

Studienarbeit

Christian Oster

Inhaltsverzeichnis

Einleitung.....	4
Digitale Bildverarbeitung.....	5
Funktionsweise Darstellung von Pixeln.....	5
Farbunterschiede erkennen	6
Übersicht Zustand vorher.....	7
Modul: cameraink_tapping_base.....	8
Kameramodi: Taps, horizontales und vertikales Timing	8
Horizontales Timing 2 Tap	9
Vertikales Timing	9
Modul: camera_data_mux_gen	10
Aufbau Intern: camera_data_mux_gen (Vereinfachte Version).....	10
Modul: SDRAM_Write_Buffer_gen	11
Aufbau Intern: SDRAM_Write_Buffer_gen (Vereinfachte Version).....	11
Modul: Memory Access Control.....	12
Funktionsweise Interface	12
Modul: SDRAM Controller	13
Modul: SDRAM_Read_Buffer_gen	14
Interne Buffer Struktur	14
Modul: SDRAM_Pixelbuffer.....	15
Aufbau Intern: SDRAM_Pixelbuffer(Vereinfachte Version)	16
Modul: Debay	17
Modul: Convolution.....	19
VGA Ausgabe	20
Übersicht Zustand (nach Studienarbeit Oster).....	21
Funktionsweise/Anknüpfung an Vorheriges	22
Übersicht aktueller Zustand	23
Modul: LINE_DETECTION_CONV	24
Interner Aufbau Modul „LINE_DETECTION_CONV“	25
Testbench „LINE_DETECTION_CONV_TB“	25
Modul: OBJ_DETECTION.....	27
Modul: DEB_OBJ_DETECTION	28
Literaturverzeichnis.....	29

ABBILDUNG 1: DIGITALES BILD BEISPIEL	5
ABBILDUNG 2: PIXELREIHE $F(x)$ UND 1. ABLEITUNG $F'(x)$, GRAUSTUFEN IN 8 BIT DARSTELLUNG	6
ABBILDUNG 3: ÜBERSICHT DER MODULE INNERHALB DES BISHERIGEN QUARTUSPROJEKT	7
ABBILDUNG 4: ÜBERSICHT VERWENDETE HARDWARE.....	7
ABBILDUNG 5: BLOCKSCHALTBILD	8
ABBILDUNG 6: ÜBERSICHT TIMING HORIZONTAL, MODI 2TAP, 642 PIXEL HORIZONTAL.....	9
ABBILDUNG 7: ÜBERSICHT TIMING VERTIKAL, 484 ZEILEN VERTIKAL	9
ABBILDUNG 8: BLOCKSCHALTBILD	10
ABBILDUNG 9: VEREINFACHTE FUNKTIONSWEISE MODUL „CAMERA_DATA_MUX_GEN“	10
ABBILDUNG 10: BLOCKSCHALTBILD	11
ABBILDUNG 11: VEREINFACHTE FUNKTIONSWEISE MODUL „SDRAM_WRITE_BUFFER_GEN“	11
ABBILDUNG 12: BLOCKSCHALTBILD „MEMORYACCESSCONTROLLER“ MIT PARAMETERN	12
ABBILDUNG 13: FUNKTIONSWEISE INTERFACE MEMORYACCESSCONTROLLER.....	12
ABBILDUNG 14: BLOCKSCHALTBILD + BESCHALTUNG	13
ABBILDUNG 15: BLOCKSCHALTBILD	14
ABBILDUNG 16: INTERNE BUFFER STRUKTUR DES MODULS „SDRAM_READ_BUFFER_GEN“	14
ABBILDUNG 17: BLOCKSCHALTBILD, ROT MARKIERT DIE WICHTIGSTEN IOS	15
ABBILDUNG 18: SCHAUBILD FUNKTIONSWEISE PIXELBUFFER	15
ABBILDUNG 19 SCHAUBILD PIXELBUFFER ZEILENENDE	16
ABBILDUNG 20 SCHAUBILD INTERNER AUFBAU SDRAM_PIXELBUFFER.....	16
ABBILDUNG 21 BLOCKSCHALTBILD	17
ABBILDUNG 22 AUFTEILUNG 5x5 BYTE BLOCK IN KLEINERE 3x3 BLÖCKE	17
ABBILDUNG 23: ZUSAMMENSETZUNG 3x3, BEISPIEL AM ROTKANAL, R11 = OUTPUT DER ERSTEN ZEILE DES ROTKANALS	18
ABBILDUNG 24: BLOCKSCHALTBILD	19
ABBILDUNG 25: GEWICHTUNG DES 3x3 ROT KANAL FÜR BERECHNUNG ROT KANAL.....	19
ABBILDUNG 26: BERECHNUNG FARBWERT R.....	19
ABBILDUNG 27: SCHALTBILD VGA AUSGABE.....	20
ABBILDUNG 28: TESTAUFBAU ERKENNUNG ROTES RECHTECK AUF BILDSCHIRM	21
ABBILDUNG 29: ZIELKREUZ AUF ERKANNTER POSITION	21
ABBILDUNG 30: BEISPIEL ERKENNUNG ROTES RECHTECK	22
ABBILDUNG 31: ÄNDERUNG DER INTENSITÄT DES ROTKANALS, 1.ABLEITUNG NACH GONSALEZ.....	22
ABBILDUNG 32: ÜBERSICHT ZUSTAND MODULE NACH STUDIENARBEIT OSTER,2021.....	23
ABBILDUNG 33: BLOCKSCHALTBILD	24
ABBILDUNG 34: VEREINFACHTE FUNKTIONSWEISE MODUL „LINE_DETECTION_CONV“	25
ABBILDUNG 35: AUSZUG MODEL SIM, SIMULATION „LINE_DETECTION_CONV_TB“	25

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. [Definition Regelung nach DIN 19226]

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.

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.

...Bezug auf den Aufbau der Studienarbeit

Digitale Bildverarbeitung

Diese Studienarbeit stützt sich in vielen Abschnitten aus Erkenntnissen 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.

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. Jeder Pixel kann mit der Funktion $f(x,y)$ einzeln ausgelesen werden.

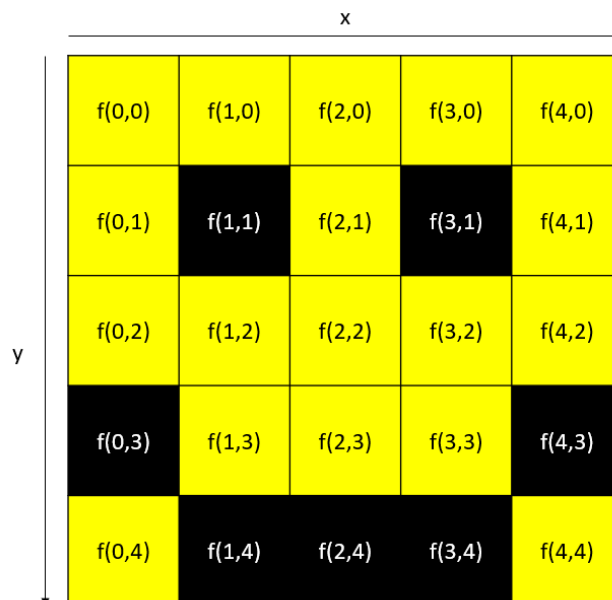


Abbildung 1: Digitales Bild Beispiel

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 maximal 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.

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.

Übersicht Zustand vorher

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

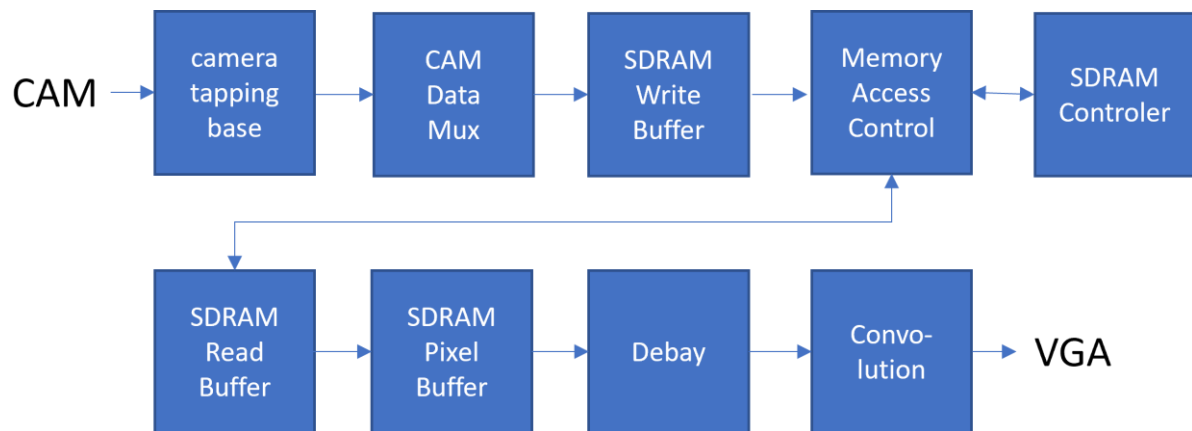


Abbildung 3: Übersicht der Module innerhalb des bisherigen Quartusprojekt

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.

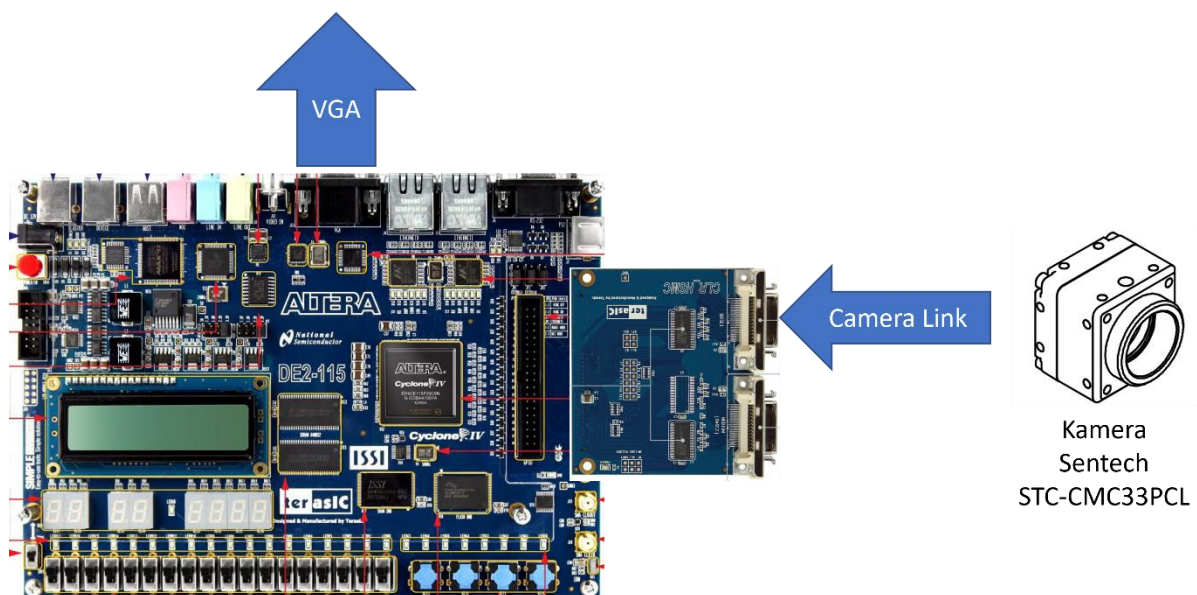


Abbildung 4: Übersicht verwendete Hardware

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.

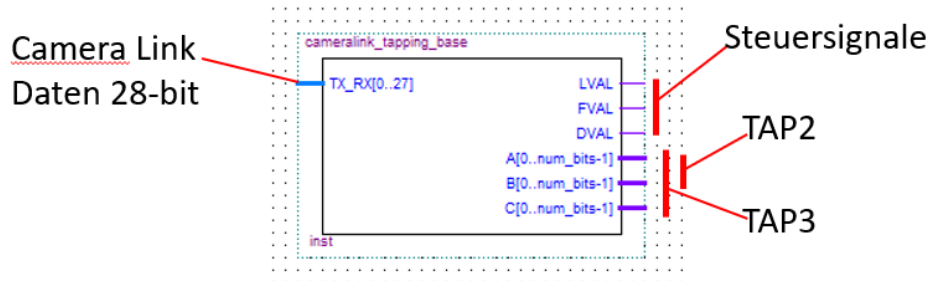


Abbildung 5: Blockschaltbild

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.

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.

Horizontales Timing 2 Tap

1CLK = 11.9 nsec.

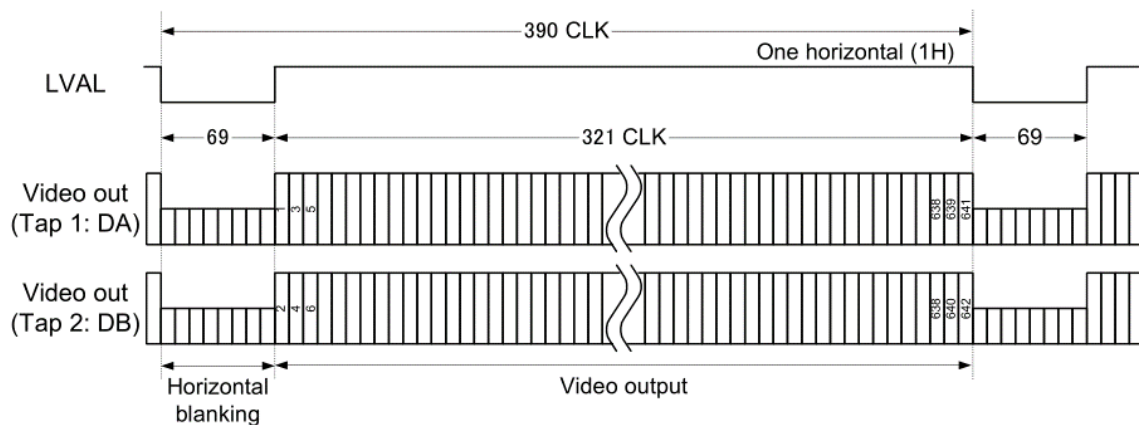


Abbildung 6: Übersicht Timing horizontal, Modi 2Tap, 642 Pixel horizontal

„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.

Vertikales Timing

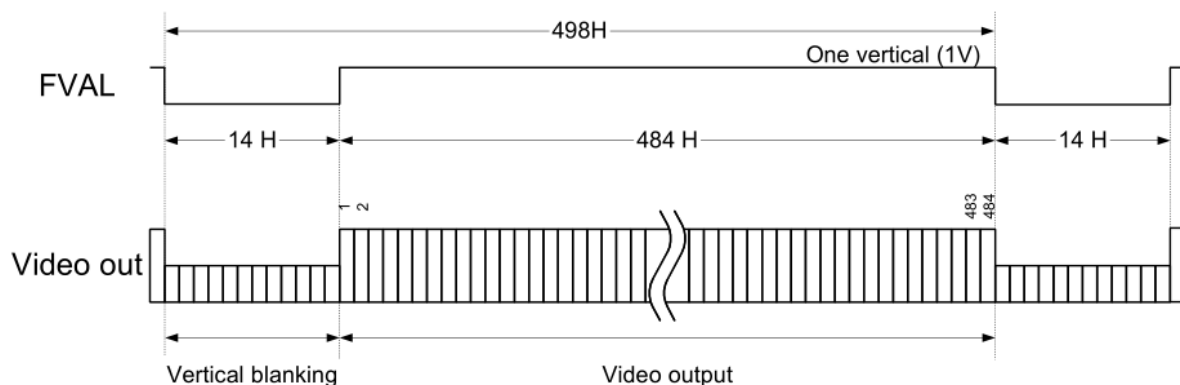


Abbildung 7: Übersicht Timing vertikal, 484 Zeilen vertikal

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) ausschlaggebend, sondern die Anzahl der vergangenen Zeilen. So steht „498H“ für 498 Zeilen die vergangen sind. (H = horizontals).

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.

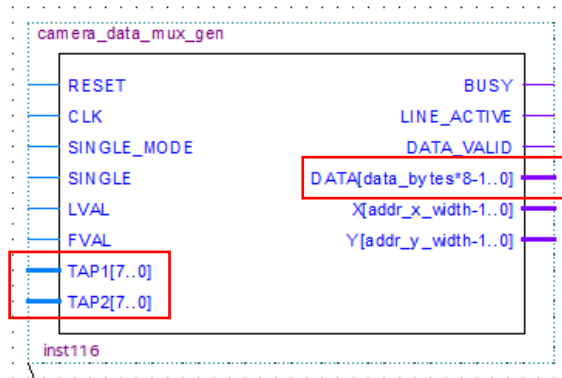


Abbildung 8: Blockschaltbild

Aufbau Intern: camera_data_mux_gen (Vereinfachte Version)

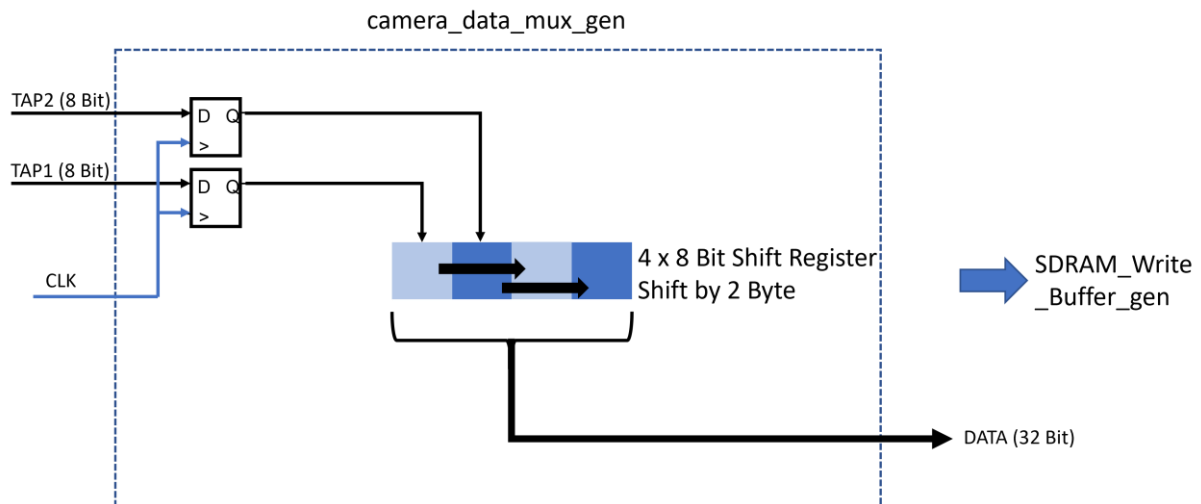


Abbildung 9: Vereinfachte Funktionsweise Modul „camera_data_mux_gen“

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.

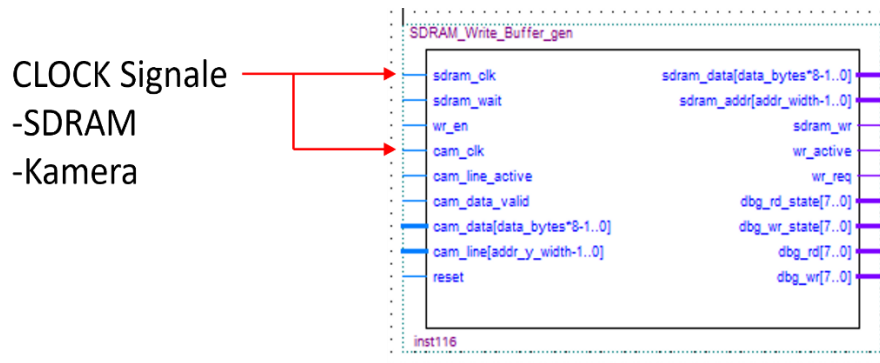


Abbildung 10: Blockschaltbild

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

Aufbau Intern: SDRAM_Write_Buffer_gen (Vereinfachte Version)

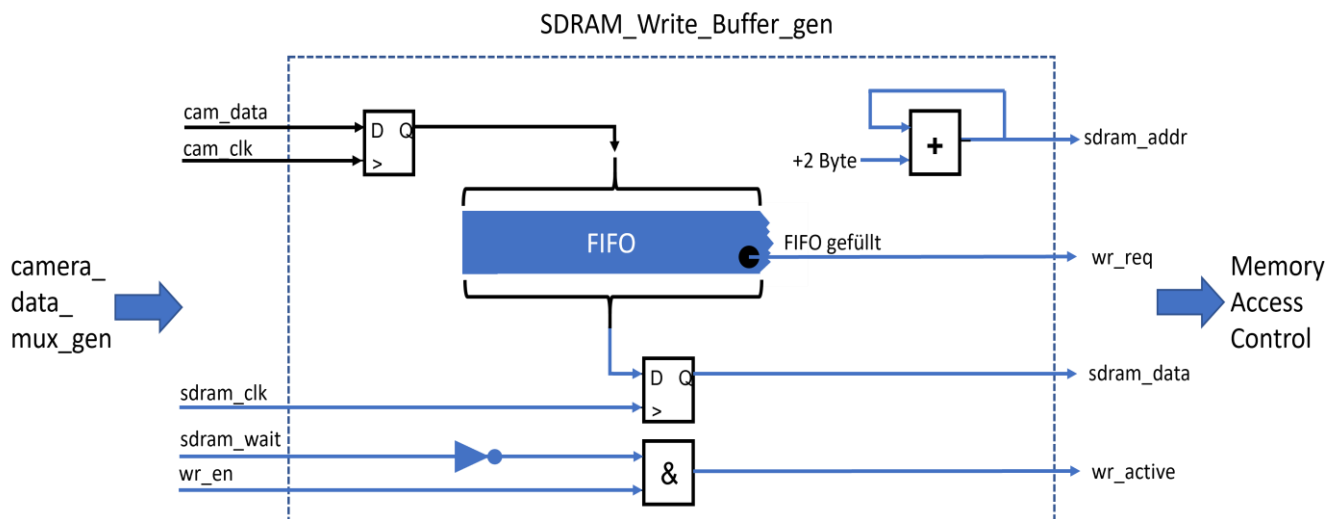


Abbildung 11: Vereinfachte Funktionsweise Modul „SDRAM_Write_Buffer_gen“

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.

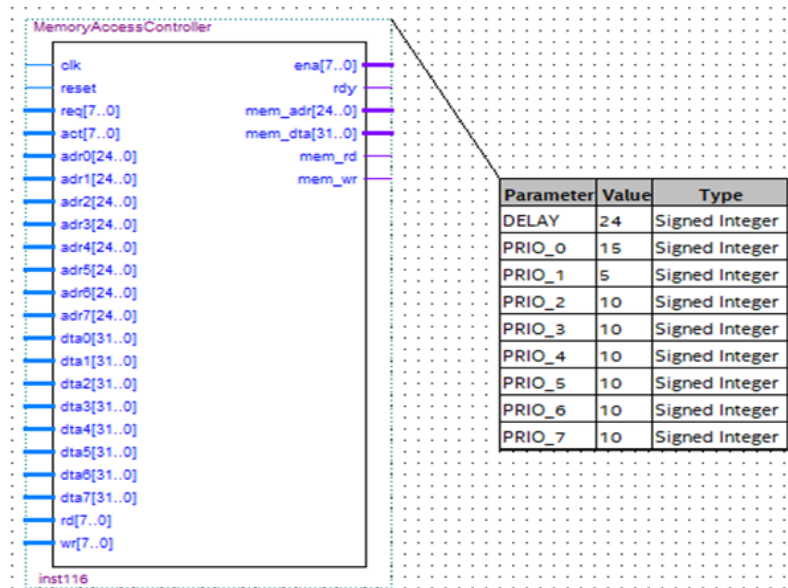


Abbildung 12: Blockschaltbild „MemoryAccessController“ mit Parametern

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

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.

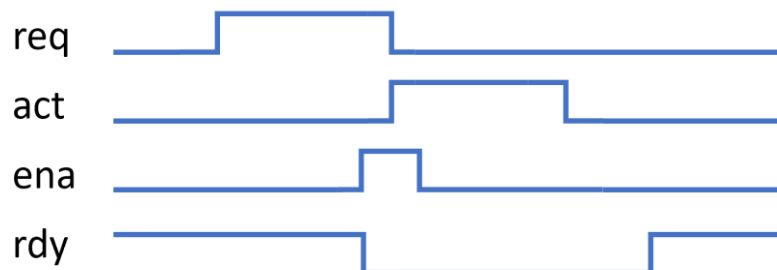


Abbildung 13: Funktionsweise Interface MemoryAccessController

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>.

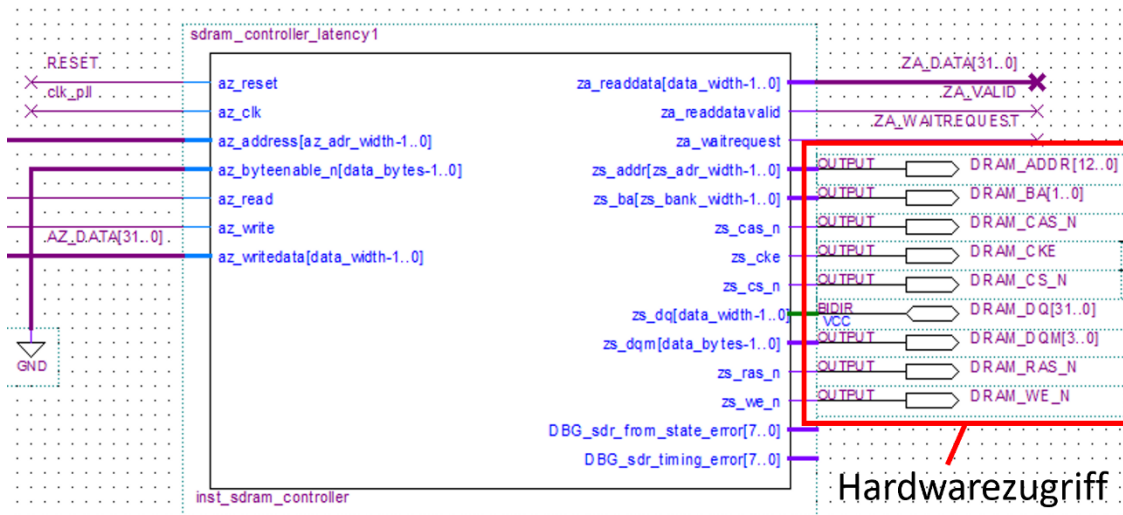


Abbildung 14: Blockschaltbild + Beschaltung

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_readdatevalid: Daten in „za_readdata“ sind gültig bei logischen High-Pegel
- za_waitrequest: Bei logischen High-Pegel werden keine Lese- und Schreibenanforderungen bearbeitet, SDRAM ist beschäftigt

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.

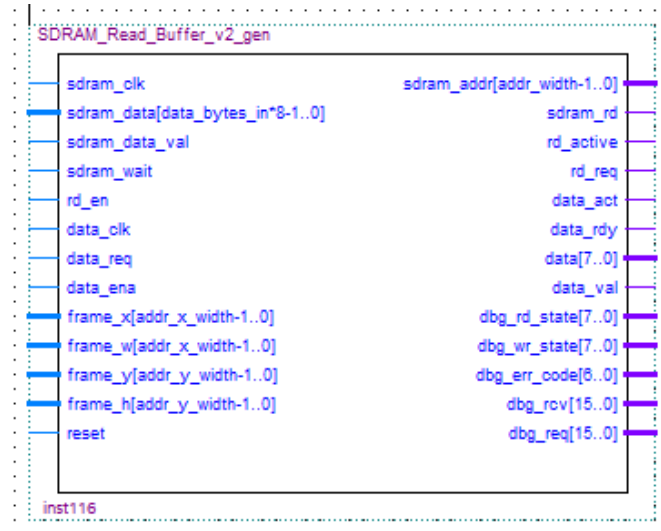


Abbildung 15: Blockschaltbild

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.

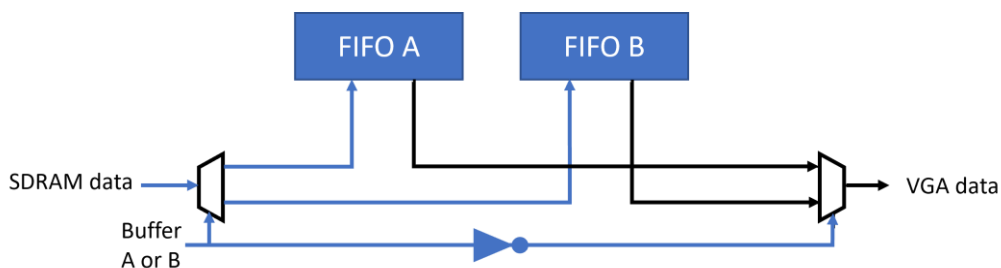


Abbildung 16: Interne Buffer Struktur des Moduls „SDRAM_Read_buffer_gen“

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.

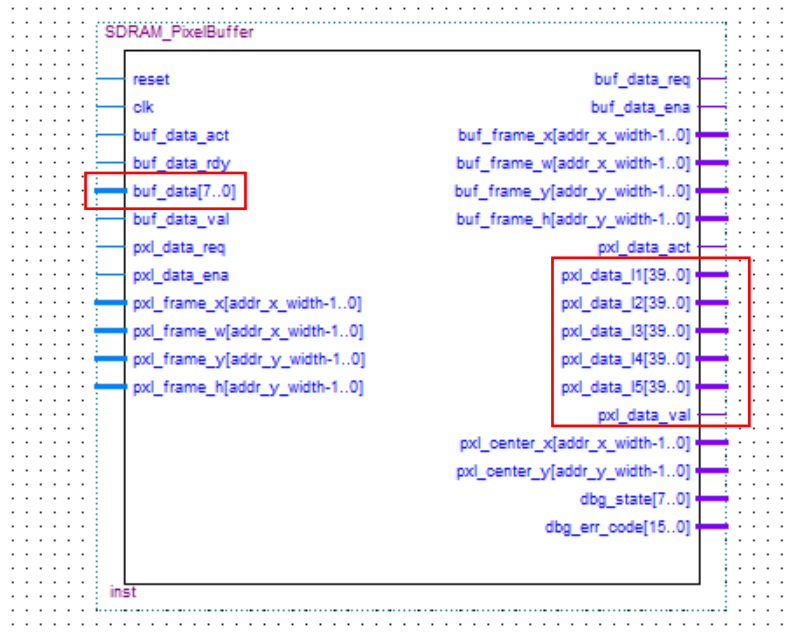


Abbildung 17: Blockschaltbild, rot markiert die wichtigsten IOs

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.

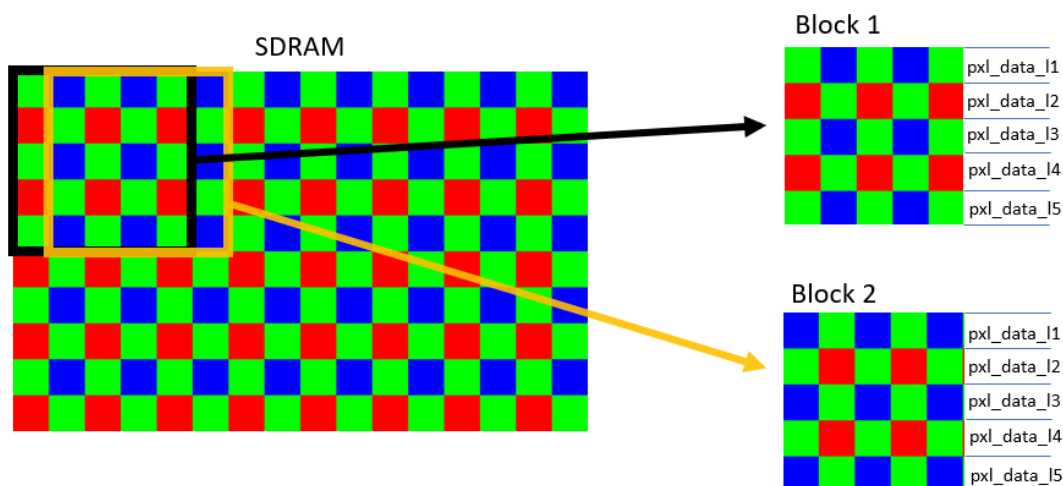


Abbildung 18: Schaubild Funktionsweise Pixelbuffer

Die fünf Ausgänge, für jeweils eine Zeile, sind auf den Eingang „pxl_data_l1 ... pxl_data_l5“ 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.

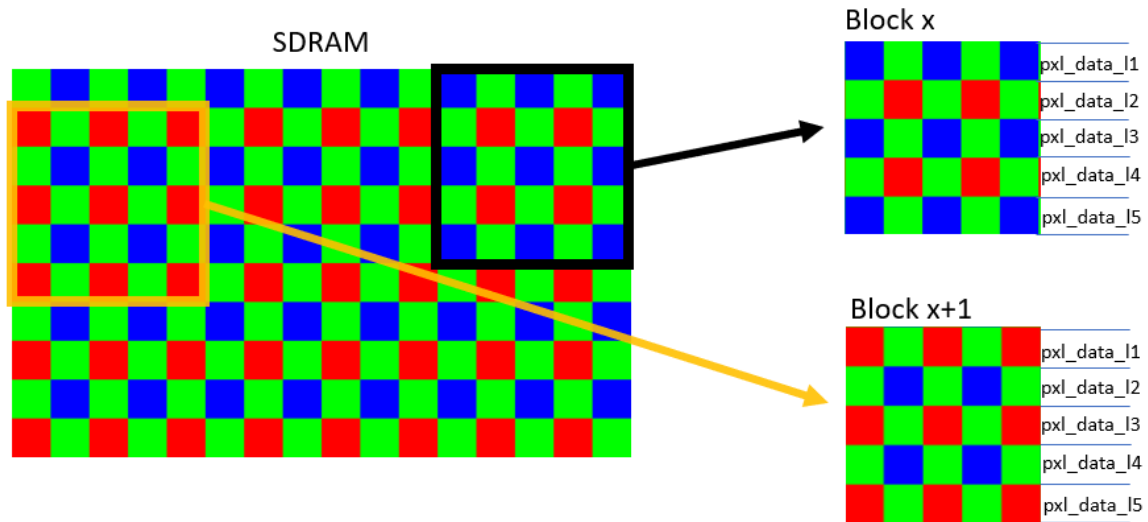


Abbildung 19 Schaubild Pixelbuffer Zeilenende

Aufbau Intern: SDRAM_Pixelbuffer(Vereinfachte Version)

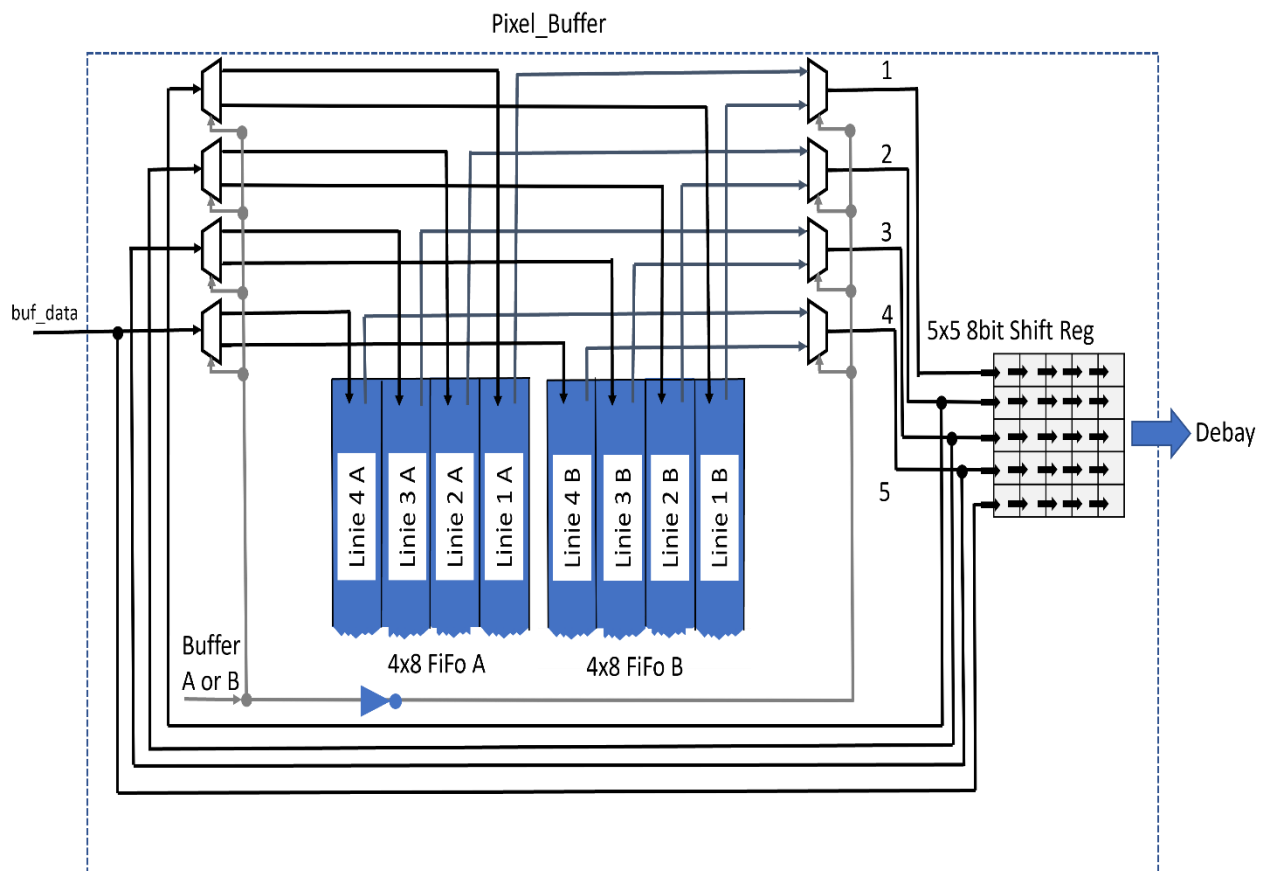


Abbildung 20 Schaubild interner Aufbau SDRAM_Pixelbuffer

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.

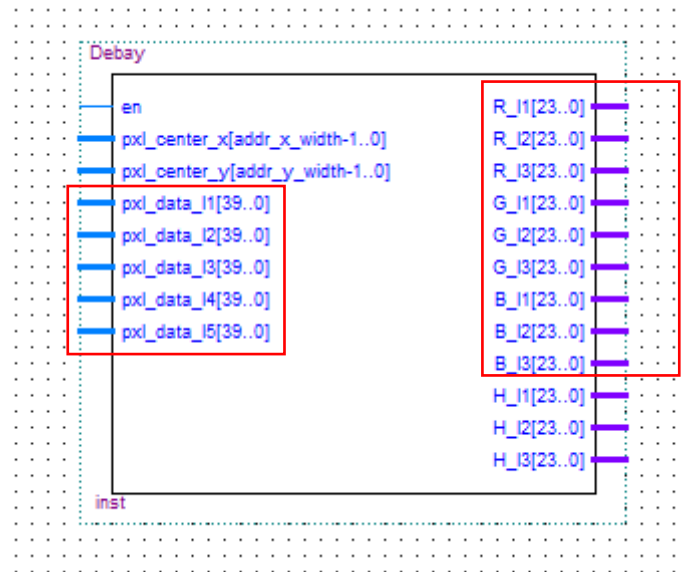


Abbildung 21 Blockschaltbild

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.

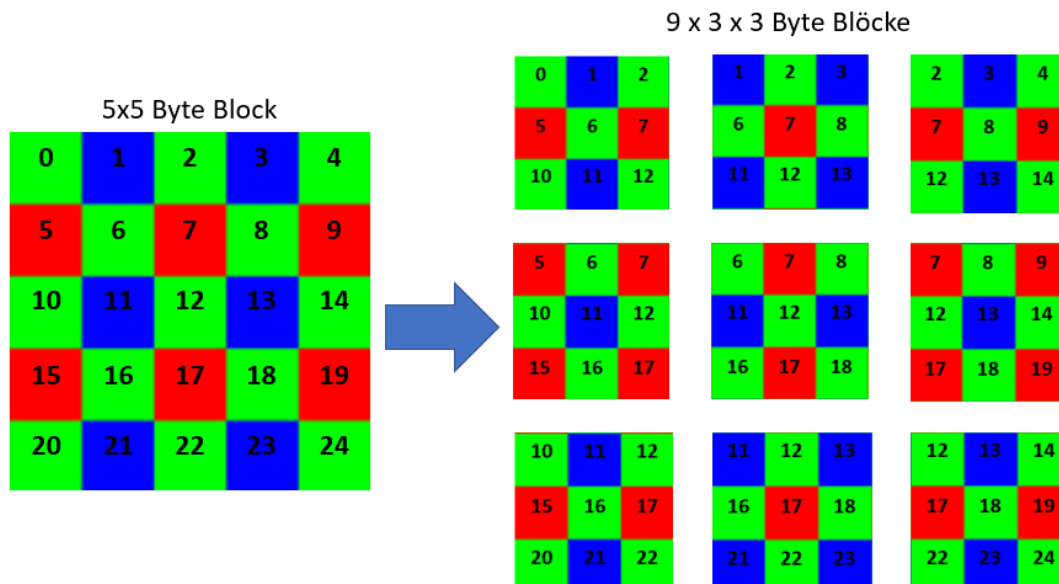


Abbildung 22 Aufteilung 5x5 Byte Block in kleinere 3x3 Blöcke

Durch die Aufteilung erhalten wir neun 3x3 Byte Blöcke. Aus jedem Block wird für alle drei Farbkänäle ein Byte extrahiert. Wie in Abbildung Debay 2 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.

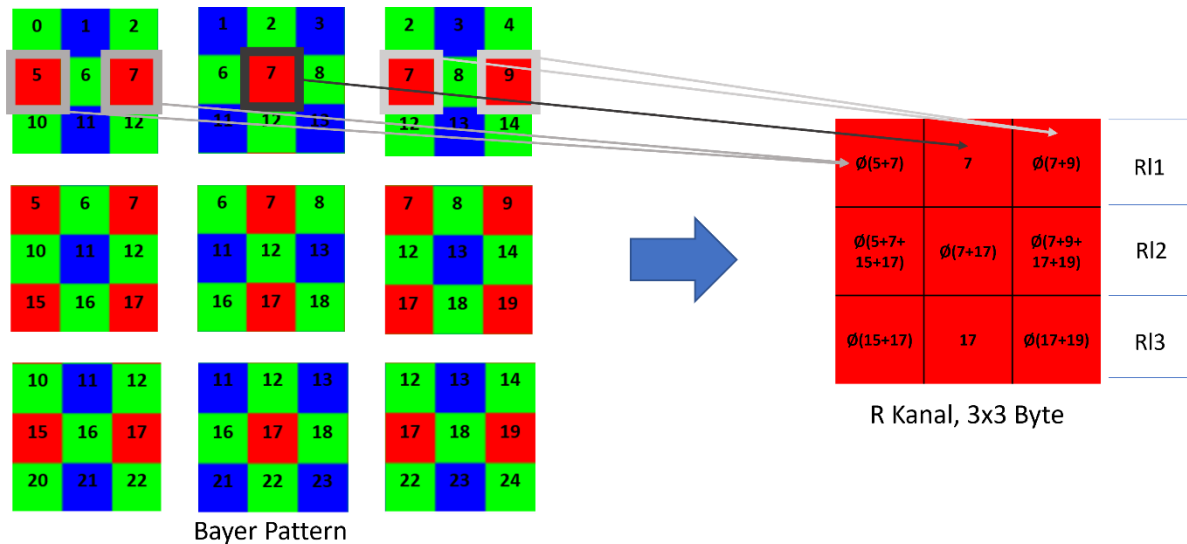


Abbildung 23: Zusammensetzung 3x3, Beispiel am Rotkanal, RI1 = Output der ersten Zeile des Rotkanals

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.

Modul: Convolution

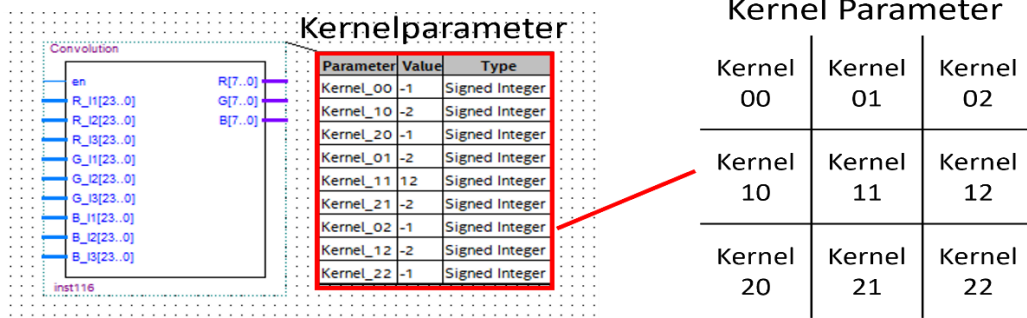


Abbildung 24: Blockschaltbild

„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.

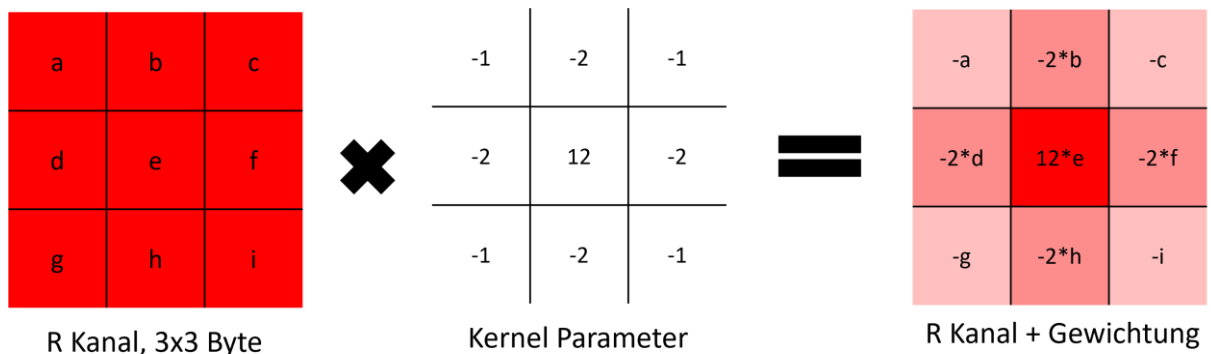


Abbildung 25: Gewichtung des 3x3 Rot Kanal für Berechnung Rot Kanal

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.

R1	-a	-2*b	-c
R2	-2*d	12*e	-2*f
R3	-g	-2*h	-i

R Kanal + Gewichtung

$$\emptyset = \frac{(-a - 2b - c - 2d + 12e - 2f - g - 2h - i)}{9} = R[0..7]$$

Abbildung 26: Berechnung Farbwert R

VGA Ausgabe

Für die Ausgabe der Bilddaten ist das Modul „vga_controller“ zuständig. Eine gute Übersicht über die VGA Spezifikation kann unter <https://forum.digikey.com/t/vga-controller-vhdl/12794> nachgelesen werden und wird an dieser Stelle nicht weiter erläutert.

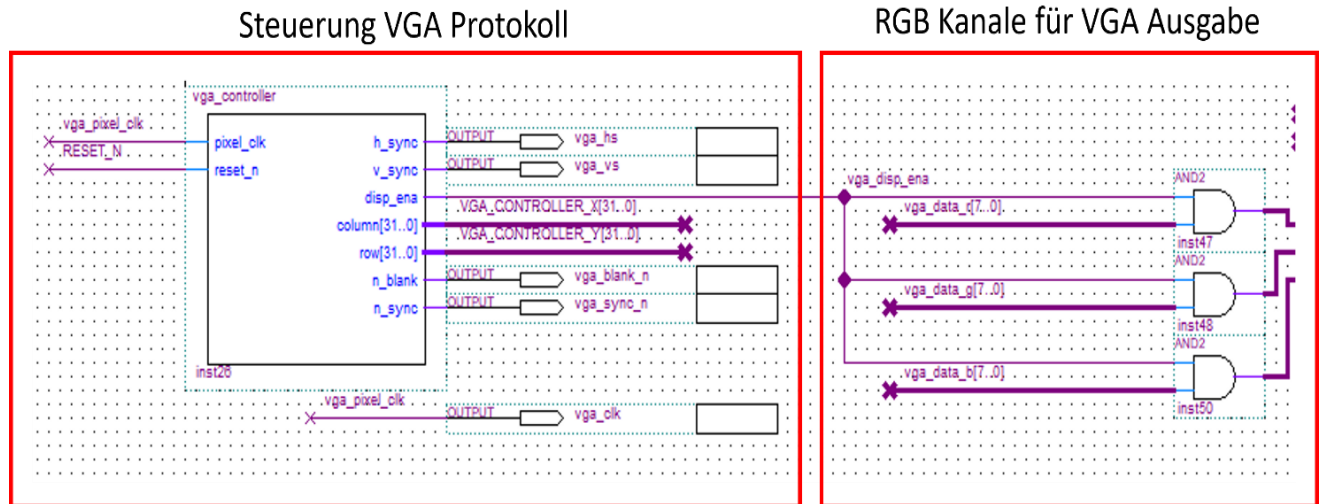


Abbildung 27: Schaltbild VGA Ausgabe

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.

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 Kamerablickwinkels 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.

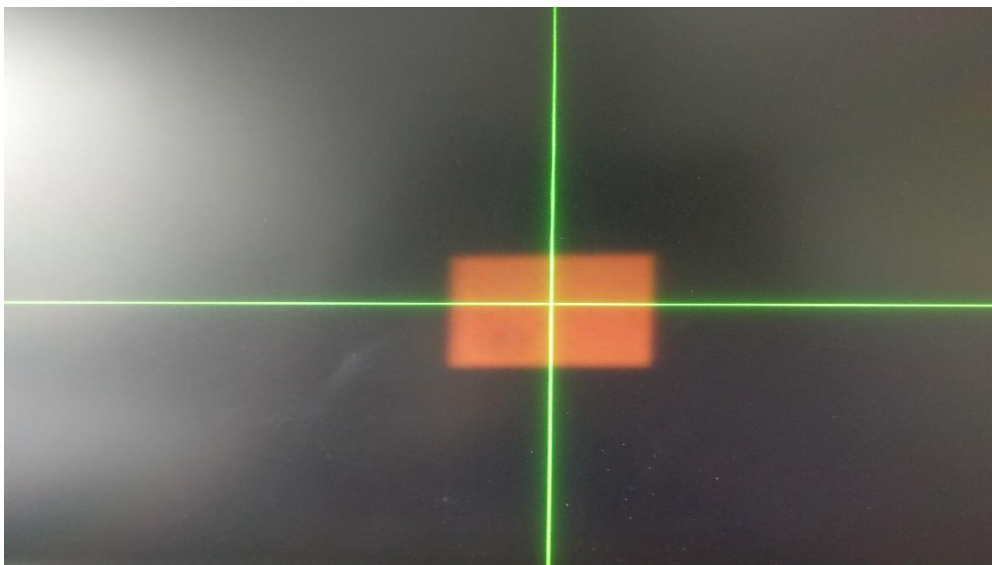


Abbildung 29: Zielkreuz auf erkannter Position

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.

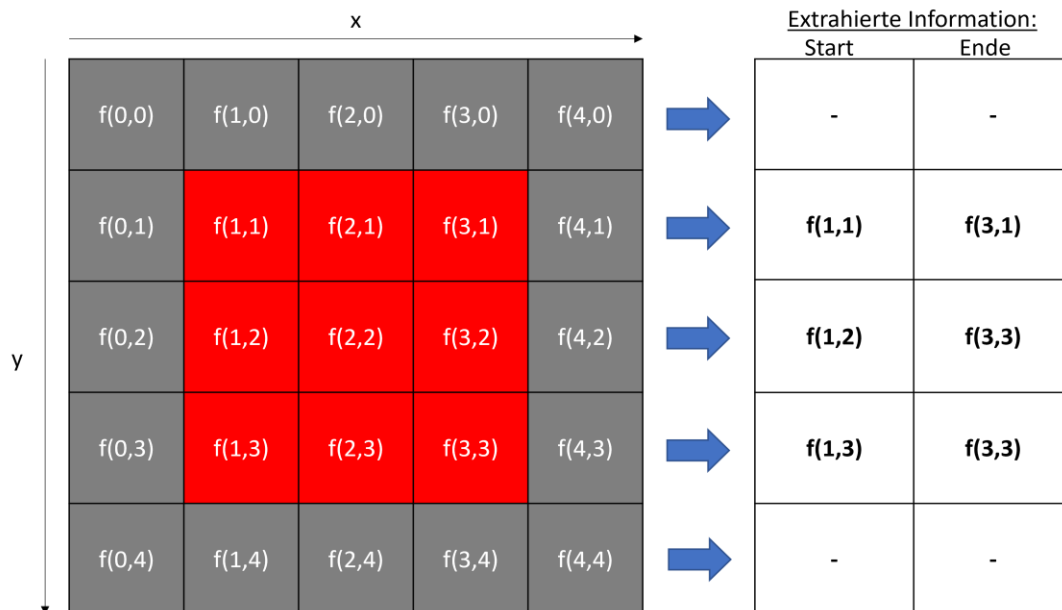


Abbildung 30: Beispiel Erkennung rotes Rechteck

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 (THRESHHOLD) beidem bestimmt werden kann, dass wir einen bestimmten Wechsel von wenig auf viel Rotanteil haben. Das Ganze in folgender Abbildung nochmal verdeutlicht.

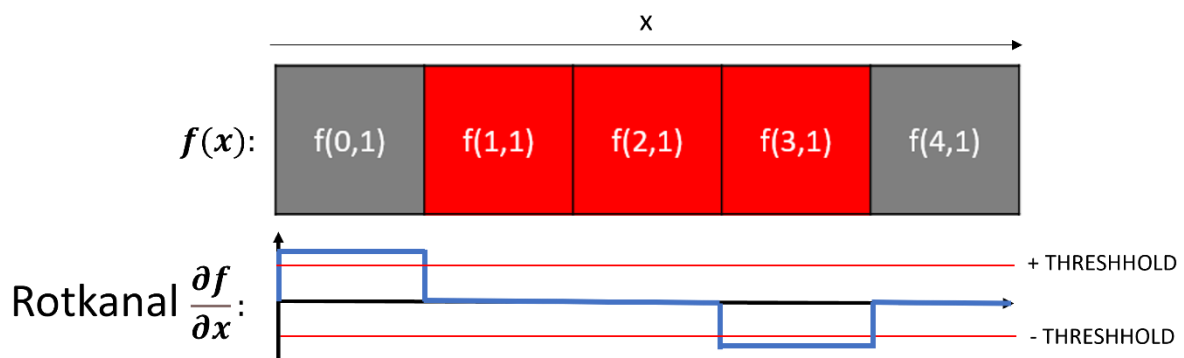


Abbildung 31: Änderung der Intensität des Rotkanals, 1. Ableitung nach Gonzalez

Abbildung 31 zeigt das Anfang und Ende eines Objektes über die Änderung der Farbintensität herauszufinden ist. 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.

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.

Übersicht aktueller Zustand

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

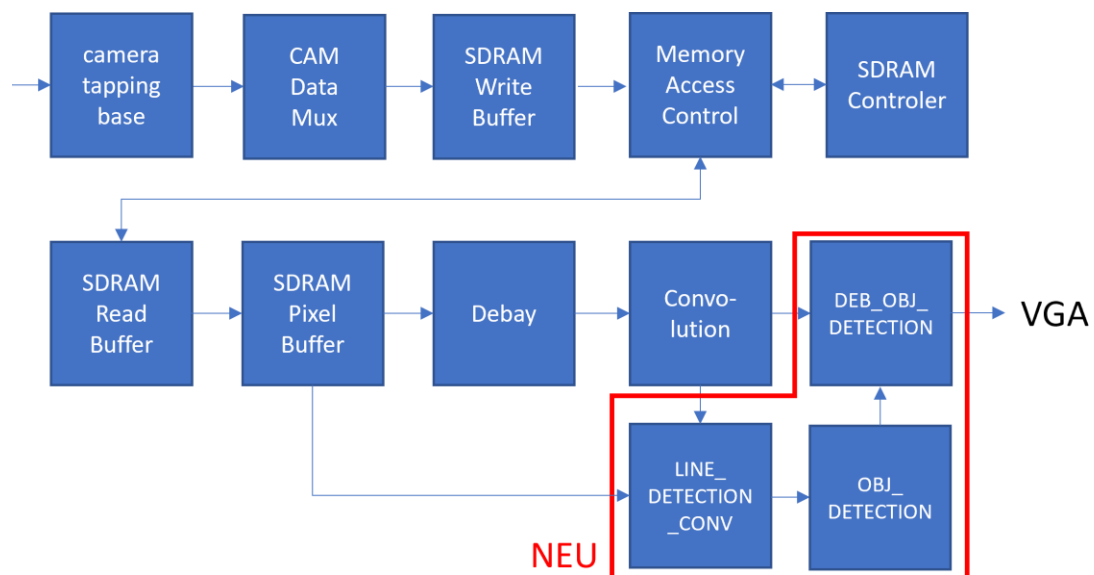


Abbildung 32: Übersicht Zustand Module nach Studienarbeit Oster,2021

Die neuen Module aus Abbildung 32, rot markiert, arbeiten auf den Ausgangssignalen des Moduls „Convolution“. In den folgenden Kapiteln werden die Module im Detail beschrieben.

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.

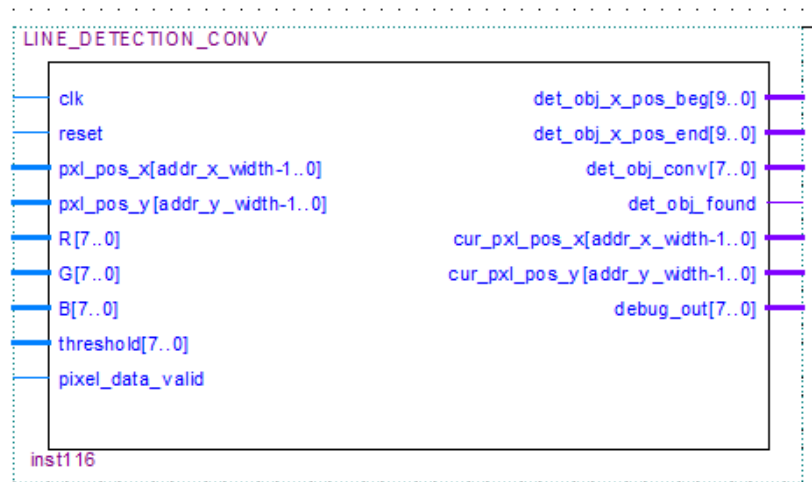


Abbildung 33: Blockschaltbild

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 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 Blaufarbkanaal, 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
- 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

Interner Aufbau Modul „LINE_DETECTION_CONV“

Ein grober Einblick in die Funktionsweise bietet Abbildung 34. 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.

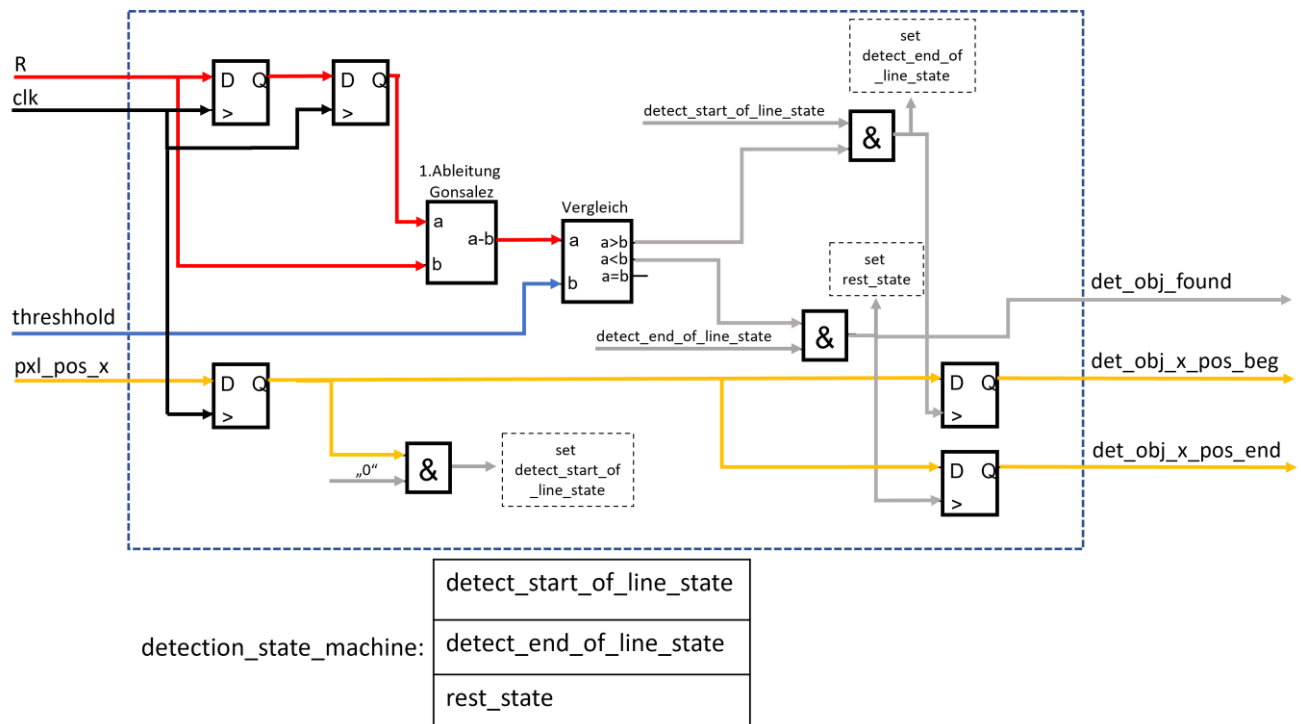
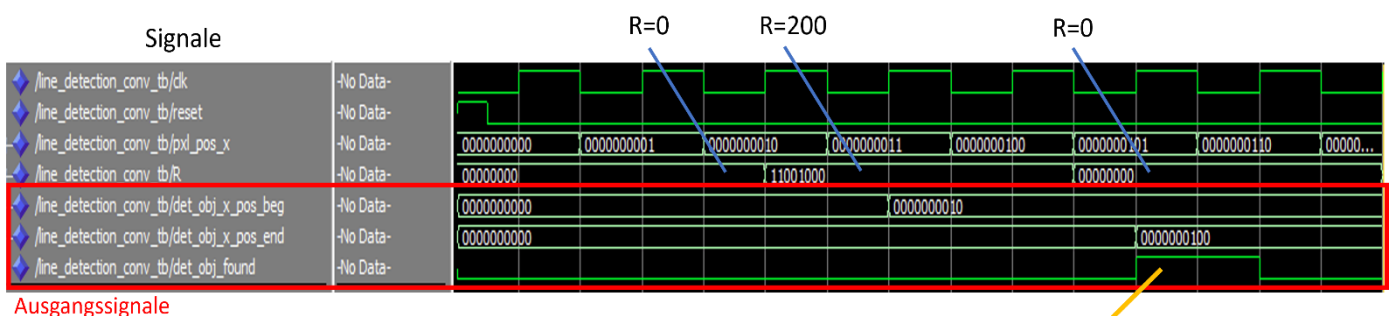


Abbildung 34: Vereinfachte Funktionsweise Modul „LINE_DETECTION_CONV“

Wie in Abbildung 34 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.

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 35 die Ergebnisse der Simulation der Testbench.



Ausgangssignale

Objekt erkannt

Abbildung 35: Auszug ModelSim, Simulation „LINE_DETECTION_CONV_TB“

Modul: OBJ_DETECTION

Modul: DEB_OBJ_DETECTION

Literaturverzeichnis

(kein Datum).

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

