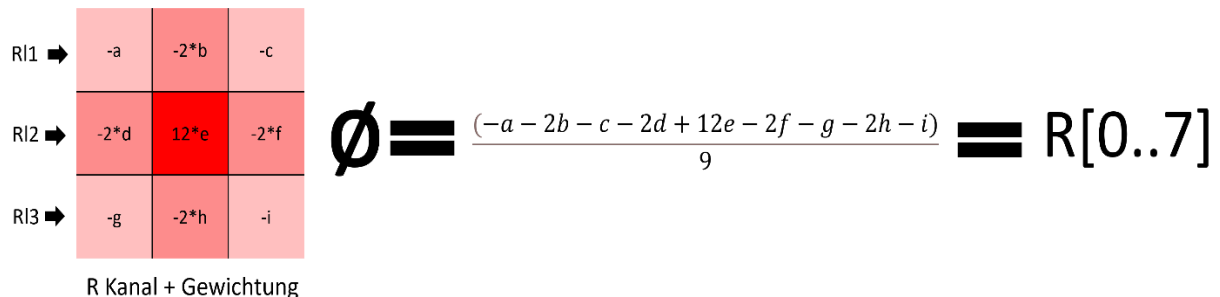


Abbildung 26: Berechnung Farbwert R

### 3.11 VGA Ausgabe

Für die Ausgabe der Bilddaten ist das Modul „vga\_controller“ zuständig. Eine gute Übersicht über die VGA Spezifikation erhält im Tech Forum der Firma Digi-Key zum Thema „VGA Controller (VHDL)“ (DigiKey, 2021) und wird an dieser Stelle nicht weiter erläutert.

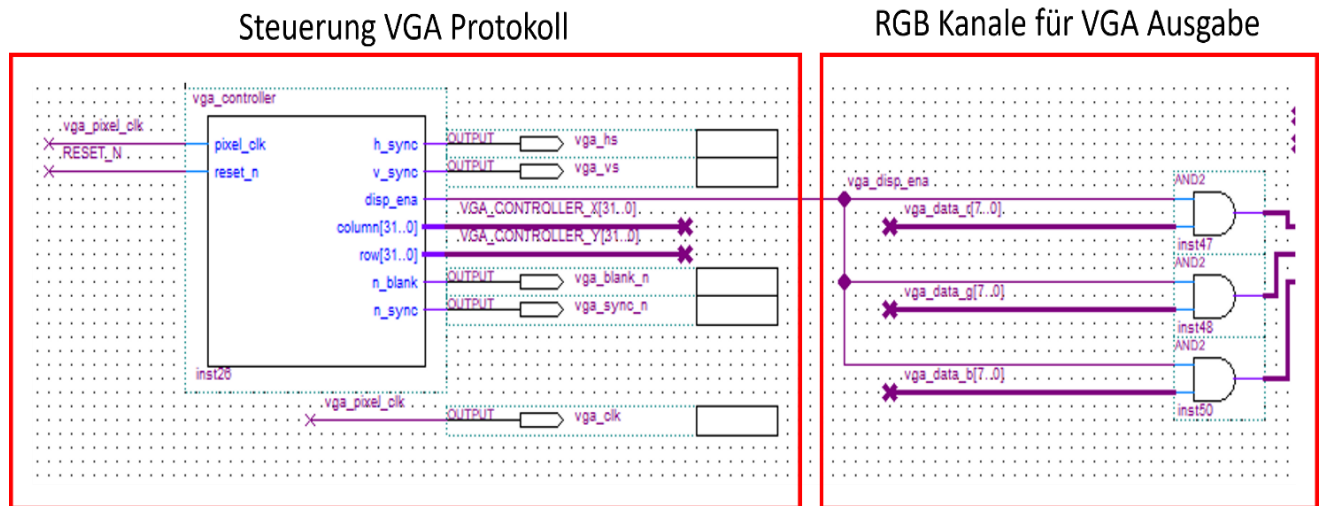


Abbildung 27: Schaltbild VGA Ausgabe, (Quelle: Quartus Prime Lite)

In Abbildung 27 zu sehen 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 „Convolution“ Block.

#### 4 Übersicht Zustand (nach Studienarbeit Oster)

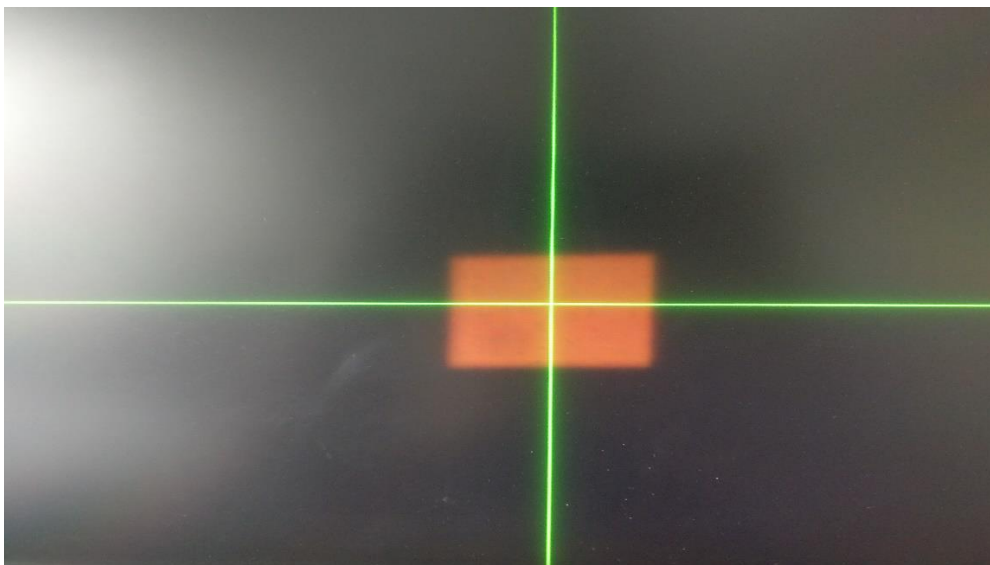
Während der Bearbeitung wurde die Erkennung eines farbigen Objektes in seiner X- und Y-Achse angestrebt. Dies wurde mit gewissen Einschränkungen erreicht, diese Einschränkungen werden im Kapitel „Beeinflussende Parameter“ erläutert. Als Beispiel für die Erkennung diente der Aufbau in Abbildung 28.



*Abbildung 28: 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, die verwendete Kamera, 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 zu sehen in Abbildung 18.



*Abbildung 29: Zielkreuz auf erkannter Position*



#### 4.1 Funktionsweise/Anknüpfung an Vorheriges

Aus der vorhergegangenen Studienarbeit von Herbst 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.

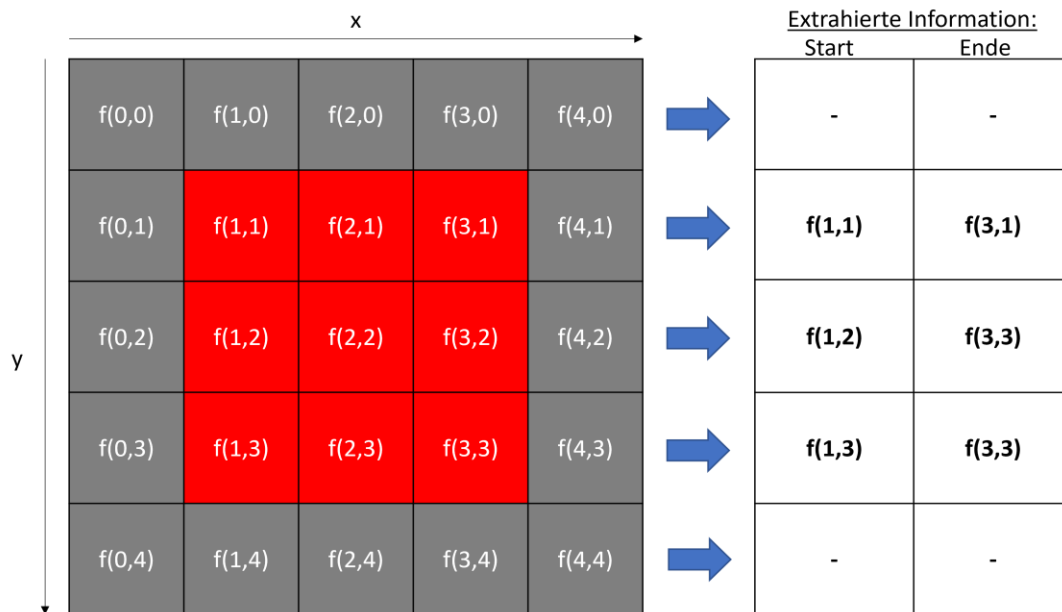


Abbildung 30: Beispiel Erkennung rotes Rechteck

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 30. Für die Erkennung stehen die RGB-Farbinformationen zur Verfügung, um starke Änderung zwischen Pixelnachbarn ausfindig zu machen. So können wir wie in Abbildung 30 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.

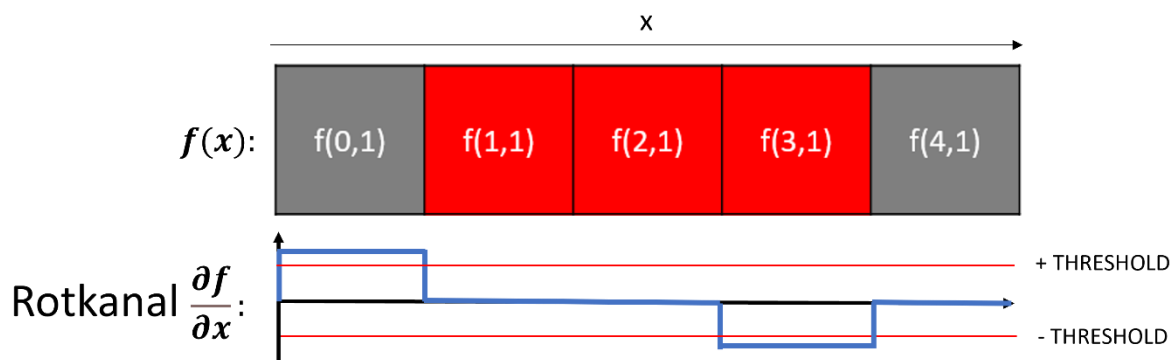


Abbildung 31: Änderung der Intensität des Rotkanals, 1. Ableitung nach Gonzalez

Abbildung 31 zeigt das Anfang und 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.

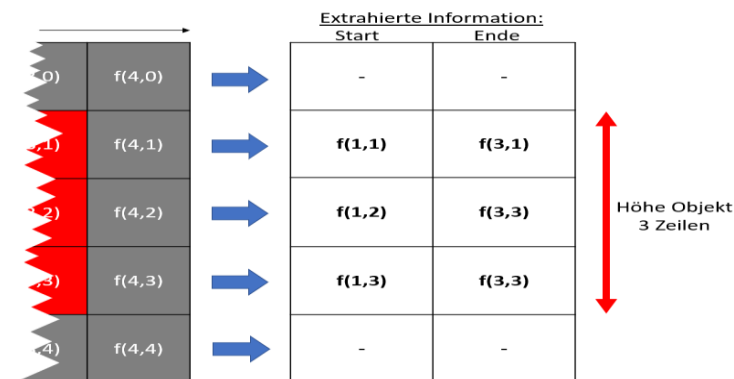


Abbildung 32: Zählen der Zeilen in dem ein Objekt erkannt wurde

Diese Funktion der Zeilenweise 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 29 auf der VGA Ausgabe angezeigt, dies ist im Modul „DEB\_OBJ\_DETECTION“ umgesetzt.

#### 4.2 Übersicht aktueller Zustand

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

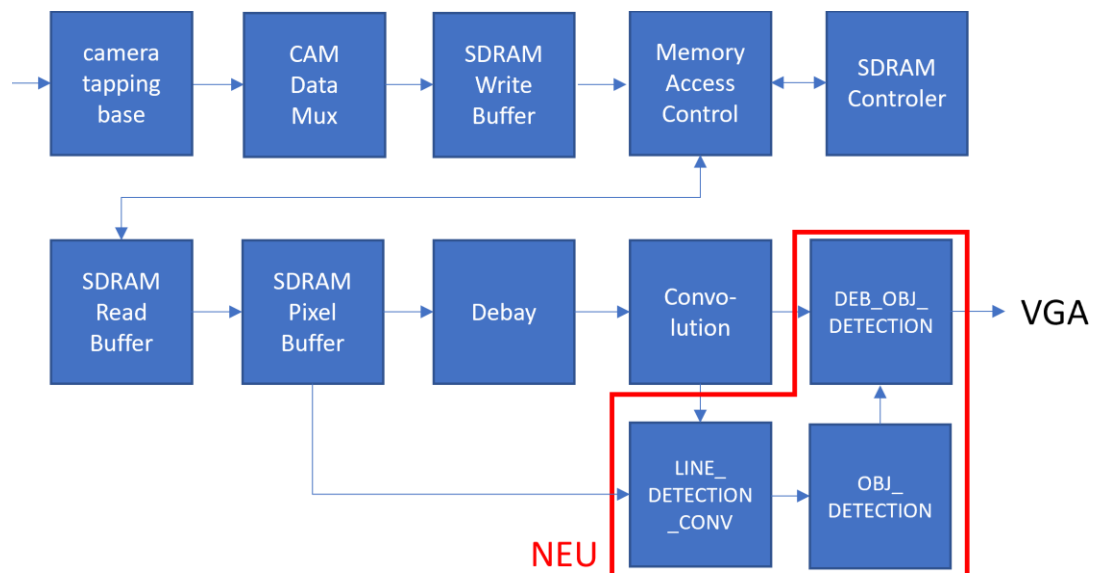


Abbildung 33: Übersicht Zustand Module nach Studienarbeit Oster,2021

Die neuen Module aus Abbildung 33, rot markiert, arbeiten auf den Ausgangssignalen des Moduls „Convolution“. In den folgenden Kapiteln werden die Module im Detail beschrieben.

### 4.3 Modul: LINE\_DETECTION\_CONV

Wie in Abbildung 31 dargestellt, übernimmt das Modul „LINE\_DETECTION\_CONV“ die Aufgabe, innerhalb einer Zeile Übergänge zwischen wenig und viel Rotanteil zu finden. Daran werden der Start und Ende eines Objektes innerhalb der Zeile fest gemacht.

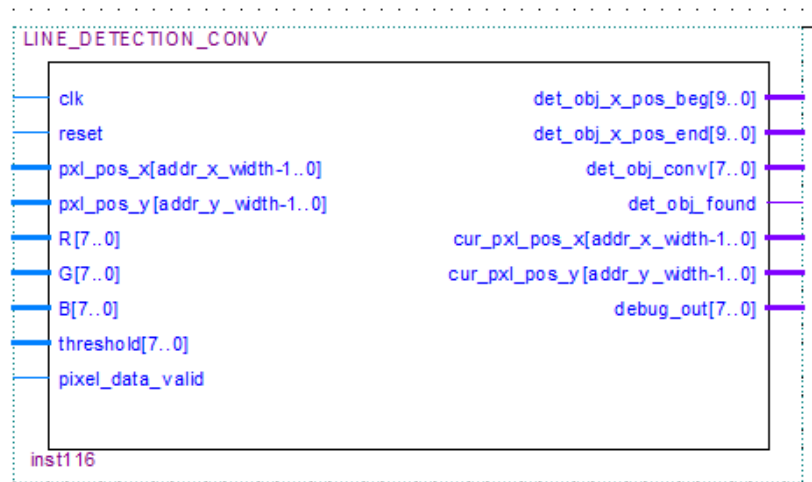


Abbildung 34: Blockschaltbild, (Quelle: Quartus Prime Lite)

#### Eingänge Modul:

- clk: Taktsignal, Takt mit dem das Modul „SDRAM\_Pixelbuffer“ arbeitet
- reset: Zustandsreset innerhalb des Moduls, HIGH-Pegel für einen Takt
- pxl\_pos\_x: X Position des Pixels aus Abbildung 26, innerhalb eines Kamerabildes
- pxl\_pos\_y: Y Position des Pixels aus Abbildung 26, innerhalb eines Kamerabildes
- R,G,B: Pixel aus dem Modul „Convolution“, Rot-,Grün- und Blaufarbkanal, Farbtiefe 8 Bit
- threshold: Mindest Unterschied der nötig ist für die Erkennung, siehe Abbildung 31
- 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

#### 4.3.1 Interner Aufbau Modul „LINE\_DETECTION\_CONV“

Ein grober Einblick in die Funktionsweise bietet Abbildung 35. In der Abbildung fehlen ausgewählte Ein- und Ausgänge, dass diese mit Hinblick auf die Beschreibung der Ein- und Ausgänge im vorherigen Kapitel, einfach zu verstehen sind.

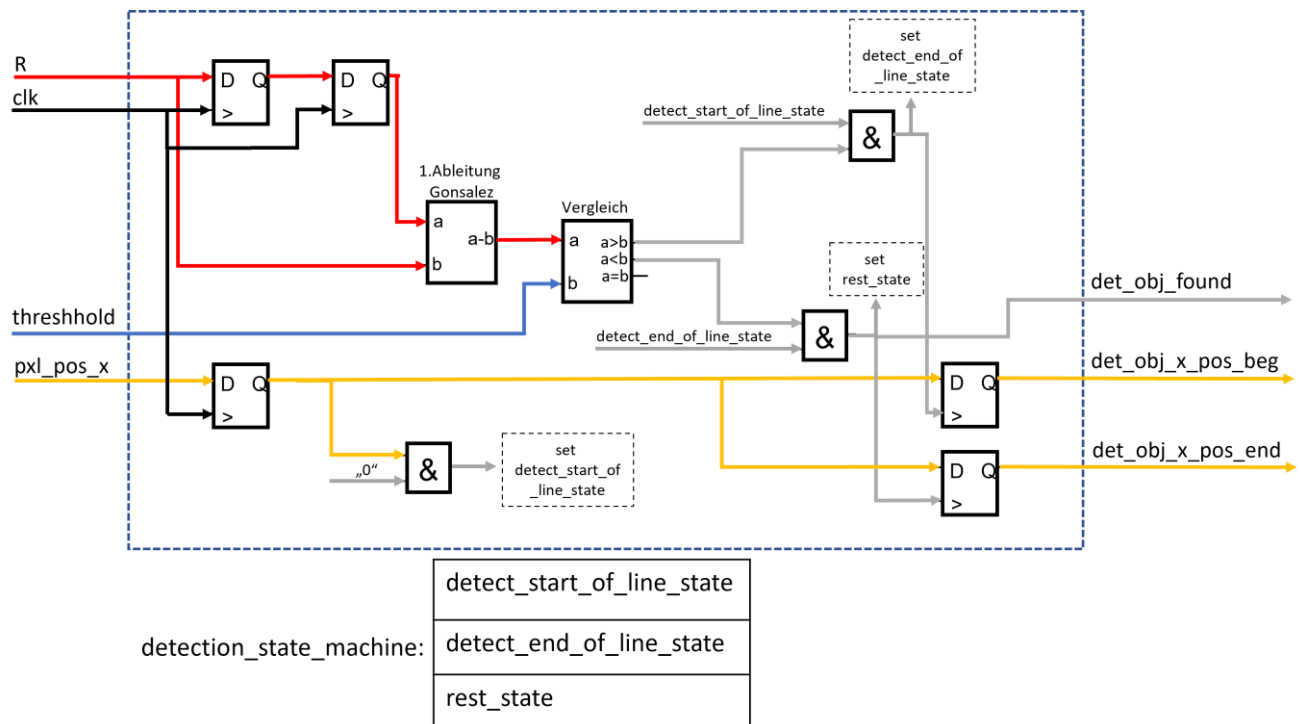


Abbildung 35: Vereinfachte Funktionsweise Modul „LINE\_DETECTION\_CONV“

Wie in Abbildung 35 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 Rames, 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. In Abbildung 36 die Ergebnisse der Simulation der Testbench.

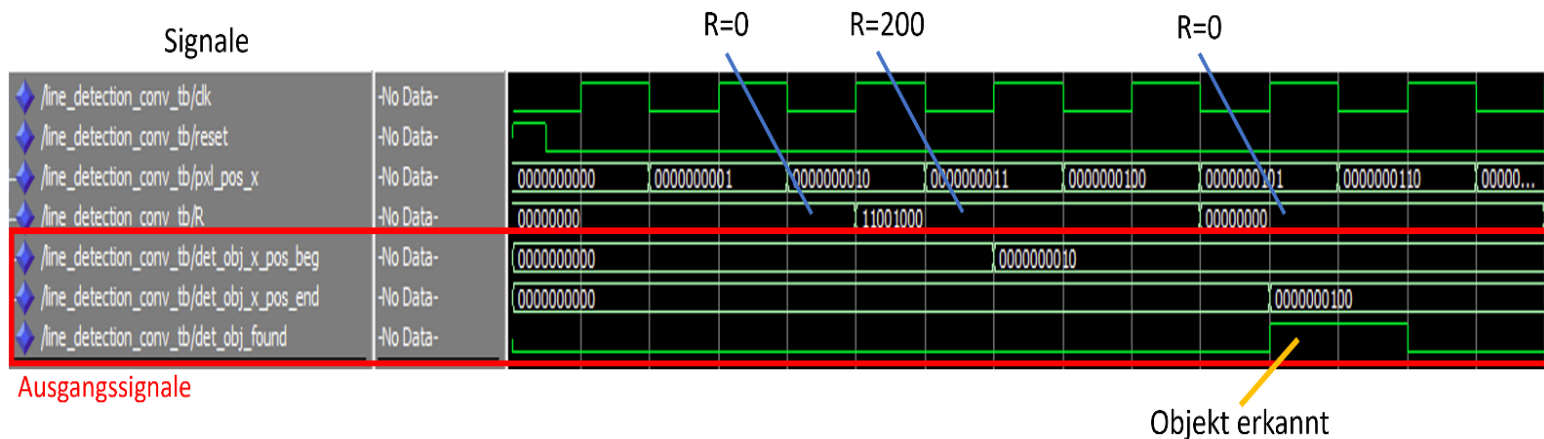


Abbildung 36: Auszug ModelSim, Simulation „LINE\_DETECTION\_CONV\_TB“

Wir betrachten Abbildung 36, 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 36 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.

#### 4.4 Modul: OBJ\_DETECTION

Das Modul „OBJECT\_DETECTION“ interpretiert, die vom 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 bestimmen und letztendlich auch damit dessen Position in seiner X- und Y-Koordinate innerhalb des Kamerabildes, die Veranschaulichung dazu in Abbildung 32.

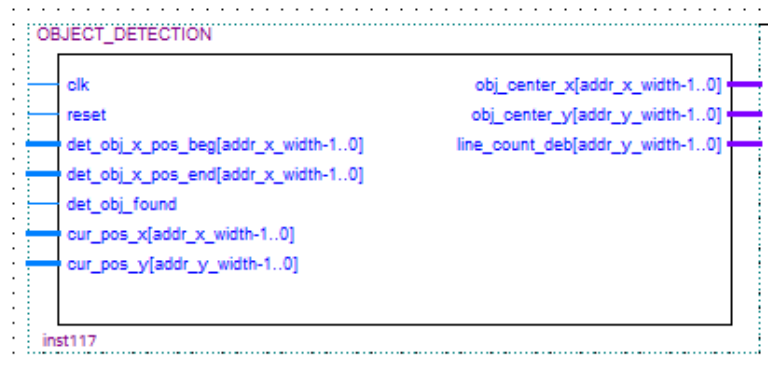


Abbildung 37: Blockschaltbild, (Quelle: Quartus Prime Lite)

##### 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 Abbildung 26, innerhalb eines Kamerabildes
- cur\_pos\_y: Y Position des Pixels aus Abbildung 26, 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: Debugging Ausgabe, Anzahl der Zeilen aus denen ein erkanntes Objekt besteht

#### 4.4.1 Interner Aufbau Modul „OBJECT\_DETECTION“

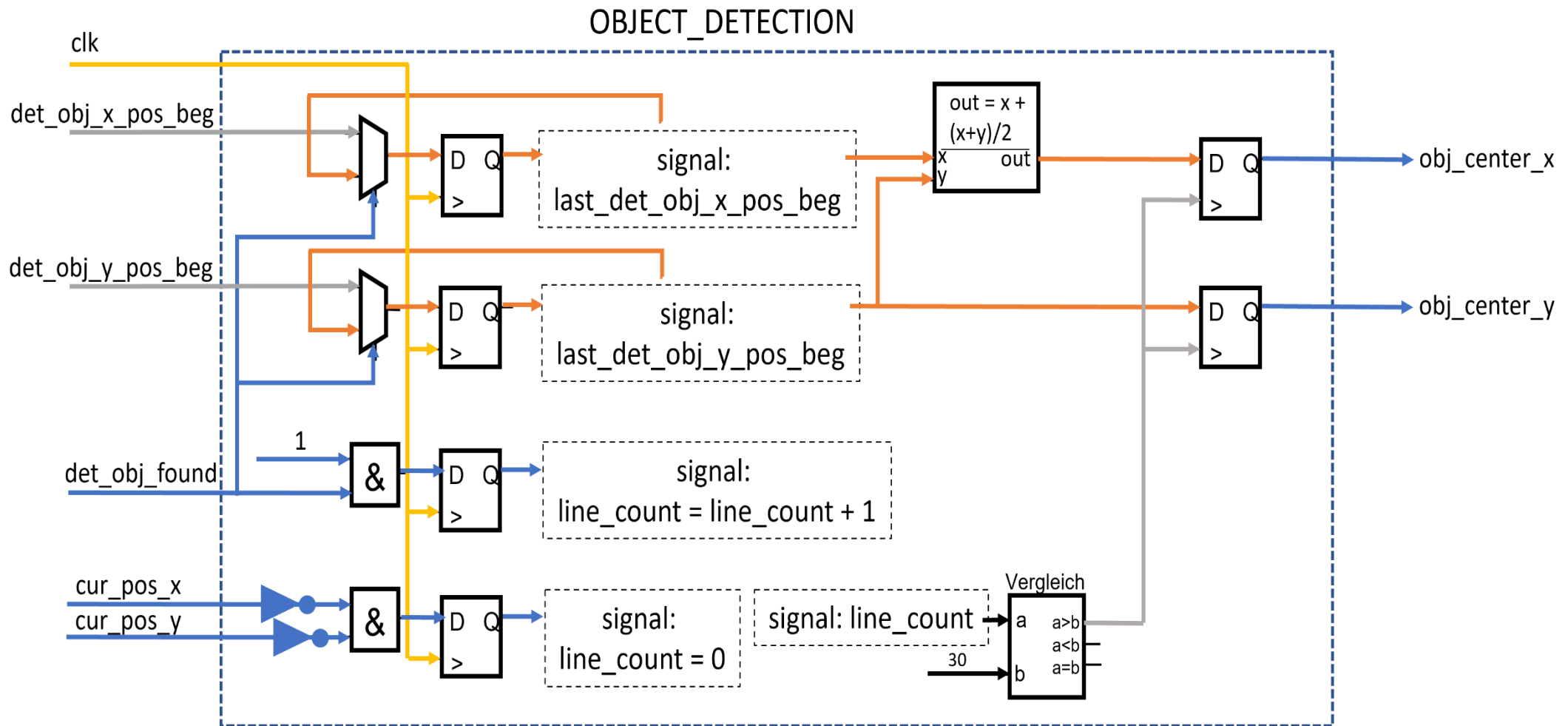


Abbildung 38: Vereinfachte Funktionsweise Modul „OBJECT\_DETECTION“

#### 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 zu sehen in Abbildung 29.

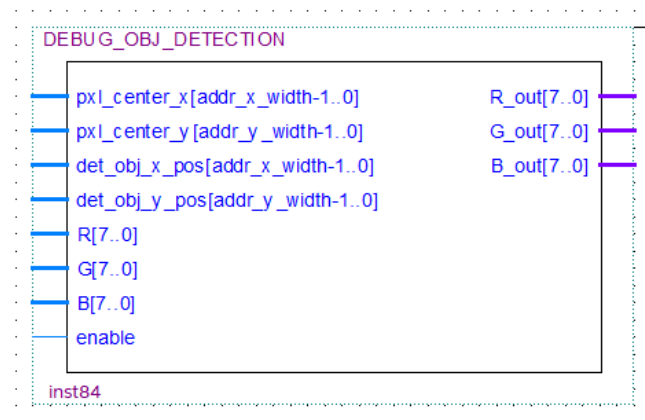


Abbildung 39: Blockschaltbild, (Quelle: Quartus Prime Lite)

##### 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.



#### 4.5.1 Interner Aufbau Modul „DEBUG\_OBJECT\_DETECTION“

Abbildung 40 zeigt die grobe Funktionsweise des Modul „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.

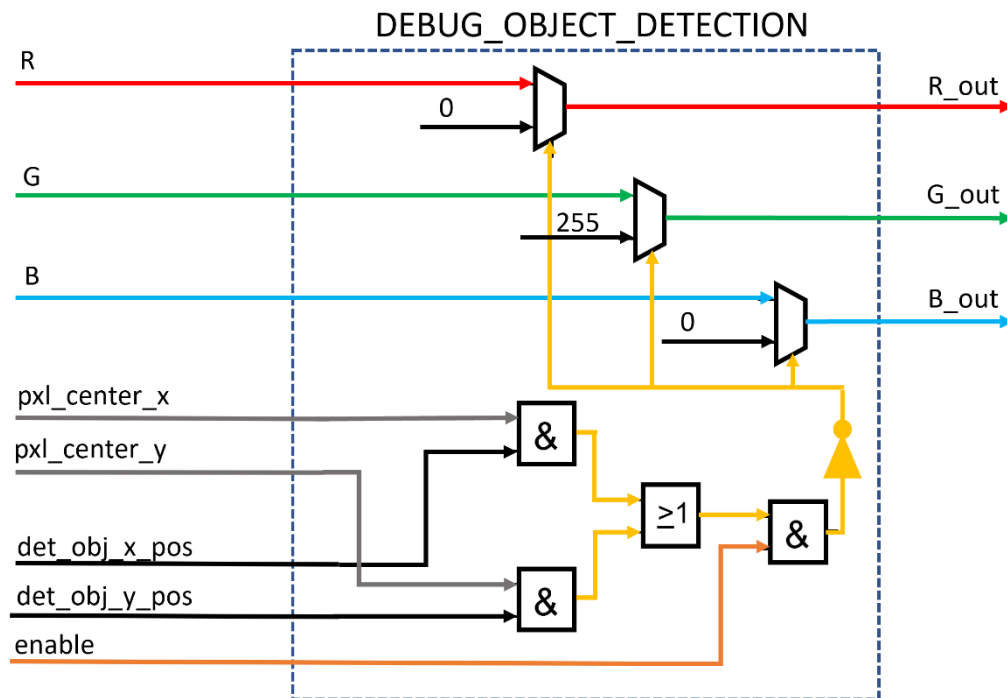


Abbildung 40: Vereinfachte Funktionsweise Modul „DEBUG\_OBJECT\_DETECTION“

## 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 Extreme, dies sind Unter- und Überbelichtung, siehe Abbildung 41, welche beide vermieden werden sollten.



Abbildung 41: Beispiel zu Beleuchtung, (Quelle: [www.flocutus.de](http://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 Threshold

Die Einstellung „threshold“ gibt an, wie groß der Unterschied zwischen zwei Nachbarpixeln sein muss, damit eine Kante eines Objekts erkannt wird. Der Unterschied berechnet sich wie in Abbildung 2 gezeigt. Für die Einstellung des „threshold“ stehen 8 Bit zur Verfügung, daher 0-255 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.

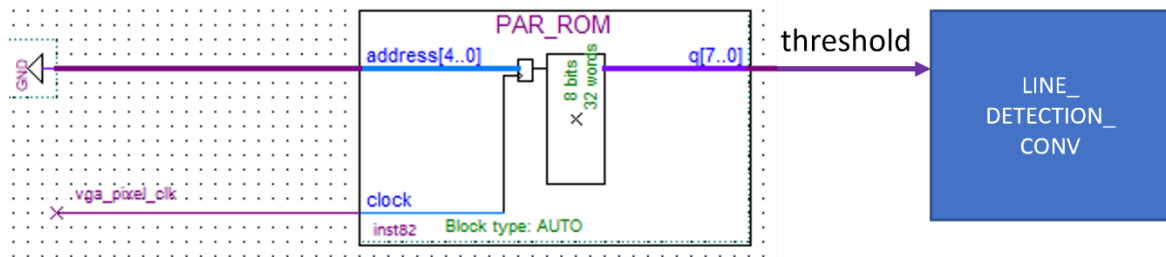


Abbildung 42: Read Only Memory Quartus Projekt, (Quelle: Quartus Prime Lite)

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 Linienobjekten 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 Refactoring 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 unter der Adresse „<https://github.com/Chriss95De/Studienarbeit.git>“. Dort kann das ganze als ZIP Archive heruntergeladen oder wie ursprünglich gedacht, als Git-Repository weitergeführt werden.

### 6.3 Erkennung von Linienobjekten innerhalb einer Bildzeile

Für die Umsetzung der Erkennung von Linienobjekten innerhalb einer Bildzeile wurden das Modul „LINE\_DETECTION\_CONV“ entworfen. Das Modul erkennt 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 einmal positiv über- und einmal negativ unterschritten wurde, siehe Abbildung 31 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.

#### 6.3.1 Problematiken

In den Überlegungen aus der Studienarbeit von Herbst (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 43 ansehen.

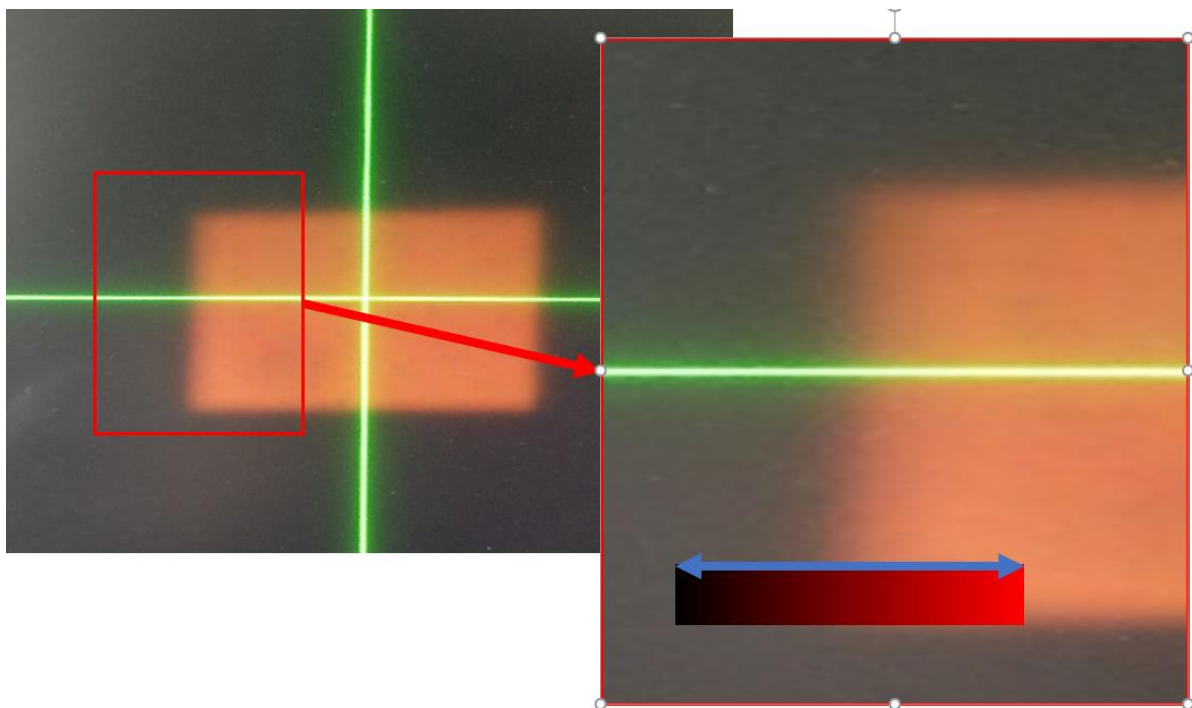


Abbildung 43: Ausschnitt rotes Objekt auf schwarzem Hintergrund, Kante mit Farbverlauf.

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 beispielsweise ein Staub Krümmel oder eine Veränderung des Winkels zwischen Szene und Kamera aus, damit die Erkennung gestört wird.

## 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\_CONV“ erhält und funktioniert auch nur wie geplant, wenn diese Daten korrekt sind. Wenn aber Probleme wie in Kapitel 6.3.1 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 wurden 7-Segmentanzeigen für jegliche Ausgaben an Informationen gegeben. So beispielsweise auch die Ausgabe der Position von isolierten Punkten und Linienobjekte in einem dimensional 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 29.

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 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. 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 zweidimensionale Positionserkennung 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 ins Leben gerufen, dieser wird Threshold genannt. Der Threshold kann über einen ROM auf dem FPGA konfiguriert werden. Mit diesen Methoden lässt sich ein Linienobjekt definieren.

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.

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

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 Implementierung in Betracht gezogen werden und mit Alternativen verglichen werden. Die implementierte Erkennung funktioniert in seinen Grundzügen, jedoch existiert kein wissenschaftlicher Beweis, ob diese von Relevanz ist.

Alle Ergebnisse aus diesem Projekt, sowie vorherige Projektstände und alle dazu relevanten Informationen wurden in einem Repository geordnet und zusammengefasst. Damit es für eine Person die dieses Projekt fortsetzt einfach ist.



## 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.floctus.de/besser-fotografieren-belichtungbetter-photography-exposure/> 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
- Rafael C. Gonzalez, R. E. (1987). *Digital Image Processing, Third Edition, 978-0131687288*. Prentice Hall.
- Technologies, T. (2013). *DE2-115 User Manuel*.