

Hochschule Koblenz  
Fachbereich Ingenieurwesen  
Elektro- und Informationstechnik



Studienarbeit

# Bilddatenvorverarbeitung in einem FPGA

Autor: Lukas Herbst  
Matrikelnummer: 532903  
25. Januar 2020  
Studiengang: Elektrotechnik

**Betreuer:** Prof. Dr. Berthold Gick  
**Zeitraum:** 26. September 2019 - 25. Januar 2020

## Kurzzusammenfassung

In dieser Studienarbeit wird das Thema der *Bilddatenvorverarbeitung in einem FPGA* behandelt. Dabei werden Kamerabilder von einem FPGA-Board aufgenommen, gespeichert und weiterverarbeitet. Die Aufgabe bestand darin, aufbauend auf bereits vorhandenen Funktionen, der Speicherung von Kamerabildern und deren Ausgabe über VGA auf einen Monitor, die Bilddatenverarbeitung im FPGA-Board weiterzuentwickeln.

Ziel war es, eine eindimensionale Positionserkennung von Objekten zu implementieren, welche die Erkennung von Punkten und Linien in einer Bildzeile umfasst. Des Weiteren sollte die Bildwiederholungsrate erhöht werden.

Zur Erkennung von Objekten in eindimensionalen Bildern wurden verschiedene Funktionsblöcke in der Hardwarebeschreibungssprache Very High Speed Integrated Circuit Hardware Description Language geschrieben. Mittels dieser Funktionsblöcke ist es möglich, Punkte und Linien als Objekte in einer Bildzeile zu erkennen und ihre Position zu ermitteln. Zudem wurde die Bildwiederholungsrate erhöht und die damit verbundenen auftretenden Probleme gelöst.

## Eigenständigkeitserklärung

Ich habe die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.<sup>1</sup>

---

(Ort, Datum)

---

(Unterschrift)

---

<sup>1</sup>Dies ist keine „Eidesstattliche“ Erklärung, da i.d.R. keine Vereidigung stattgefunden hat.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Tabellenverzeichnis</b>	<b>VII</b>
<b>Abkürzungsverzeichnis</b>	<b>VIII</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Digitale Bildverarbeitung</b>	<b>3</b>
2.1 Digitales Bild und digitale Bilddaten(vor)verarbeitung . . . . .	3
2.2 Beziehungen zwischen Pixeln . . . . .	4
2.3 Mathematische Operationen zur Bilddatenverarbeitung . . . . .	5
<b>3 Hardwareboard und Hardwarebeschreibung</b>	<b>9</b>
3.1 FPGA-Board . . . . .	9
3.2 Quartus Prime 18.1 Lite . . . . .	10
3.3 VHDL . . . . .	11
<b>4 Aufbau und genutzte Hardwarekomponenten</b>	<b>12</b>
4.1 Kamera STC-CMB33PCL . . . . .	12
4.2 FPGA-Board Altera DE2-115 . . . . .	15
<b>5 Aktueller Stand</b>	<b>18</b>
5.1 cameralink_tapping_base . . . . .	18
5.2 camera_data_mux_gen . . . . .	19
5.3 MemoryAccessController . . . . .	20
5.4 SDRAM_Write_Buffer_gen . . . . .	21
5.5 dram . . . . .	22
5.6 sram_controller_latency1 . . . . .	23
5.7 SDRAM_Read_Buffer_gen . . . . .	23
5.8 vga_controller . . . . .	25
<b>6 Eindimensionale Positionserkennung verschiedener Objekte</b>	<b>26</b>
6.1 Extraktion einer Bildzeile . . . . .	26
6.2 Eindimensionale Positionserkennung eines isolierten Punktes . . . . .	31
6.3 Eindimensionale Positionserkennung mehrerer isolierter Punkte . . . . .	33
6.4 Eindimensionale Positionserkennung einer Linie . . . . .	35
6.5 Eindimensionale Positionserkennung mehrerer Linien . . . . .	38
6.6 Funktionen von Tastern und Schaltern . . . . .	39
<b>7 Bildwiederholungsrate</b>	<b>42</b>
7.1 Maximale Bildwiederholungsrate . . . . .	42
7.2 Bilddopplung . . . . .	43
7.3 Belichtungszeit . . . . .	45

<b>8 Ergebnis und Diskussion</b>	<b>46</b>
8.1 Ausgabe einer Bildzeile . . . . .	46
8.2 Erkennung isolierter Punkte in einem eindimensionalen Bild . . . . .	46
8.3 Erkennung von Linienobjekten in einem eindimensionalen Bild . . . . .	47
8.4 Erhöhung der Bildwiederholungsrate . . . . .	48
<b>9 Zusammenfassung und Ausblick</b>	<b>49</b>
<b>A Literatur</b>	<b>51</b>

## Abbildungsverzeichnis

1	Aufbau eines digitalen Bildes . . . . .	3
2	Pixel $p f(x,y)$ . . . . .	4
3	Darstellung der $N_4(p)$ , $N_D(p)$ und $N_8(p)$ Nachbarn des Pixels $p$ . . . . .	4
4	Darstellung der 4-, 8- und m-Nachbarschaftsart . . . . .	5
5	Matrizen aus 1. und 2. Ableitung eines Bildpunktes $f(x)$ . . . . .	6
6	Durchführung einer Faltung . . . . .	7
7	Folge von Pixeln mit Ergebnis der 1. und 2. Ableitung . . . . .	7
8	Laplace-Matrix . . . . .	8
9	Aufbau eines Field Programmable Gate Arrays . . . . .	9
10	Aufbau eines konfigurierbaren Logikbausteins und eines Programmable Logic Device . . . . .	10
11	Aufbau mit Kamera STC-CMB33PLC und FPGA-Board DE2-115 . . . . .	12
12	Kamera STC-CMB33PCL und Anschlüsse . . . . .	12
13	Implementierungen und Aufbau von Camera Link . . . . .	13
14	Horizontales Timing mit dem Signal LVAL . . . . .	14
15	Reihenfolge der Übertragung der einzelnen Pixel für ein Bild . . . . .	14
16	Vertikales Timing mit dem Signal FVAL . . . . .	15
17	FPGA-Board Altera DE2-115 . . . . .	15
18	DE2-115 Blockdiagramm . . . . .	16
19	CLR-HSMC Board . . . . .	16
20	Ablauf der Bilddatenverarbeitung und Funktionsblöcke altll0 und seg7dec . . . . .	18
21	Funktionsblock cameralink_tapping_base mit Parameterbeschreibung . . . . .	19
22	Funktionsblock camera_data_mux_gen mit Parameterbeschreibung . . . . .	20
23	Funktionsblock MemoryAccessController mit Parameterbeschreibung . . . . .	21
24	Funktionsblock SDRAM_Write_Buffer_gen mit Parameterbeschreibung . . . . .	22
25	Funktionsblock dram . . . . .	23
26	Funktionsblock sram_controller_latency 1 . . . . .	23
27	Funktionsblock SDRAM_Read_Buffer_gen mit Parameterbeschreibung . . . . .	24
28	Funktionsblock vga_controller mit Parameterbeschreibung . . . . .	25
29	Funktionsblock SINGLE_LINE_READ_BUFFER mit Parameterbeschreibung . . . . .	26
30	Zustandsmaschinen des SINGLE_LINE_READ_BUFFER's . . . . .	27
31	Verhalten der Signale des vga_controllers . . . . .	28
32	Anzahl der Takte der Abschnitte der Signale des vga_controllers . . . . .	29
33	Ausgabe einer Zeile über VGA . . . . .	29
34	Erzeugung der Eingangssignale rd_line_next und wr_en des SINGLE_LINE_READ_BUFFERS . . . . .	30
35	Funktionsblock GREY_LEVEL_Generator . . . . .	30
36	Funktionsblock ISOLATED_PONIT_DETECTION mit Parameterbeschreibung . . . . .	31
37	Zustandsmaschine ISOLATED_PONIT_DETECTION . . . . .	31
38	Programmausschnitt zur Berechnung der Faltung . . . . .	32
39	Darstellung eines isolierten Punktes in einer Bildzeile und Ausgabe der x-Koordinate auf dem FPGA-Board . . . . .	32
40	Funktionsblock ISOLATED_POINT_GENERATOR mit Parameterbeschreibung . . . . .	33

41	Funktionsblock ISOLATED_PONIT_DETECTION_2 mit Parameterbeschreibung . . . . .	34
42	Zustandsmaschine ISOLATED_PONIT_DETECTION_2 . . . . .	34
43	Darstellung mehrerer isolierter Punkte in einer Bildzeile und Ausgabe deren x-Koordinate auf dem FPGA-Board . . . . .	35
44	Funktionsblock ISOLATED_POINT_GENERATOR_2 mit Parameterbeschreibung des . . . . .	35
45	Funktionsblock LINE_DETECTION mit Parameterbeschreibung . . . . .	36
46	Zustandsmaschine LINE_DETECTION . . . . .	37
47	Darstellung eines Linienobjekts in einer Bildzeile und die Ausgabe der x-Koordinate auf dem FPGA-Board . . . . .	37
48	Funktionsblock LINE_DETECTION_2 mit Parameterbeschreibung . . . . .	38
49	Zustandsmaschine LINE_DETECTION_2 . . . . .	39
50	Darstellung mehrerer Linienobjekte in einer Bildzeile und die Ausgabe der x-Koordinate auf dem FPGA-Board . . . . .	40
51	Teilausschnitt der Logik für die Ausgabe über die Siebensegmentanzeigen . . . . .	41
52	CLCtrol2_Vertical_ROI . . . . .	42
53	Bilddopplung bzw. Mehrfachausgabe eines Bildes durch Erhöhung der Bildwiederholungsrate . . . . .	44
54	Erhöhung der Bildwiederholungsrate auf 0,841 kHz und 1,995 kHz ohne Bilddopplung bzw. Mehrfachausgabe . . . . .	44

**Tabellenverzeichnis**

1	Funktionsbeschreibung der Taster des FPGA-Boards . . . . .	40
2	Beschreibung der Einstellmöglichkeiten über die Schalter des FPGA-Boards . . . . .	41
3	Erhöhung der Bildwiederholungsrate mit der Anzahl der Bildzeilen und der dazugehörigen Bildwiederholungsfrequenz . . . . .	43

## Abkürzungsverzeichnis

AIA	Automated Imaging Association
AS	Active Serial
ASCI	Application Specific Integrated Circuit
CBL	konfigurierbarer Logikblock
CLR-HSMC	Camera Link Receiver High Speed Mezzanine Card
EEPROM	Electrically Erasable Programmable Read Only Memory
FIFO	FirstInFirstOut
FPGA	Field Programmable Gate Array
FPS	Frames per Second
I/O	Input/Output
LUT	Look UP Table
LVDS	Low Voltage Differential Signaling
PLD	Programmable Logic Device
PLL	Phase Locked Loop
pof	Programmer Object File
RAM	Random Access Memory
RS	Recommended Standard
SDR	Shrunk Delta Ribbon
sof	SDRAM Object File
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language

## 1 Einleitung

Die digitale Bilddaten(vor)verarbeitung nutzt Bilder, die aus verschiedenen Bereichen des elektromagnetischen Spektrums stammen. Das genutzte elektromagnetische Spektrum reicht vom Gamma- und Röntgenstrahlenbereich über den Bereich des ultravioletten, sichtbaren und infraroten Lichtes bis in den Mikro- und Radiowellenbereich. Zudem gibt es neben Bildern im elektromagnetischen Spektrum auch akustische und computergenerierte Bilder sowie die Elektronenmikroskopie.

Anwendungsbereiche für digitale Bildverarbeitung lassen sich u.a. in der Industrie, Medizin, Astronomie, Geografie und der Wetterüberwachung finden. So werden in der Industrie Röntgenstrahlen genutzt, um z.B. Bilder von Platinen aufzunehmen, damit auf diese Weise Produktionsmängel entdeckt werden können, wie etwa fehlende Bauteile. Zudem wird dort der sichtbare Bereich genutzt, um Objekte bzw. Fehler in Objekten, wie Lufteinschlüsse, zu erkennen oder die Positionen von Objekten zu bestimmen. In der Medizin werden Gammastrahlen für die Emissionscomputertomographie, Röntgenstrahlen für Röntgenbilder, Radiowellen für die Magnetresonanztomographie und Ultraschallwellen für Ultraschallbilder genutzt [Gonzalez, R. C.; Woods, R. E., 2018].

Diese Arbeit beschäftigt sich vorrangig mit dem Anwendungsbereich der Positionserkennung von Objekten im Bereich des sichtbaren Lichts. Dabei werden Bilder mittels einer Kamera aufgenommen und in einem Field Programmable Gate Array - Board (FPGA-Board) verarbeitet. Dabei baut diese Arbeit auf bereits vorhandenen bildverarbeitenden Funktionen im FPGA-Board auf. 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. Diese zwischengespeicherten Daten konnten über den VGA-Ausgang auf einem angeschlossenen Monitor ausgegeben werden.

Auf Basis der zuvor bereits beschriebenen laufenden Funktionen werden neue Funktionen auf dem FPGA-Board implementiert. Die Implementierung dieser Funktionen zur Positionserkennung von Objekten wird schrittweise durchgeführt. Zuerst erfolgt die Positionserkennung eindimensional in einem Schwarzweißbild. Anschließend wird die eindimensionale Positionserkennung auf mehrere Objekte erweitert. Danach erfolgt die Erhöhung der Bildwiederholungsrate.

Darauf aufbauend kann eine Positionserkennung für ein zweidimensionales Objekt in einem Schwarzweißbild implementiert werden, d.h., dass im gesamten Bild oder in einem mehreren Zeilen großen Bildausschnitt ein Objekt gesucht. Dies kann anschließend auf die Erkennung der Position mehrerer Objekte ausgeweitet werden. Durch die Positions-erkennung eines oder mehrerer Objekte soll es später möglich sein, die Beschleunigung und Geschwindigkeit eines Objektes zu erfassen. Die Daten der Beschleunigung und Geschwindigkeit können dann für die Ansteuerungen der Bewegung dieser Objekte genutzt werden.

In Kapitel 2 wird darauf eingegangen, was ein digitales Bild und digitale Bildverarbeitung sind. Zudem werden Zusammenhänge zwischen benachbarten Pixeln erläutert und mathematische Bildoperationen zur Kanten- und Objekterkennung beschrieben.

Darauf folgend wird in Kapitel 3 auf die bildverarbeitende Komponente(FPGA-Board) sowie die Möglichkeit, die Hardwarebeschreibung durchzuführen.

Kapitel 4 beschreibt den Aufbau und die Funktion der genutzten Hardwarekomponenten

der Studienarbeit.

Danach wird in Kapitel 5 der Stand des Projekts, auf dem die Weiterentwicklung der Bilddatenverarbeitung während der Studienarbeit beruht, beschrieben.

Auf diesen Stand aufbauend, wird in Kapitel 6 das Extrahieren einer Bildzeile aus einem Bild beschrieben und die eindimensionale Positionserkennung in dieser Zeile erläutert.

In Kapitel 7 wird die Erhöhung der Bildwiederholungsrate und deren Auswirkung auf die eindimensionale Positionserkennung eingegangen.

Anschließend werden in Kapitel 8 die erzielten Ergebnisse dargestellt und bewertet.

Abschließend wird in Kapitel 9 die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

## 2 Digitale Bildverarbeitung

Die digitale Bilddaten(vor)verarbeitung umfasst die Aufnahme und Verarbeitung digitaler Bilder. Im Nachfolgenden wird erläutert, wie ein digitales Bild aufgebaut ist und wie die Grenzen zwischen den einzelnen Stufen der Bildverarbeitung verteilt sind. Zudem wird auf die Beziehungen zwischen Pixeln und mathematischen Anwendungsmethoden zur Verarbeitung digitaler Bilder eingegangen.

### 2.1 Digitales Bild und digitale Bilddaten(vor)verarbeitung

Die digitale Bilddaten(vor)verarbeitung ist eine Methode, digitale Bilder zu bearbeiten und aus diesen Informationen zu gewinnen. Dabei wird ein Bild als zweidimensionale Funktion  $f(x,y)$  beschrieben, bei der die x-Koordinate die Spalte und die y-Koordinate die Zeile eines Bildes beschreibt, wie es in Abbildung 1 dargestellt ist. Die Amplitude der Funktion  $f(x,y)$  ist die Intensität bzw. Graustufe des Bildes an diesen Koordinaten. Ein Bild wird als digitales Bild bezeichnet, wenn x, y und die Amplitude der Funktion einer finiten und diskreten Menge entstammen. Diese digitalen Bilder bestehen somit aus einer endlichen Anzahl an Elementen, die meist Pixel genannt werden [Gonzalez, R. C.; Woods, R. E., 2018].

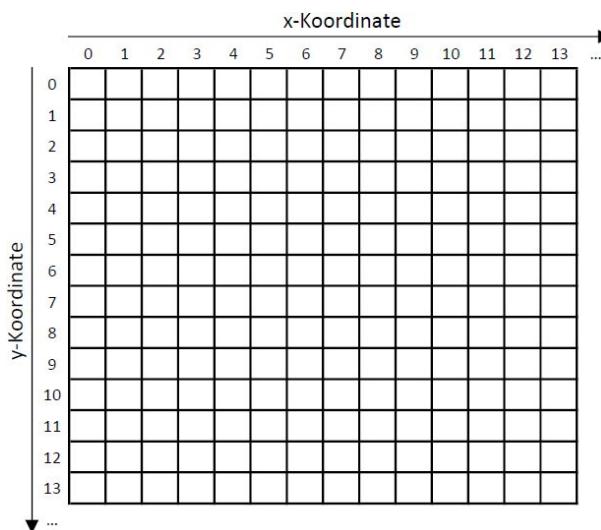


Abbildung 1: Aufbau eines digitalen Bildes

Gonzalez definiert die digitale Bildverarbeitung als Verarbeitung von Bildern, bei der der Input ein digitales Bild ist und der Output ein digitales Bild oder aus diesem Bild extrahierte Eigenschaften. Diese Eigenschaften dienen zur Wiedererkennung individueller Objekten. Die Schritte der digitalen Bilddatenverarbeitung sind der Erwerb eines Bildes, dessen Vorverarbeitung, das Extrahieren des Charakters, die Beschreibung des Charakters für die Verarbeitung in einem Computer und die Wiedererkennung des Charakters. Da es keine klar definierte Abgrenzung zwischen digitaler Bildverarbeitung und künstlichen/virtuellen Sehens (computer vision) gibt, unterteilt Gonzalez diese in drei Stufen des Verarbeitungsgrades, um eine Abgrenzung dieser zwei Bereiche zu finden. Diese Stufen sind der *low-level process*, der *mid-level process* und der *high-level process*. Im *low-level process* findet die Bilddatenvorverarbeitung statt, die u.a. die Reduktion von Störungen,

Kontrastverbesserung und Schärfung umfasst. Der *mid-level process* umfasst die Segmentation eines digitalen Bildes in Teilbilder und Objekte und die Aufbereitung dieser, um sie für die Verarbeitung mittels Computer nutzen zu können. Der *high-level process* beschäftigt sich mit dem *Sinngeben* der erkannten Objekte, das der Wahrnehmung und Verarbeitung des menschlichen Sehens sehr nahe kommt.

## 2.2 Beziehungen zwischen Pixeln

Auf Basis der Definition eines digitalen Bildes als Funktion  $f(x,y)$  wird in diesem Abschnitt auf die Beziehungen der Pixel eines digitalen Bildes zueinander eingegangen.

$f(x-1,y-1)$	$f(x,y-1)$	$f(x+1,y-1)$
$f(x-1,y)$	$f(x,y)$	$f(x+1,y)$
$f(x-1,y+1)$	$f(x,y+1)$	$f(x+1,y+1)$

Abbildung 2: Pixel p  $f(x,y)$

So hat ein Pixel p an der Stelle  $f(x,y)$  laut Gonzalez zwei vertikale ( $f(x+1,y)$ ,  $f(x-1,y)$ ), zwei horizontale ( $f(x,y+1)$ ,  $f(x,y-1)$ ) und 4 diagonale ( $f(x+1,y+1)$ ,  $f(x+1,y-1)$ ,  $f(x-1,y+1)$ ,  $f(x-1,y-1)$ ) Nachbarn. Diese benachbarten Pixel des Pixels p sind in Abbildung 2 dargestellt. Zudem werden die Nachbarpixel von p, wie in Abbildung 3 gezeigt, in drei verschiedene Gruppen von Nachbarn eingeteilt. Die zwei vertikalen und die zwei horizontalen sind die  $N_4(p)$  und die vier diagonalen sind die  $N_D(p)$  Nachbarpixel. Alle Pixelnachbarn zusammen werden als  $N_8(p)$  Nachbarpixel bezeichnet.

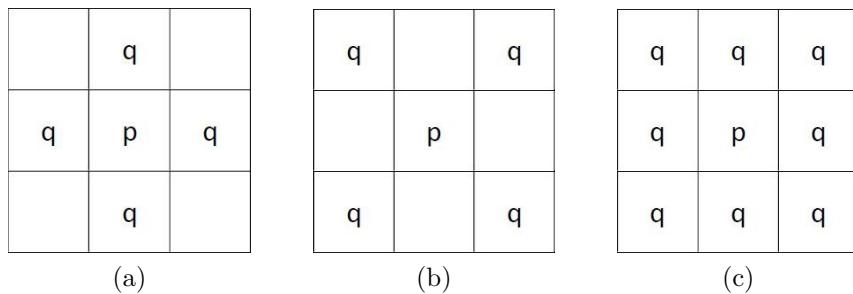


Abbildung 3: Darstellung der  $N_4(p)$  (a),  $N_D(p)$  (b) und  $N_8(p)$  (c) Nachbarn des Pixels p

Des Weiteren definiert Gonzalez drei verschiedene Nachbarschaftsarten, bei denen zwei Pixel p und q nur unter bestimmten Voraussetzungen als benachbart gelten. Dies ist zum einen die 4-Nachbarschaft, bei der die zwei Pixel p und q benachbart sind, wenn ihre Intensitäten aus einer gemeinsamen Menge V entspringen und q gleichzeitig ein  $N_4(p)$  Nachbar ist. Zum anderen gibt es die 8-Nachbarschaft, bei der die Pixel p und q benachbart sind, wenn ihre Intensitäten aus V stammen und q ein  $N_8(p)$  Nachbar ist. Die dritte Art der Nachbarschaft ist die m-Nachbarschaft. Hierbei sind die Pixel p und

$q$  benachbart, wenn  $p$  und  $q$  Intensitätswerte aus  $V$  haben und sie  $N_4(p)$  Nachbarn sind oder es handelt sich um  $N_D(p)$  Nachbarn, deren Pixel  $N_4(p) \wedge N_4(q)$  Intensitäten haben müssen, die nicht aus der Menge  $V$  stammen. Die Beispiele in Abbildung 4 a, b und c zeigen die verschiedenen Nachbarschaftsarten. Dabei wird davon ausgegangen, dass die Intensität der Pixel aus der Menge  $V = \{1\}$  stammen müssen, um benachbart zu sein. Ob zwei Pixel benachbart sind, wird durch die schwarze Linie zwischen den Pixeln gezeigt.

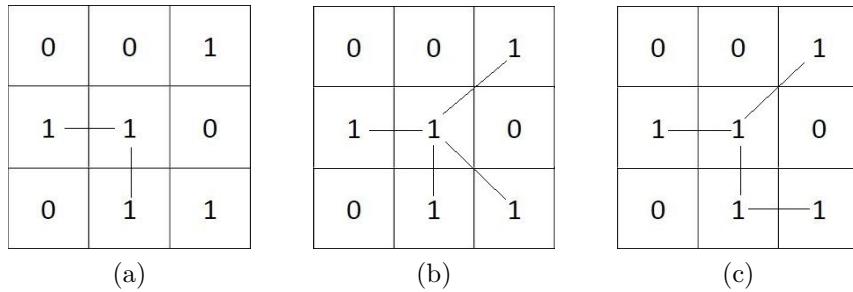


Abbildung 4: Darstellung der 4- (a), 8- (b) und m-Nachbarschaftsart (c); benachbarte Pixel sind durch einen schwarzen Strich miteinander verbunden

Neben den verschiedenen Nachbarschaftsarten kann ein digitales Bild zusätzlich in verschiedene Teilmengen unterteilt werden. Die beiden Mengen  $S$  und  $R$  sind Teilmengen von Pixeln aus einem digitalen Bild. Dabei sind zwei Pixel  $p$  und  $q$  in der Teilmenge  $S$  miteinander verbunden, wenn es einen digitalen Pfad zwischen diesen beiden Pixeln gibt, wobei die Pixel des Pfades auch aus der Teilmenge  $S$  stammen müssen. Ein digitaler Pfad ist laut Gonzalez ein Pfad zwischen einem Pixel  $p$  mit den Koordinaten  $(x_0, y_0)$  und einem Pixel  $q$  mit den Koordinaten  $(x_n, y_n)$ , wobei die einzelnen Pixel des Pfades eine 4-, 8- oder m-Nachbarschaft aufweisen müssen, je nachdem welche Nachbarschaftsart für den digitalen Pfad gewählt wurde. Zudem ist ein digitaler Pfad ein geschlossener Pfad, wenn der Endpunkt dem Ausgangspunkt entspricht. Die Teilmenge  $R$ , die auch Region eines Bildes genannt wird, ist eine verbundene Menge von Pixeln mit Intensitäten aus der Menge  $V$ . Zwei Regionen  $R_i$  und  $R_j$  sind je nach gewählter Nachbarschaftsart benachbart, wenn mindestens ein Pixel aus  $R_i$  mit einem Pixel aus  $R_j$  benachbart ist. Die innere Grenze einer Region bilden die Pixel von  $R$ , die neben Pixeln liegen, die nicht zur Region  $R$  gehören. Die äußere Grenze bilden die Pixel, die die Region  $R$  umschließen.

## 2.3 Mathematische Operationen zur Bilddatenverarbeitung

Mithilfe von mathematischen Operationen können aus den Bilddaten Informationen über das Bild erlangt werden. Nach Gonzalez ist es möglich, anhand des Ergebnisses der 1. (Gleichung 1) und 2. (Gleichung 2) Ableitung der eindimensionalen Funktion  $f(x)$ , die ein eindimensionales Bild beschreibt, Unterschiede zwischen benachbarten Pixeln zu erkennen. Dies sind zum Beispiel sprunghafte oder rampenförmige Intensitätswechsel, mit deren Hilfe Objekte erkannt werden können.

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \quad (1)$$

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1) - 2f(x) + f(x-1) \quad (2)$$

Aus der 1. und 2. Ableitung kann jeweils eine Gewichtungsmatrix (Abbildung 5 a und b) abgeleitet werden. Mithilfe dieser Gewichtungsmatrix kann eine Faltung mit den Pixelwerten einer Zeile durchgeführt werden. Anhand der Ergebnisse der Faltung für jedes Pixel einer Zeile kann darauf geschlossen werden, wie sich ein Pixel in seiner Intensität zu seinen Nachbarpixeln verhält.

(a)	(b)	(c)	(d)

Abbildung 5: Matrizen aus 1. (a, b) und 2. Ableitung (c, d) eines Bildpunktes  $f(x)$  zur Gewichtung der einzelnen Faltungselementen

Die oben erwähnte mathematische Operation der Faltung dient zur Gewichtung der einzelnen Elemente einer Funktion und ist beispielhaft in Abbildung 6 gezeigt. Dort wird eine Folge von Pixelwerten mit der Gewichtungsmatrix der 2. Ableitung aus Abbildung 5 d gefaltet. Dabei wird zuerst eine der beiden Funktionen in der Mitte gespiegelt; dies ist hier mit der Gewichtungsmatrix an der Stelle  $-2$  geschehen. Da sich die Werte  $1$ ,  $-2$  und  $1$  allerdings symmetrisch um verteilen, entsteht eine Gewichtungsmatrix, die genauso aussieht wie die Ausgangsmatrix. Anschließend wird das Ergebnis der Faltung für die Startposition berechnet, indem die Elemente der Gewichtungsmatrix mit überschneidenden Pixelwerten multipliziert und aufsummiert werden. Das Ergebnis dieser Berechnung beträgt  $0$  und wird in der Zeile *Ergebnis* an die Stelle des mittleren Elementes der Gewichtungsmatrix geschrieben. Anschließend wird die Gewichtungsmatrix ein Feld weiter geschoben (1. Verschiebung). Dort werden die Elemente der Gewichtungsmatrix erneut mit den überschneidenden Pixelwerten multipliziert und aufsummiert. Das Ergebnis wird wieder an die Stelle des mittleren Elementes der Gewichtungsmatrix in die Zeile *Ergebnis* geschrieben. Dieser Vorgang wird solange wiederholt, bis die in der Abbildung 6 beschriebene Endposition erreicht ist. In der Zeile *Ergebnis* steht nun das Ergebnis der Faltung aller Pixelwerte mit der Gewichtungsmatrix.

Wie bereits erläutert, kann das Ergebnis der Faltung aus einer Gewichtungsmatrix und den Pixelwerten einer Zeile auf Intensitätsunterschiede zwischen benachbarten Pixeln hinweisen. Dies ist beispielhaft in Abbildung 7 dargestellt. In dieser Abbildung ist das Ergebnis der Faltung von Pixelwerten einer Zeile, die einen Wert zwischen  $0$  und  $10$  annehmen können, mit der Gewichtungsmatrix der 1. und der 2. Ableitung dargestellt. Diese Zeile weist von der  $x$ -Koordinate  $f(x)$  bis zur  $x$ -Koordinate  $f(x+2)$  als erstes eine Folge von Pixeln mit dem gleichen Intensitätswert, der  $10$  beträgt, auf. Danach folgt eine rampenförmige Abnahme der Intensität, bis diese den Wert  $0$  an der Stelle  $f(x+7)$  erreicht und die nächsten zwei Pixel den Wert  $0$  haben. Danach erfolgt von  $f(x+9)$  zu  $f(x+10)$  ein Sprung mit einem Intensitätswechsel von  $0$  auf  $10$ . Anschließend haben die Pixel den Intensitätswert  $10$ , außer an der Stelle  $f(x+13)$ . Dort ist ein isolierter Punkt zu sehen.

Ein Punkt ist ein isolierter Punkt, wenn er sich in seiner Intensität stark von seinen Nachbarpixeln unterscheidet.

Pixelintensität	0	0	0	1	3	3	1	0	0	0
Startposition	1	-2	1							
1. Verschieben		1	-2	1						
2. Verschieben			1	-2	1					
:										
Endposition								1	-2	1
Ergebnis	-	0	1	-1	-2	-2	-1	1	0	-

Abbildung 6: Beispiel der Durchführung einer Faltung mit der Gewichtungsmatrix der 2. Ableitung

x-Koordinate	f(x)	f(x+1)	f(x+2)	f(x+3)	f(x+4)	f(x+5)	f(x+6)	f(x+7)	f(x+8)	f(x+9)	f(x+10)	f(x+11)	f(x+12)	f(x+13)	f(x+14)	f(x+15)	f(x+16)
Pixelintensität	10	10	10	8	6	4	2	0	0	0	10	10	10	0	10	10	10
1. Ableitung	0	0	-2	-2	-2	-2	-2	-2	0	10	0	0	-10	10	0	0	0
2. Ableitung	-	0	-2	0	0	0	0	2	0	10	-10	0	-10	20	-10	0	-

Abbildung 7: Folge von Pixelwerten und Ergebnisse der 1. und 2. Ableitung für alle x-Koordinaten

Die beschriebenen Formen des Intensitätswechsels (Folgen gleicher Intensität, Rampen, Sprünge und isolierte Punkte) lassen sich anhand der Ergebnisse der Faltung mit der 1. und 2. Ableitung bestimmen. So ergibt das Ergebnis der 1. Ableitung 0, wenn sich ein Pixel in seiner Intensität nicht von seinem nachfolgenden Pixel unterscheidet, wie es bei den Pixeln an den x-Koordinaten  $f(x)$  und  $f(x+1)$  zu sehen ist. Unterscheiden sich die Intensitätswerte zwei aufeinander folgender Pixel, so ist das Ergebnis ungleich 0. Eine Folge von Nullen als Ergebnis der 1. Ableitung weist auf eine Folge von Pixeln mit gleicher Intensität hin ( $f(x)$  bis  $f(x+1)$ ). Der Beginn einer Rampe ist durch ein Ergebnis ungleich 0 erkennbar und entlang einer Rampe ist das Ergebnis ebenfalls ungleich 0, wie es an Werten der 1. Ableitung von der Stelle  $f(x+2)$  bis  $f(x+7)$  zu erkennen ist. Ein Sprung in der Intensität ist daran auszumachen, dass die 1. Ableitung an der Stelle vor dem Sprung einen großen Wert aufweist, wie das Ergebnis für die Pixel  $f(x+9)$  und  $f(x+10)$  zeigt, an denen ein Intensitätswechsel von 0 auf 10 stattfindet. Danach folgt an der x-Koordinate  $f(x+13)$  ein isolierter Punkt, der sich durch einen betragsmäßig großen Wert als Ergebnis der 1. Ableitung am vorherigen Pixel  $f(x+12)$  und einem großen Wert mit invertiertem Vorzeichen an seiner Koordinate bemerkbar macht.

Für die 2. Ableitung kann an der Stelle  $f(x)$  kein Ergebnis für die Faltung berechnet werden, da drei Pixelwerte für die Berechnung benötigt werden. Dies gilt ebenfalls für das letzte Pixel  $f(x+16)$  der Zeile. Eine Folge von Pixeln mit gleicher Intensität hat das Ergebnis 0, wie es an der Stelle  $f(x+1)$  zu sehen ist. Ein rampenförmiger Intensitätswechsel kündigt sich mit einem Ergebnis für die 2. Ableitung an, welches

ungleich 0 ist, folgend von Ergebnissen gleich 0 entlang der Rampe und einen Ergebnis ungleich 0 am Ende der Rampe, welches aber ein invertiertes Vorzeichen im Vergleich zum Ergebnis des Beginns der Rampe aufweist. Dies ist von der Stelle  $f(x+2)$  bis  $f(x+7)$  mit dem Wert -2 am Beginn und dem Wert 2 am Ende der Rampe zu erkennen. Ein Sprung in der Intensität ist dadurch zu erkennen, dass die Faltung mit der 2. Ableitung ein Ergebnis ungleich 0 an der Stelle vor dem Sprung und ein Ergebnis ungleich 0 mit invertiertem Vorzeichen an der Stelle des Sprungs vorweist. Dies ist an den Ergebnissen für die Stellen  $f(x+9)$  und  $f(x+10)$  zu erkennen. Dort findet in der Pixelintensität ein Sprung von 0 auf 10 statt und das Ergebnis der 2. Ableitung beträgt an diesen Stellen 10 und -10. Der sich an der x-Koordinate  $f(x+13)$  befindliche isolierte Punkt ist daran zu erkennen, dass er an dieser Stelle einen großen Wert ungleich 0 vorzuweisen hat und dass seine Nachbarpixel Werte ungleich 0 mit invertiertem Vorzeichen besitzen.

Je größer der Betrag des Ergebnisses der 1. oder 2. Ableitung ist, desto größer sind auch die Intensitätsunterschiede bei Rampen, Sprüngen oder isolierten Punkten. Mittels der Berechnung der Faltung der Gewichtungsmatrix von der 1. oder 2. Ableitung mit den Pixelwerten und eines Auswertungsalgorithmus ist es möglich, Objekte zu erkennen und deren Position in einer Zeile zu bestimmen.

Mittels der Laplace-Operation (Gleichung 3 und Gleichung 4), die einer 2. Ableitung einer zweidimensionalen Funktion  $f(x,y)$  entspricht, können laut Gonzalez Unterschiede zwischen benachbarten Pixeln in einem zweidimensionalen Bild erkannt werden. Aus der Laplace-Operation ist eine Matrix (Abbildung 8 a) ableitbar, die zur Faltung mit einer entsprechenden Matrix aus Pixelwerten genutzt werden kann. Anhand eines Auswertungsalgorithmus können aus dem Faltungsergebnis für jedes Pixel eines Bildes Kanten und Objekte erkannt werden.

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \quad (3)$$

$$\nabla^2 f(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y) \quad (4)$$

0	1	0
1	-4	1
0	1	0

(a)

1	1	1
1	1	1
1	1	1

(b)

Abbildung 8: Laplace-Matrix (a) und Pixelwerte für  $f(x,y)$  und dessen Nachbarpixel (b)

Wird beispielsweise die Laplace-Matrix aus Abbildung 8 (a) mit den Pixelwerten aus Abbildung 8 (b) verrechnet, lautet das Ergebnis der Berechnung 0. Dieses Ergebnis weist darauf hin, dass sich das mittlere Pixel in seiner Intensität nicht von seinen Nachbarpixel, die in der Berechnung berücksichtigt werden, unterscheidet.

### 3 Hardwareboard und Hardwarebeschreibung

Die in Kapitel 2 beschriebene digitale Bilddatenverarbeitung findet in dieser Studienarbeit in einem FPGA-Board statt. In diesem Kapitel wird erläutert, was ein FPGA-Board und dessen wichtigste Komponente, das FPGA, ist. Des Weiteren wird auf die Software *Quartus Prime 18.1 Lite* eingegangen, mit der das FPGA-Board programmiert wird. Zudem wird die Hardwarebeschreibungssprache VHDL erläutert, mit der Funktionsblöcke im Programm erstellt werden können.

#### 3.1 FPGA-Board

Ein Field Programmable Gate Array (FPGA)-Board, ist ein Hardware-Board, das als Basis ein FPGA besitzt, in dem die Verarbeitung der Daten stattfindet. Zudem weist es eine Vielzahl an Anschlüssen auf, um mit Peripheriegeräten kommunizieren zu können, und es besitzt verschiedene Speicher zur Daten- und Programmspeicherung. Des Weiteren kann es Taster und Schalter zur Bedienung aufweisen sowie Anzeigeelemente, wie z.B. Siebensegmentanzeigen.

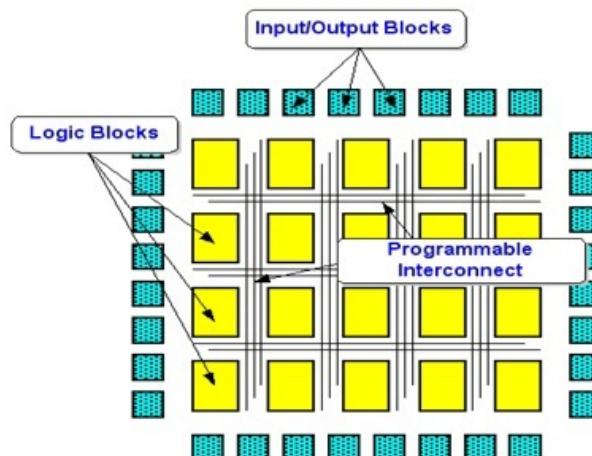


Abbildung 9: Aufbau eines Field Programmable Gate Arrays (FPGA) [microcontrollerslab.com, o.J.]

Ein FPGA, das in Abbildung 9 dargestellt ist, ist ein programmierbarer digitaler Baustein, der aus konfigurierbaren logischen Blöcken (CLBs) besteht, die in einer Matrix angeordnet sind und über programmierbare Verbindungsanäle miteinander verbunden werden können. Jeder der in Abbildung 10 a gezeigten konfigurierbaren Logikblöcke (CLBs) eines FPGAs basiert in seinem Aufbau auf einem Programmable Logic Device (PLD), das aus einer verknüpften UND-Matrix und ODER-Matrix besteht, wie es in Abbildung 10 b gezeigt ist. Durch das Aufstellen von Wahrheitstafeln, die in Look Up Tables (LUT) gespeichert werden, wird die logische Funktion eines PLDs beschrieben. Anhand dieser Wahrheitstafel kann nun eine Programmierabelle erstellt werden, mit deren Hilfe die UND- und ODER-Matrix programmiert wird. Die Verbindung der Eingänge und der Ausgänge eines PLDs an ein anderes erfolgt über die Programmierung der Verbindungsanäle, wobei der Ausgang eines PLDs entweder direkt angebunden oder zusätzlich über ein Flip-Flop geführt werden kann. Die Vielzahl an CLBs in einem

FPGA ermöglicht es, große und komplexe Funktionen in dieses zu implementieren [Tietze, U. et al., 2016].

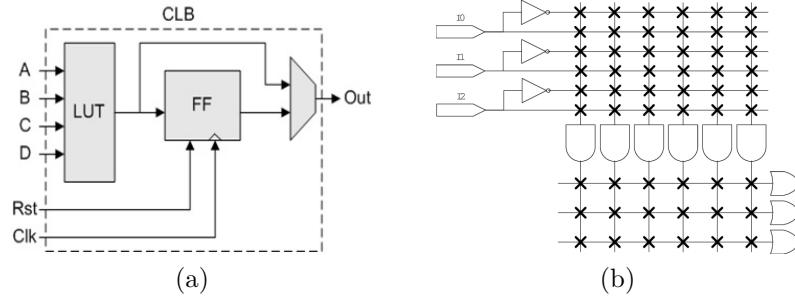


Abbildung 10: Aufbau eines konfigurierbaren Logikbausteins (CLB) [microcontrollerslab.com, o.J.] (a) und Aufbau eines Programmable Logic Device (PLD) aus programmierbarer UND- und ODER-Matrix [Upegui, 2006] (b)

Auf einem FPGA-Board befinden sich neben dem FPGA auch noch weitere Bauteile wie i.d.R ein Random Acces Memory (RAM) zur Datenspeicherung und ein Flash-Speicher zur Programmspeicherung. Zudem hat es Input/Output (I/O)-Anschlüsse, um Daten mit Peripheriegeräten auszutauschen bzw. von diesen Daten zu empfangen, zu verarbeiten und wieder an ein Peripheriegerät auszugeben. Die I/O-Anschlüsse können z.B. ein Video Graphics Array (VGA)-Anschluss, um Bilder an einem Monitor darzustellen, ein USB-Anschluss, ein RS232-Anschluss oder eine Ethernetverbindung sein [Tietze, U. et al., 2016].

Die Programmierung der Funktion eines FPGAs erfolgt über die Beschreibung des Schaltungsverhaltens der Hardware mittels einer Software, wie z.B. Quartus Prime 18.1 Lite (Kapitel 3.2). Dieses Schaltungsverhalten wird mittels Logikplänen und Hardwarebeschreibungssprachen, wie z.B. VHDL (Kapitel 3.3), beschrieben. Über eine Schaltpl eingabe im Logikplan können Verbindungen zwischen einzelnen Teilfunktionen, die mit VHDL beschrieben wurden, und schon in Bibliotheken hinterlegten Funktionen, wie Addierer, Multiplexer und Demultiplexer, gezogen werden. Sind alle Eingaben getätig, werden alle Verbindungen und VHDL-Beschreibungen in logische Funktionen überführt, auf ihre Syntax überprüft und gegebenenfalls von einem Simplifier vereinfacht. Danach wird das Programm auf das FPGA übertragen, wobei durch einen *Device-Fitter* noch Anpassungen an die spezifische Architektur des FPGAs durchgeführt werden, damit die verfügbaren Ressourcen optimal genutzt werden [Tietze, U. et al., 2016].

### 3.2 Quartus Prime 18.1 Lite

Die Software *Quartus Prime 18.1 Lite* ist eine FPGA-Design Software der Firma Intel Corporation. Die Hardwarebeschreibung eines FPGAs kann mittels einer Schematic, dies ist ein Schalt- bzw. Logikplan, und den Hardwarebeschreibungssprachen VHDL (Kapitel 3.3) und Verilog erfolgen.

Mit dieser Software ist es möglich, Projekte zu erstellen, um Hardwarebeschreibungen für FPGAs durchzuführen. Nach der Erstellung des Projektes wird in der Schematic die Schaltung bzw. der Logikplan der Hardware erstellt. Dazu werden die benötigten Ein- und

Ausgänge, logischen Blöcke, arithmetischen Blöcke und selbsterstellte Funktionsblöcke, die mithilfe von VHDL oder Verilog geschrieben werden können, platziert und verbunden. Mittels des integrierten Compilers wird das Hardwaredesign synthetisiert, platziert, gerootet und anschließend eine Geräteprogrammierungsdatei, ein SRAM Object File (.sof), erstellt. Mithilfe des Programmers wird die erstellte Geräteprogrammierungsdatei genutzt, um das FPGA über den JTAG-Modus zu konfigurieren und die erstellte Funktion auf das Board zu übertragen. Da dies lediglich eine flüchtige Programmierung des FPGAs ist, die bei Verlust der Spannungsversorgung verloren geht, kann über einen Programmer Object File (.pof) eine dauerhafte Speicherung erfolgen. Die .pof-Datei kann über eine Konvertierung der .sof-Datei erstellt werden und mittels des Programmers im Active Serial (AS) Programming Mode auf ein serial configuration device mit Flashspeicher im FPGA-Board geladen werden. Aus diesem wird bei erneuter Spannungsversorgung das Programm in das FPGA geladen [Intel, 2019].

### 3.3 VHDL

Die *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) ist eine Hardwarebeschreibungssprache, mit der digitale, logische Schaltungen entworfen werden. Mittels Konstrukten und Befehlen werden Abläufe und das zeitliche Verhalten von Schaltungen in einem FPGA oder Application Specific Integrated Circuit (ASIC) beschrieben. Diese Konstrukte umfassen Logikkonstrukte wie z.B. UND-, Oder- und NAND-Schaltungen, Bedingungs- und Schleifenkonstrukte wie z.B. if-Anweisungen oder for-Schleifen sowie mathematische Formeln. Die Befehle dienen dazu Funktionen und Teilschaltungen in Module zusammenzufassen, wodurch Hierarchien aufgebaut werden können [Mikrocontroller.net, o.J.].

Eine Hardwarebeschreibung mittels VHDL basiert auf einer bestimmten Syntaxnotation. Zuerst wird eine *entity* erstellt, die die Schnittstellenbeschreibung des VHDL-Funktionsblocks nach außen darstellt. Dort werden in der *generic*-Umgebung Parameter deklariert, wenn diese benötigt werden. Im Abschnitt *port* werden die Ein- und Ausgangssignale des Funktionsblocks beschrieben, die die Kommunikation nach außen, also die Schnittstellenbeschreibung, festlegen. Nach der *entity* erfolgt das Erstellen der *architecture*, die die Funktionalität des VHDL-Funktionsblocks beschreibt. Dies umfasst die Deklaration von lokalen Signalen, die nur in dieser *architecture* benötigt werden, um zwischen Funktionselementen und Blöcken in der *architecture* Informationen auszutauschen sowie die VHDL-Anweisungen, mit denen das Verhalten des Funktionsblocks beschrieben wird. Diese VHDL-Anweisungen umfassen über die im obigen Abschnitt beschriebenen Konstrukte und Befehle hinaus viele weitere [Reichard, J.; Schwarz, B., 2015].

## 4 Aufbau und genutzte Hardwarekomponenten

Der Aufbau mit der verwendeten Hardware ist in Abbildung 11 dargestellt. Dieser umfasst die Kameras *STC-CMB33PCL* der Firma Semtech und das FPGA-Board *DE2-115* der Firma Terasic mit der Erweiterungskarte *CLR-HSMC Daughter Card* (Camera Link Receiver High Speed Mezzanine Card). Die Kamera ist über die Kameraschnittstelle Camera Link mit der CLR-HSMC Daughter Card verbunden, die über den HSMC-Anschluss des DE2-115-Boards angeschlossen ist.

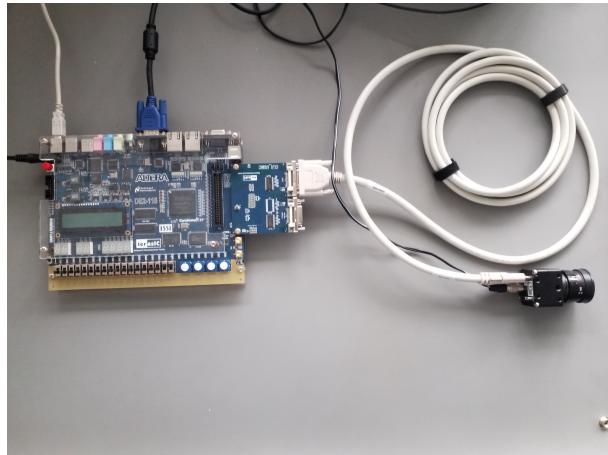


Abbildung 11: Aufbau mit Kamera STC-CMB33PCL und FPGA-Board DE2-115

### 4.1 Kamera STC-CMB33PCL

In Abbildung 12 ist die verwendete Kamera STC-CMB33PCL dargestellt, welche maximal 494 frames per second (FPS) aufnehmen kann. Die aufgenommenen Bilder sind monochrome Bilder, die in Graustufen dargestellt werden. Die Kamera besitzt drei Anschlüsse, einen Base Camera Link Connector, einen Medium Camera Link Connector und einen Power I/O Connector.

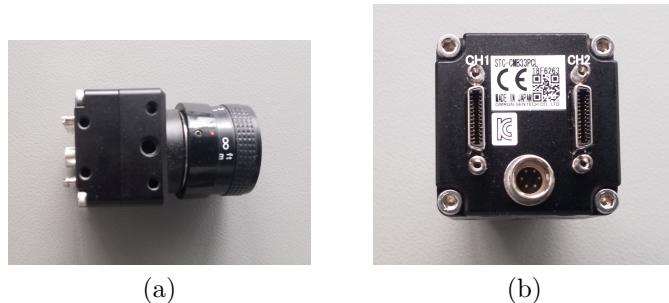


Abbildung 12: Kamera STC-CMB33PCL(a) und Anschlüsse(b)

Über den Power I/O Connector wird die Kamera mit Spannung versorgt und über den Base Camera Link Connector (Channel 1) ist die Kamera mit der Erweiterungskarte verbunden. Diese Verbindung erfolgt über ein Camera Link Kabel, das über einen 26-Pin

Shrunk Delta Ribbon (SDR)-Anschluss an die Kamera und einen Recommended Standard (RS)-232 Anschluss an die Erweiterungskarte angeschlossen ist.

Die verwendete Schnittstelle Camera Link ist ein Standard der Automated Imaging Association (AIA), der in der industriellen Bildverarbeitung genutzt wird und über die die Daten mehrerer Pixel gleichzeitig übertragen werden können. Sie weist die Implementierungen Base, Medium und Full auf, die verschiedene Datenübertragungsraten ermöglichen, siehe Abbildung 13 a [Stemmer, o.J.]. So können beispielsweise bei Base 3 Pixel mit je 8 Bit oder 2 Pixel mit 10 oder 12 Bit übertragen werden. Die zu übertragenden Daten werden über einen Multiplexer geführt und dann mittels Low Voltage Differential Signaling (LVDS)-Aderpaaren auf einen Demultiplexer geführt, der die Daten wieder demultiplext und für die weitere Verarbeitung zur Verfügung stellt. Dieser Aufbau ist beispielhaft an der Base Camera Link Konfiguration in Abbildung 13 b dargestellt [Wikipedia, 2018].

CameraLink-Zusammenfassung			
Konfiguration	Anzahl der Chips	Anzahl der Stecker	Datenübertragung
<b>Base</b>	1	1	3 x 8 Bit, 2 x 10/12 Bit 1 x 14/16 Bit, 1 x 24 Bit (RGB)
<b>Medium</b>	2	2	4 x 8/10/12 Bit 30/36 Bit (RGB)
<b>Full</b>	3	2	8 x 8 Bit (und mehr)
<b>Full (DECA-Modus)</b>	3	2	10 x 8 Bit

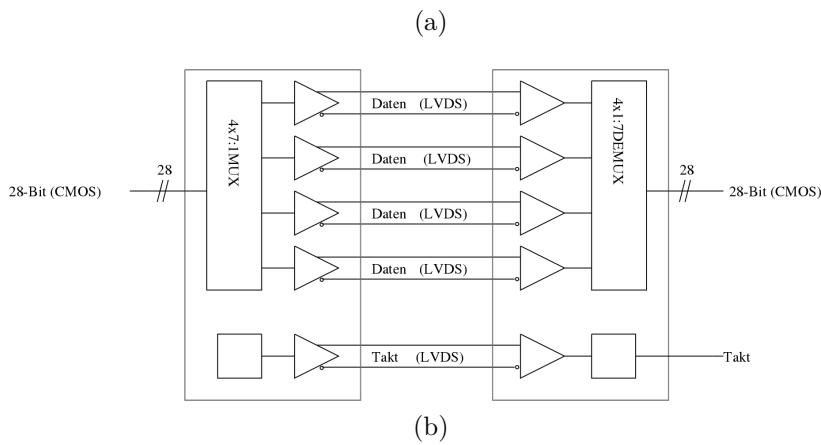


Abbildung 13: Implementierungen (a) [Stemmer, o.J.] und Aufbau (b) [Wikipedia, 2018] der Camera Link Schnittstelle

Die Einstellung der Kamera kann unterschiedlich mittels der Software *CLCctrl2* parametriert werden. So kann über verschiedene Parameter eingestellt werden, wie viele Zeilen und Spalten ein Bild hat, ob mehrere kleine Teilbilder aus dem Kamerabild übertragen werden sollen und ob die Daten für 2 oder 3 Pixel mit je 8, 10 oder 12 Bit ausgegeben werden. Die Parameter werden so eingestellt, dass ein Kamerabild 484 Zeilen und 642 Spalten hat und dass je 2 Pixel, die je 8 Bit lang sind, pro Takt ausgegeben werden, entsprechend der Übertragung der Daten über den Base Camera Link Connector der Kamera. Zudem werden der Zustand der Signale LVAL,

FVAL, DVAL und die Pixelfrequenz, mit der die Ausgabe der Pixel stattfindet, übertragen. Diese hat bei den eingestellten Parametern eine Frequenz von 84 MHz [Sentech, o.J.].

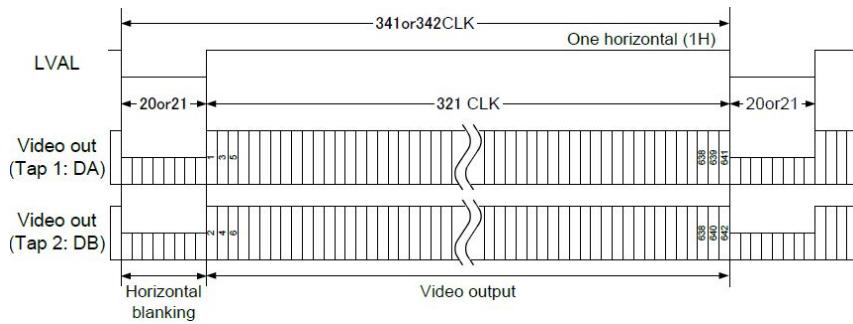


Abbildung 14: Horizontales Timing mit dem Signal LVAL [Sentech, o.J.]

Die Abbildung 14 zeigt die Funktion des Signals LVAL. Dieses signalisiert, ob die Kamera gerade Daten für die Pixel einer Zeile ausgibt oder nicht. Das Signal ist für 321 Takte (1 Takt = 11,9 ns) auf high gesetzt. In dieser Zeit werden die Daten für die Pixel einer Zeile ausgegeben, wobei pro Takt Daten für 2 Pixel, Tap1 und Tap2, ausgegeben werden. Danach ist das Signal für 20 oder 21 Takte auf low gesetzt. Durch den Signalwechsel von low auf high wird signalisiert, dass die Ausgabe einer neuen Zeile beginnt. In der Abbildung 15 ist zudem dargestellt, in welcher Reihenfolge die Pixel eines Bildes ausgegeben werden. Es werden immer die Pixel einer Zeile von links nach rechts und die Zeilen von oben nach unten ausgegeben. Das Signal DVAL besitzt die gleiche Funktion wie das Signal LVAL [Sentech, o.J.].

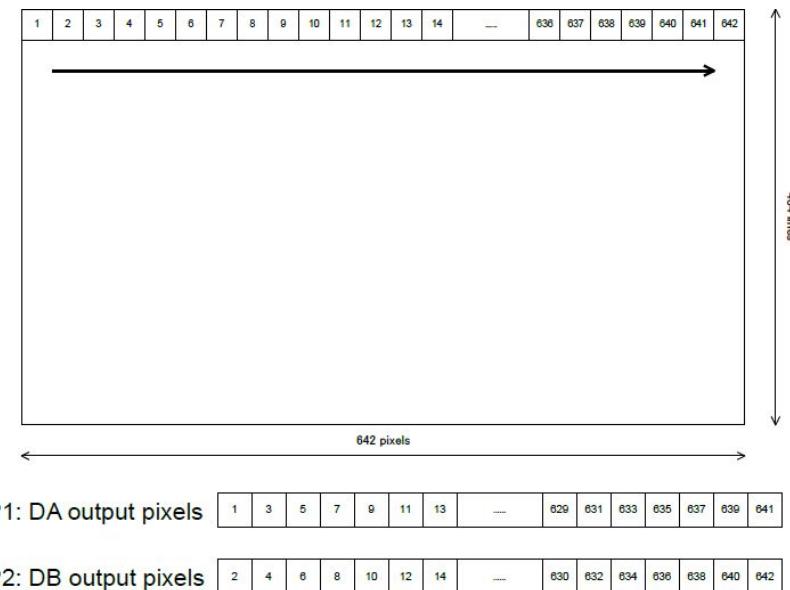


Abbildung 15: Reihenfolge der Übertragung der einzelnen Pixel für ein Bild [Sentech, o.J.]

Die Funktion des Signals FVAL ist in Abbildung 16 dargestellt. Das Signal ist high, solange die 484 Zeilen eines Bildes ausgegeben werden. Nach der Ausgabe der 484 Zeilen

wechselt das Signal von high auf low. In diesem Zustand verbleibt es für 14 Takte und durch den Wechsel von low auf high wird signalisiert, dass die Ausgabe der Pixeldaten eines neuen Bildes beginnt [Sentech, o.J.].

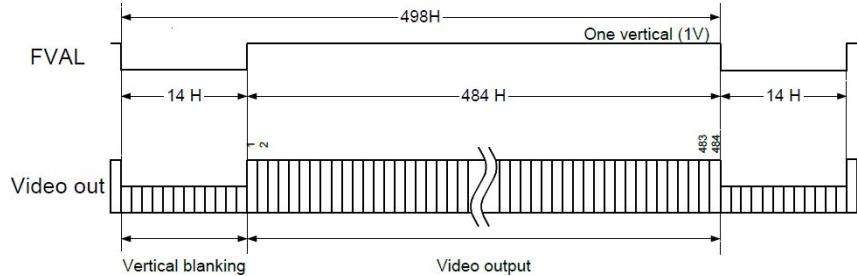


Abbildung 16: Vertikales Timing mit dem Signal FVAL (Quelle: [Sentech, o.J.])

Die Kamera STC-CMB33PCL der Firma Semtech stand ebenfalls zur Verfügung, wurde aber nicht genutzt, da diese Kamera Farbbilder liefert und daher nicht benötigt wurde, weil sich die Studienarbeit zuerst mit der Verarbeitung von Schwarzweißbildern beschäftigt.

## 4.2 FPGA-Board Altera DE2-115

Die Verarbeitung der Kameradaten findet in dem in Abbildung 17 dargestellten FPGA-Board DE2-115 der Firma Terasic statt. Als FPGA zur Datenverarbeitung dient ein *Altera Cyclone IV EP4CE115* mit 114.480 Logikeinheiten, 432 M9K-Speicherblöcken, 3.888 Kbit eingebettetem Speicher und 4 Phase Locked Loop (PLL).

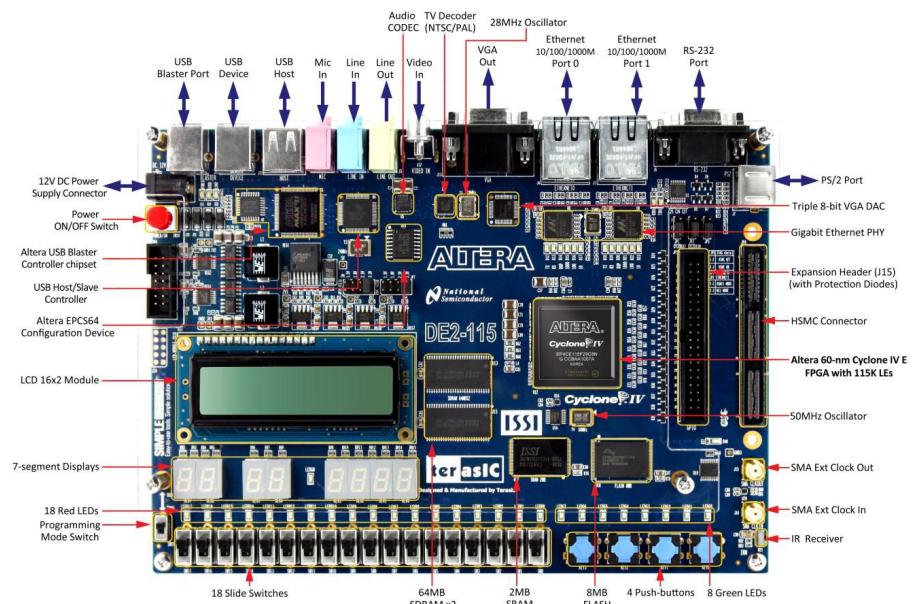


Abbildung 17: DE2-115 (Quelle: [Terasic, 2012])

Das Board hat diverse Anschlüsse wie z.B. einen USB-Anschluss, einen VGA-Anschluss und einen RS232-Anschluss, um Daten an Peripheriegeräte ausgeben oder von diesen erhalten zu können. Zudem hat es einen 2MB SRAM, zwei 64MB SDRAMs und einen 8MB Flash Speicher. Des Weiteren hat das DE2-115 Board 18 Schalter und 4 low-aktive Taster und als Anzeigeelemente besitzt es ein LCD 16x2 Modul, acht Siebensegment-Anzeigen, 18 rote LEDs und neun grüne LEDs. Zur Generierung von Taktsignalen fungieren drei 50MHz Oszillatoren. Das Blockdiagramm des DE2-115 ist in Abbildung 18 dargestellt, welches die internen Verbindungen der einzelnen Komponenten des DE2-115 zeigt [Terasic, 2012].

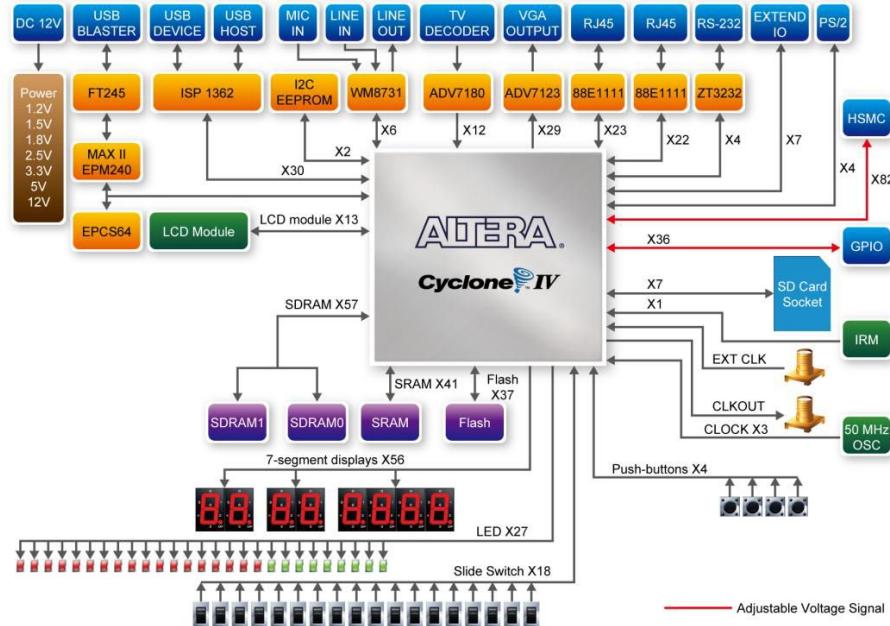


Abbildung 18: DE2-115 Blockdiagramm (Quelle: [Terasic, 2012])

Über den HSMC-Anschluss ist die in Abbildung 19 dargestellte Erweiterungskarte CLR-HSMC Daughter Card an das FPGA-Board angeschlossen. Diese Erweiterungskarte wird benötigt, um die Bilddaten über die Camera Link Schnittstelle der Kamera auf das FPGA-Board zu übertragen, da das FPGA-Board keine eigene Camera Link Schnittstelle hat.

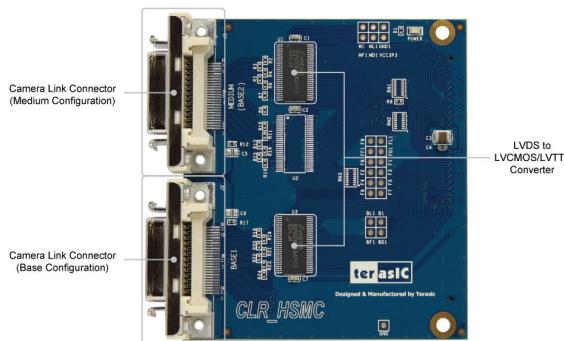


Abbildung 19: CLR-HSMC Board (Quelle: [Terasic, 2012])

Die Konfigurierung des FPGAs und damit die Verarbeitungsweise der Bilddaten erfolgt über den USB-Blaster-Anschluss, indem die mit der Software *Quartus Prime 18.1 Lite*

erstellte Konfigurationsdatei mittels des Programmers dieser Software auf das FPGA-Board übertragen werden, wie es in Kapitel 3.2 beschrieben ist. Wenn die .sof-Datei im JTAG-Mode in das FPGA geladen werden soll, muss der Programming Mode Switch des FPGA-Board in die RUN-Stellung gebracht werden. Wird die .pof-Datei im AS-Mode in das serial configuration device EPCS64 geladen, um das Programm dauerhaft zu speichern, muss der Programming Mode Switch auf PROG gestellt werden[Terasic, 2012].

## 5 Aktueller Stand

Zu Beginn der Studienarbeit war es möglich, das Kamerabild im FPGA zwischenspeichern und über den VGA-Anschluss an einen Monitor auszugeben. Dazu wurden mehrere Funktionsblöcke mittels VHDL in der Software *Quartus Prime 18.1 Lite* geschrieben und mit weiteren Funktionsblöcken verknüpft, die in Abbildung 20 a dargestellt sind. Dabei wurden die Bausteine *cameralink\_tapping\_base*, *camera\_data\_mux\_gen*, *MemoryAccessController*, *SDRAM\_Write\_Buffer\_gen* und *SDRAM\_Read\_Buffer\_gen* entworfen. Der Funktionsblock *vga\_controller* ist ein fertiger Baustein von Scott Larson [Larson, 2018]. Der *dram* sowie der *sdram\_controller\_latency1* sind Funktionsblöcke der Firma Altera Cooperation. Zusätzlich zu diesen Funktionsblöcken erzeugt der Funktionsblock *altpll0* (Abbildung 20 b), der ebenfalls von der Altera Cooperation stammt, die drei Taktsignal *vga\_pixel\_clk* mit 25 MHz, *clk\_pll* mit 100 MHz und *DRAM\_CLK\_TEST* mit 100MHz und  $-60^\circ$  Phasenverschiebung, mit denen ein Teil der anderen Funktionsblöcke arbeitet. Zudem gibt es den Funktionsblock *seg7dec*, der die Ansteuerung der Elemente der Siebensegmentanzeige übernimmt (Abbildung 20 c).

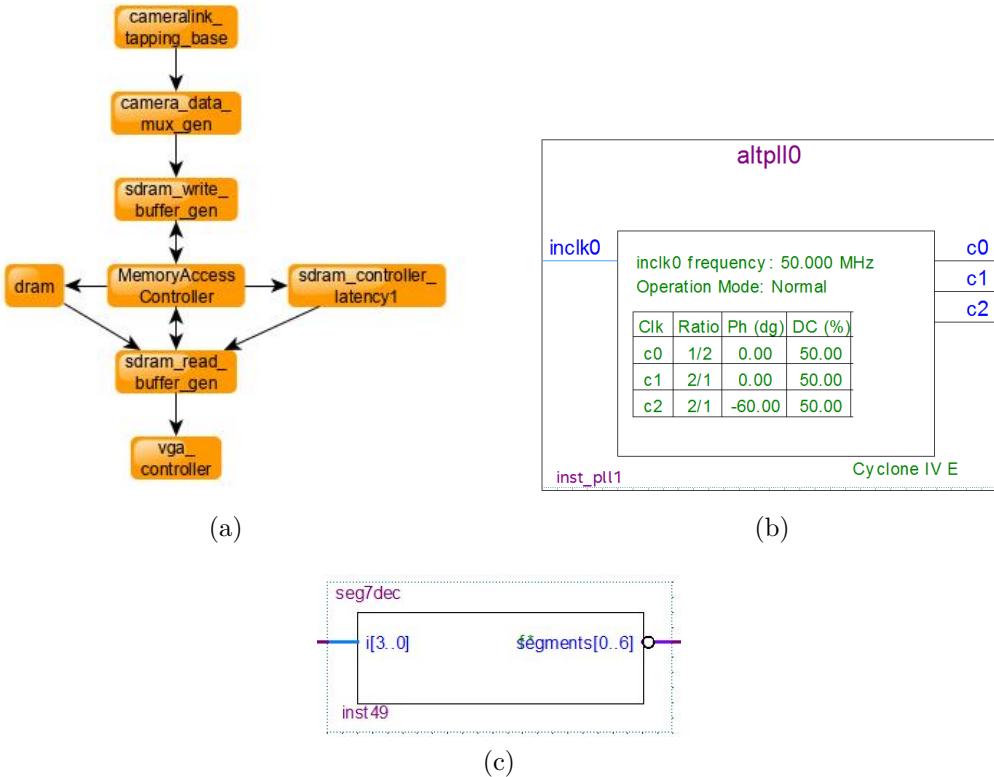


Abbildung 20: Ablauf der Bilddatenverarbeitung vor Beginn der Studienarbeit (a), Funktionsblock *altpll0* zur Generierung der benötigten Taktsignale und Funktionsblock *seg7dec* zur Ansteuerung der Siebensegmentanzeigen (c)

### 5.1 cameralink\_tapping\_base

Über den in Abbildung 21 a abgebildeten Funktionsblock *cameralink\_tapping\_base* werden die Daten, die von der Kamera kommen und an der Eingangsschnittstelle TX\_RX des Funktionsblocks angelegt werden, sortiert. Dies ist notwendig, da die

Daten durch die Übertragung von der Kamera über die Schnittstelle Camera Link zum CLR\_HSMC-Board und anschließend zum FPGA-Board DE2-115 nicht in der richtigen Reihenfolge ankommen. So liegen z.B. die Daten für das erste Pixel in den Bits 0-4, 6, 26 und 5. Diese Funktion sortiert die Daten und gibt diese an seinen Ausgangsschnittstellen aus. An Port A werden die Daten für das erste Pixel ausgegeben, an Port B die für das zweite Pixel und an Port C die Daten für das dritte. Das Ausgangssignal LVAL gibt an, ob die Daten für eine Bildzeile gerade übertragen werden und das Ausgangssignal FVAL, ob momentan ein Bild übertragen wird.

Über die in Abbildung 21 b dargestellten Parameter NUM\_TAPS und NUM\_BITS wird eingestellt, wie viele Pixel mit welcher Bitanzahl pro Takt von der Kamera empfangen und anschließend verarbeitet werden, wobei die Anzahl der Pixel und Bits mit den eingestellten Werten der Kamera, die in Kapitel 4.1 Kamera STC-CMB33PCL beschrieben sind, übereinstimmen müssen (zwei Pixel je acht Bit).

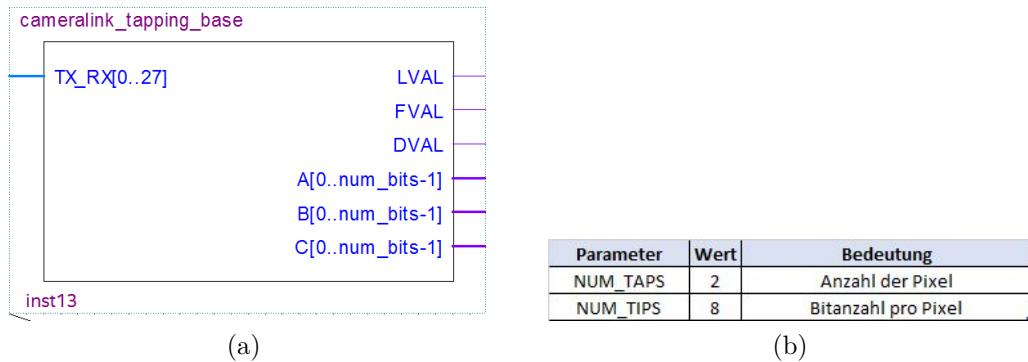


Abbildung 21: Funktionsblock *cameralink\_tapping\_base* (a) mit Parameterbeschreibung (b)

## 5.2 camera\_data\_mux\_gen

Die von dem Funktionsblock *cameralink\_tapping\_base* ausgegebenen Daten LVAL, FVAL, Port A und Port B werden von dem in Abbildung 22 a dargestellten Funktionsblock *camera\_data\_mux\_gen* mit den in Abbildung 22 b beschriebenen Parametern aufgenommen und verarbeitet.

Die Daten für zwei Pixel werden an den Eingangsschnittstellen TAP1[7..0] und TAP2[7..0] aufgenommen und zusammen am Ausgang DATA wieder ausgegeben, was über eine Zustandsmaschine mit sechs Zuständen realisiert wurde. Dabei wird im Startzustand WAIT\_STATE darauf gewartet, dass ein neues Bild verarbeitet werden darf. Dies geschieht entweder dadurch, dass kontinuierlich Bilder verarbeitet werden sollen (SINGLE\_MODE = '0') oder sich ein Signalwechsel des einsynchronisierten Signals SINGLE von '1' auf '0' ereignet, der durch den Taster KEY[1] erzeugt wird, wenn SINGLE\_MODE auf '1' steht. Dadurch wird das Ausgangssignal BUSY auf '1' gesetzt, welches zeigt, dass der Funktionsblock beschäftigt ist, und es wird in den nächsten Zustand WAITFRAME\_STATE gewechselt. In diesem Zustand wird gewartet, bis ein Signalwechsel von '0' auf '1' des Signals FVAL erfolgt, das anzeigt, dass ein neues Bild übertragen wird. Es wird in den Zustand WAITLINE\_STATE gewechselt, in dem verharrt wird, bis die Übertragung der Daten einer neuen Zeile erfolgt. Dies wird durch den Signalwechsel von '0'

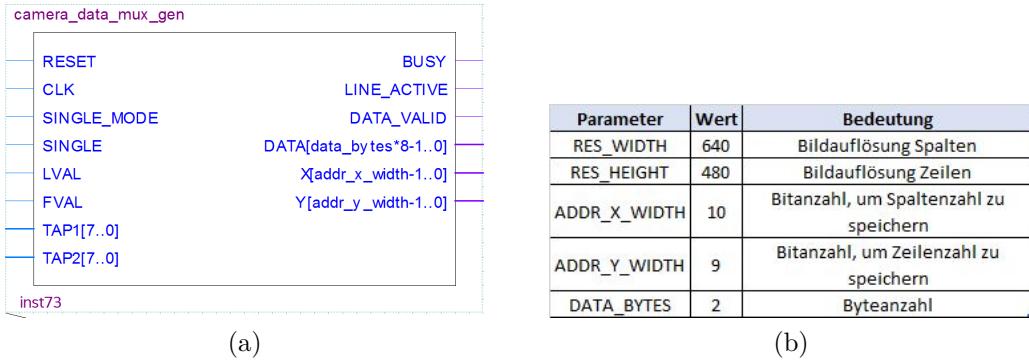


Abbildung 22: Funktionsblock `camera_data_mux_gen` (a) mit Parameterbeschreibung (b)

auf '1' des Signals LVAL angezeigt. Dann werden die Daten der Eingänge TAP1 und TAP2 auf den Ausgang DATA gelegt und das Ausgangssignal DATA\_VALID auf '1' gesetzt, welches anzeigt, dass die Daten des Signals DATA gültig sind und verarbeitet werden dürfen. Zudem wird das Signal LINE\_ACTIVE auf '1' gesetzt, welches signalisiert, dass die Pixeldaten einer Bildzeile ausgegeben werden. Es erfolgt der Wechsel in den Zustand DATA\_OUT\_16BIT. In diesem Zustand wird verblieben, bis alle Pixeldaten der Bildzeile über DATA ausgegeben wurden. Danach wird entweder in den Zustand WAITLINE\_STATE gewechselt, wenn noch nicht alle Zeilen eines Bildes ausgegeben wurden, oder in den Zustand WAIT\_STATE, wenn ein Bild komplett ausgegeben wurde. Die Größe des Bildes wird über die Parameter RES\_WIDTH und RES\_HEIGHT festgelegt, wobei diese auch auf einen kleineren Wert eingestellt werden können als auf die Größe des Bildes, welches die Kamera liefert. Die überschüssigen Zeilen und Spalten gehen dann verloren. Des Weiteren gibt es noch die Zustände DATA\_OUT\_32BIT\_1 und DATA\_OUT\_32BIT\_2, die die gleiche Aufgaben wie der Zustand DATA\_OUT\_16BIT haben. Allerdings werden diese nur benötigt, wenn die Ausgabe über DATA mit 4 Bytes erfolgt anstatt mit 2 Bytes.

Zudem werden auf die Ausgangssignale X und Y immer die x-Koordinate des ersten Pixels und die y-Koordinate der beiden Pixel, die über DATA ausgegeben werden, gelegt. Der Funktionsblock arbeitet mit dem an CLK anliegenden Takt CLRRXCLK\_BASE. Dieser Takt ist der 84 MHz der Kamera, mit dem die Pixel übertragen werden.

### 5.3 MemoryAccessController

Der Funktionsblock MemoryAccessController, der in Abbildung 23 a dargestellt ist, regelt den Zugriff auf dem SDRAM des FPGA-Boards und nutzt das Taktsignal clk\_pll, das am Eingang clk angelegt wird. Zudem sind in Abbildung 23 b die Parameter des Funktionsblockes dargestellt.

Funktionsblöcke, die auf den Speicher zugreifen möchten, senden dem MemoryAccessController eine Zugriffsanfrage an einen der Eingänge von req[7..0] und müssen die Anzahl an Takten warten, die über den Parameter DELAY eingestellt werden, ob nicht ein Funktionsblock mit höherer Priorität auf den SDRAM zugreifen möchte. Die Prioritäten werden über die Parameter PRIO\_0 bis PRIO\_7 vorgegeben. Zusätzlich zu den Requestsignalen muss ein Funktionsblock dem MemoryAccessController mitteilen, auf welche Speicheradresse er zugreifen möchte, indem die Speicheradresse an den entsprechenden Adresseingang adr0[24..0] - adr7[24..0] gelegt wird, und ob er vom Speicher lesen oder schreiben

möchte. Für einen Lesezugriff muss das Signal `rd[7..0]` auf '1' und für einen Schreibzugriff muss das Signal `wr[7..0]` auf '1' gesetzt werden. Will ein Funktionsblock schreiben, muss dieser während des Schreibvorgangs zusätzlich auch die zu schreibenden Daten auf den entsprechenden Dateneingang `dta0[31..0] - dta7[31..0]` legen. Über das Signal `ena[7..0]` erteilt der MemoryAccessController die Zugriffsrechte und koordiniert danach den Zugriff, indem es die Schreib- oder Leseanfrage mit entsprechender Speicheradresse und gegebenenfalls die zu schreibenden Daten an den Speicher weiter gibt. Dies geschieht über die Signale `mem_adr[24..0]`, `mem_data[31..0]`, `mem_rd` und `mem_wr`. Greift ein Funktionsbaustein gerade auf den Speicher zu, signalisiert er dies dem MemoryAccessController über das Setzen des Signals `act[7..0]` auf '1'. Wird in dieser Zeit eine neue Zugriffsanfrage gestellt, wird diese erst akzeptiert, wenn der letzte Zugriff abgeschlossen ist. Dass ein Speicherzugriff abgeschlossen ist, wird dadurch signalisiert, dass das `act[7..0]` wieder auf '0' gesetzt wird.

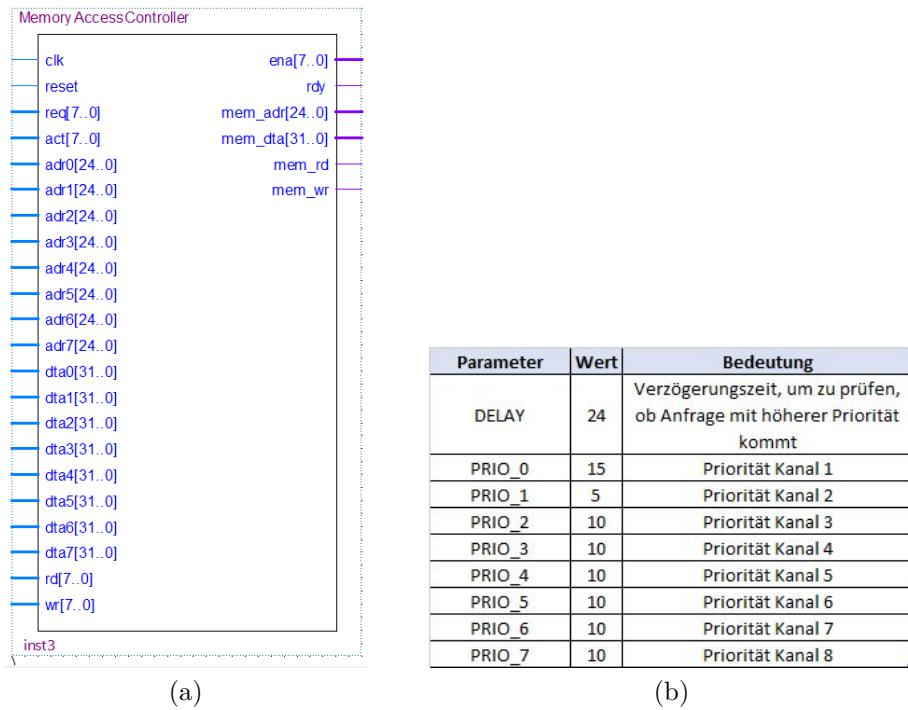


Abbildung 23: Funktionsblock `MemoryAccessController` (a) mit Parameterbeschreibung (b)

## 5.4 SDRAM\_Write\_Buffer\_gen

Mit Hilfe des in Abbildung 24 a gezeigten Funktionsblocks `SDRAM_Write_Buffer_gen` und den in Abbildung 24 b aufgelisteten Parametern werden die Pixeldaten, die er vom Funktionsblock `camera_data_mux_gen` erhält, im SDRAM gespeichert. Über mehrere FirstInFirstOut-Buffer (FIFO), die über eine Zustandsmaschine gefüllt und über eine weitere wieder geleert werden, werden die Kameradaten zwischengespeichert und sobald das Freigabesignal des `MemoryAccessController` erfolgt, werden die Daten im SDRAM gespeichert. Am Eingang `cam_data` erhält der Funktionsblock die Pixeldaten und am Eingang `cam_data_valid` wird deren Gültigkeit signalisiert. Am Eingangssignal `cam_line` wird die Bildzeile angegeben, aus der die aktuellen Pixel stammen.

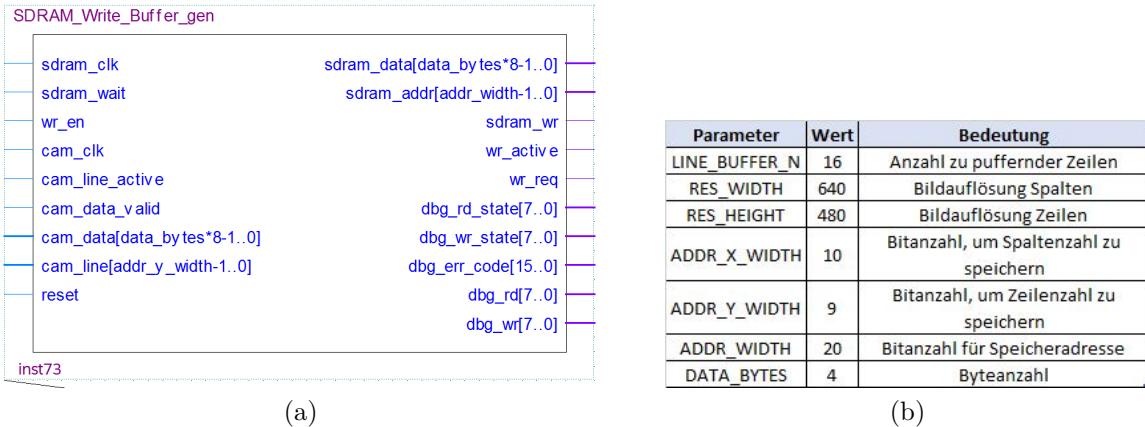


Abbildung 24: Funktionsblock `SDRAM_Write_Buffer_gen` (a) mit Parameterbeschreibung (b)

Über die Zustandsmaschine `rd_state` werden mit dem an `cam_clk` anliegenden Taktsignal `CLRRXCLK_BASE` die Pixeldaten in den FIFO-Buffer geschrieben. Über den Parameter `LINE_Buffer_N` kann eingestellt werden, wie viele Zeilen gepuffert werden sollen. Dazu wird pro Zeile ein FIFO-Buffer angelegt. Sind alle FIFO-Buffer gefüllt, wird die nächste Zeile erst in einen FIFO-Buffer geschrieben, wenn dieser durch das Lesen der Daten aus diesem wieder leer ist.

Das Lesen der Daten aus den FIFO-Buffern, um sie im SDRAM zu speichern, wird mit dem Takt `clk_pll` am Eingang `sdram_clk` und der Zustandsmaschine `wr_state` ermöglicht. Es wird eine Schreibanfrage an den `MemoryAccessController` mittels des Signals `wr_req` gesendet und sobald das Eingangssignal an `wr_en` von '0' auf '1' wechselt, beginnt das Auslesen der FIFO-Buffer. Die Pixeldaten für die über den Parameter `DATA_BYTES` eingestellte Anzahl an Bytes, was der Anzahl der Pixel entspricht, werden auf den Ausgang `sdram_data` gelegt und auf den Ausgang `sdram_addr` die dazugehörige Speicheradresse. Diese wird mittels der Komponente `c_xy_to_address` aus der Zeile und der Spalte, die den x- und y-Koordinaten des Pixels im Bild entsprechen, berechnet und über die Parameter `RES_WIDTH` und `RES_HEIGHT`, die die Größe des Bildes enthalten, wird überprüft, ob das Zeilenende bzw. das Bildende erreicht wurde. Während des Schreibens der Daten auf den SDRAM wird das Signal `wr_active` auf '1' gesetzt, um dem `MemoryAccessController` zu signalisieren, dass dieser Funktionsbaustein noch auf den SDRAM zugreift. Sollte der SDRAM beschäftigt sein und kann deshalb momentan keine Daten verarbeiten, signalisiert er dies dem Funktionsblock durch eine '1' am Eingang `sdram_wait`. Dann wird gewartet, bis der SDRAM nicht mehr beschäftigt ist und das Schreiben der Pixeldaten in diesen wird fortgesetzt.

## 5.5 dram

Dieser Funktionsblock übernimmt die Aufgabe, Daten in den SDRAM des DE2-115-Boards zu speichern oder von diesem zu lesen. Die Schnittstellen sind in Abbildung 25 abgebildet. Der Baustein arbeitet mit dem an `clock` anliegenden Taktsignal. Wenn am Eingang `wren` eine '1' anliegt werden die an `data[31..0]` anliegenden Daten in der Speicheradresse, die am Eingang `address[16..0]` anliegt, abgespeichert. Soll von SDRAM gelesen werden, wird dies über das Signal `rden` signalisiert. Ist dieses auf '1' gesetzt, werden die Daten aus der an `address[16..0]` anliegenden Speicheradresse gelesen und auf den Aus-

gang q[31..0] gelegt. Das Schreiben und Lesen des SDRAM erfolgt über die Komponente altsyncram aus der altera\_mf Bibliothek.

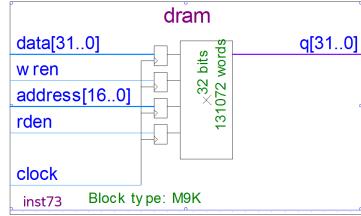


Abbildung 25: Funktionsblock dram

## 5.6 sdram\_controller\_latency1

Der in Abbildung 26 dargestellte Funktionsblock **sdram\_latency\_controller1** hat prinzipiell die gleiche Aufgabe wie der **dram** Funktionsblock. Auch er speichert die Pixeldaten in den SDRAM. An den Eingängen **az\_read** und **az\_write** wird signalisiert, ob gelesen oder geschrieben werden soll. Wenn geschrieben werden soll, werden die Daten des Eingang **az\_writedata** an der Adresse, die an **az\_address** anliegt, gespeichert. Soll gelesen werden, werden die an der angelegten Speicheradresse liegenden Daten auf den Ausgang **za\_readdata** gelegt und durch das Signal **za\_readdatavalid** wird deren Gültigkeit angezeigt. Der Baustein arbeitet mit dem an der Eingangsschnittstelle **az\_clk** anlegten Taktsignal **clk\_pll**. Mittels des Ausgangssignals **za\_waitrequest** signalisiert der **sdram\_controller\_latency1**, dass der auf den Speicher zugreifende Funktionsblock warten muss, da er gerade beschäftigt ist und die Anfrage derzeit nicht bearbeiten kann. Über die restlichen Ausgänge erfolgt das Schreiben der Daten in den Speicher bzw. über das INOUT-Signal **za\_dq** erfolgt auch das Auslesen der Daten aus diesem.

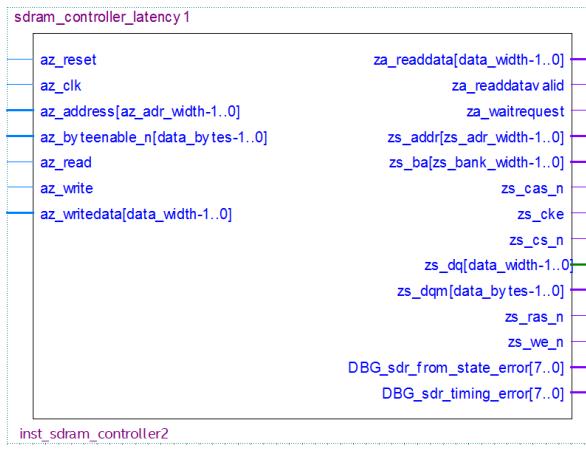


Abbildung 26: Funktionsblock sdram\_controller\_latency 1

## 5.7 SDRAM\_Read\_Buffer\_gen

Die Aufgabe des in Abbildung 27 a dargestellten **SDRAM\_Read\_Buffer\_gen** ist es, die Daten eines Bildes vom SDRAM zu lesen und diese einem nachfolgenden Funk-

tionsbaustein zur Verfügung zu stellen. Er wird genutzt, um die Pixeldaten über den VGA-Ausgang an einem Monitor auszugeben, sodass das gespeicherte Bild wiedergegeben werden kann. Die Bedeutungen der Parameter des Funktionsblocks sind in Abbildung 27 b beschrieben.

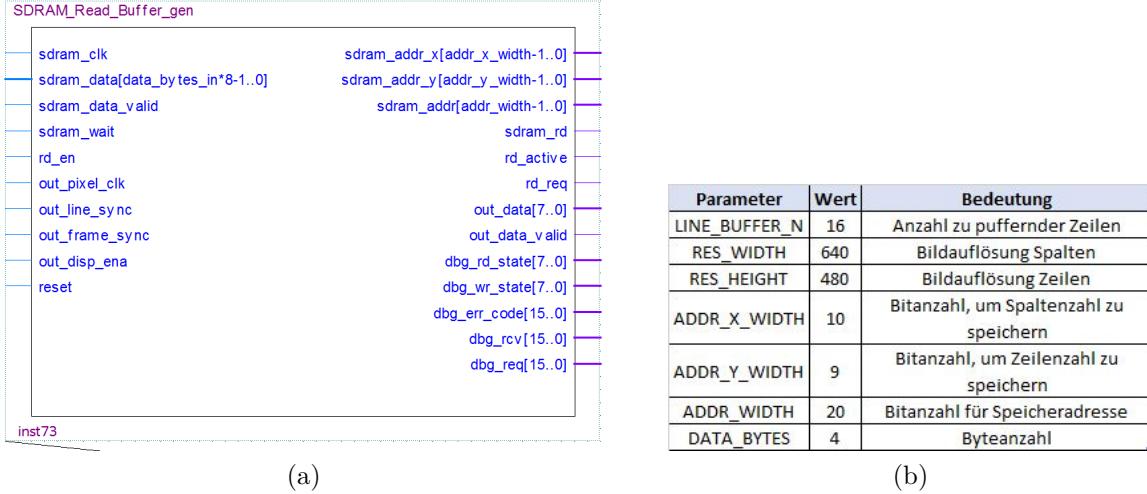


Abbildung 27: Funktionsblock `SDRAM_Read_Buffer_gen` (a) mit Parameterbeschreibung (b)

Das Lesen der Daten vom SDRAM erfolgt über die Zustandsmaschine `rd_state`, die mit dem Takt `clk_pll` an der Eingangsschnittstelle `sdram_clk` arbeitet. Über den Ausgang `rd_req` wird eine Zugriffsanfrage an den MemoryAccessController geschickt und durch den Ausgang `sdram_rd` wird signalisiert, dass vom SDRAM Daten gelesen werden möchten. Der MemoryAccessController signalisiert am Eingang `rd_en`, dass nun auf den SDRAM zugegriffen werden darf. Dann erfolgt das Lesen der Daten, deren Gültigkeit durch das Signal am Eingang `sdram_data_valid` bestätigt wird. Dabei werden die einzelnen Zeilen in einer über den Parameter `LINE_BUFFER_N` einstellbaren Anzahl an FIFO-Buffern gepuffert, wobei pro FIFO-Buffer eine Zeile gespeichert wird. Wenn alle beschrieben wurden, wird das Auslesen des SDRAM solange unterbrochen, bis ein FIFO-Buffer ausgelesen ist.

Über die Zustandsmaschine `wr_state` erfolgt die Ausgabe der in FIFO-Buffern gespeicherten Pixeldaten mit dem Takt `vga_pixel_clk` am Eingang `out_pixel_clk`. Dabei werden pro Takt die Daten für ein Pixel auf den Ausgang `out_data` gelegt und durch eine '1' auf dem Ausgangssignal `out_data_valid` werden diese als gültig gekennzeichnet. Damit die Pixeldaten über VGA auch an der richtigen Stelle und zum richtigen Zeitpunkt ausgegeben werden, wird die Datenausgabe zusätzlich über die Signale an den Eingängen `out_line_sync`, `out_line_frame_sync` und `out_disp_ena` gesteuert, die mit teils invertierten Signalen des `vga_controller` (Kapitel 5.8 `vga_controller`) gespeist werden. Durch einen Signalwechsel vom Signal `out_frame_sync` von '0' auf '1' wird angekündigt, dass die Ausgabe eines neuen Bildes in ein paar Takten beginnen kann. Während das Signal `out_disp_ena` auf '1' ist, erfolgt die Ausgabe der Pixel. Das Signal `out_line_sync` wird zwar aufgenommen, allerdings nicht benutzt.

## 5.8 vga\_controller

Der Funktionsblock vga\_controller von Scott Larson hat die Aufgabe, die Ausgabe der Pixeldaten über den VGA-Ausgang zu steuern [Larson, 2018]. Dieser in Abbildung 28 a dargestellte Block arbeitet mit dem an dem Eingang pixel\_clk angelegten Takt vga\_pixel\_clk, der 25 Mhz beträgt. Zur Steuerung und Synchronisierung der Pixelausgabe werden die low-aktiven Signale h\_sync, v\_sync und das Signal disp\_ena erzeugt, deren Verhalten über die in Abbildung 28 b beschriebenen Parameter eingestellt wird.

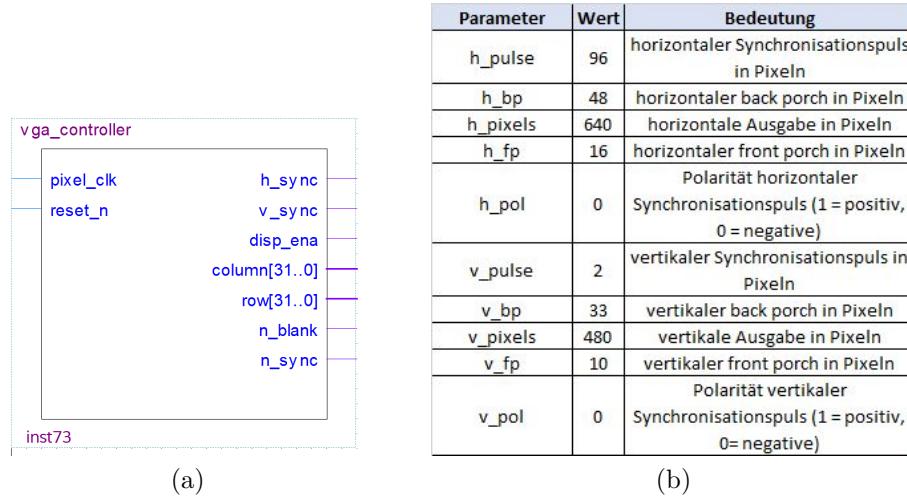


Abbildung 28: Funktionsblock vga\_controller (a) mit Parameterbeschreibung (b)

Zu Beginn der ersten Zeile wechselt das Signal disp\_ena von '0' auf '1'. Dann werden die Pixeldaten für die erste Zeile übertragen. Wurden die Daten für die erste Zeile ausgegeben, geht das Signal disp\_ena wieder von '1' auf '0' zurück. Nach ein paar Takten signalisiert das h\_sync durch einen kurzen '0' Impuls, dass die Übertragung der nächsten Zeile in wenigen Takten beginnen kann. Für die Zeit der Übertragung der Zeile ist das Signal disp\_ena wieder auf '1' gesetzt. Dies wird solange wiederholt, bis ein komplettes Bild ausgegeben wurde und das Signal v\_sync durch einen '0' Impuls ankündigt, dass ein neues Bild ausgegeben werden kann.

## 6 Eindimensionale Positionserkennung verschiedener Objekte

Aufbauend auf den schon vorhandenen Funktionen und den in Kapitel 5 beschriebenen Funktionsblöcken, war es das Ziel, die eindimensionale Positionserkennung von Punkten und Linien und die Ausgabe einer Zeile über den VGA-Ausgang auf einen Monitor zu erreichen. Um dies umzusetzen, wurde zuerst ein Funktionsblock geschrieben, der eine einzige Bildzeile ausgibt, in der nach einem Objekt gesucht werden kann. Anschließend wurden verschiedene Funktionsblöcke entworfen, die die Suche nach einem Objekt in dieser einen Zeile durchführen.

### 6.1 Extraktion einer Bildzeile

Um das Ziel der eindimensionalen Positionserkennung von Objekten, also Punkten und Linien, zu erreichen, wurde als erstes die Pufferung und anschließende Ausgabe von genau einer Zeile benötigt. Dazu ist der Funktionsblock SINGLE\_LINE\_READ\_BUFFER entworfen worden, der mit zusätzlicher äußerer Beschaltung eine vorgegebene Bildzeile puffert und ausgibt, damit diese über den VGA-Ausgang auf dem angeschlossenen Monitor dargestellt werden kann. Um diesen Funktionsblock zu realisieren, ist der in Kapitel 5.7 beschriebene Funktionsblock SDRAM\_Read\_Buffer\_gen als Vorlage genutzt worden.

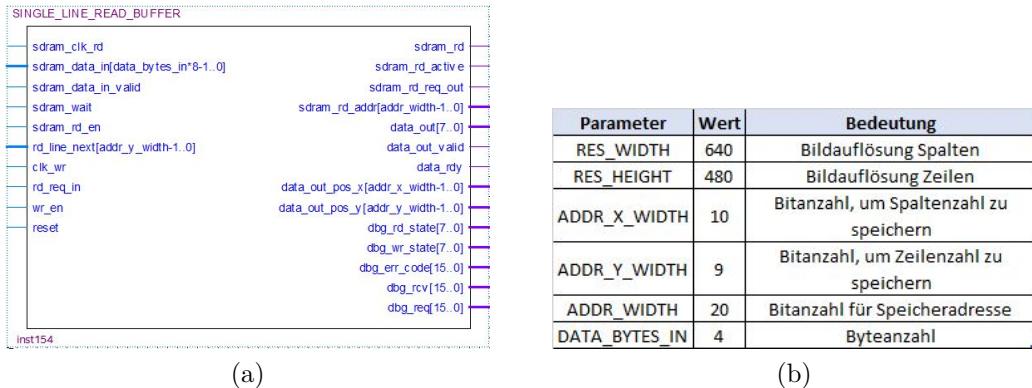


Abbildung 29: Funktionsblock SINGLE\_LINE\_READ\_BUFFER (a) mit Parameterbeschreibung (b)

Auf Grundlage dieses Funktionsblocks wurden einige Funktions- und Signalnamensänderungen vorgenommen, wobei die grundlegende Funktion die gleiche geblieben ist. So wird nun nur noch ein FIFO-Buffer erzeugt, in dem eine Zeile gespeichert wird, die durch das Signal an der Eingangsschnittstelle `rd_line_next` vorgegeben wird. Zudem werden die x- und y-Koordinaten der Pixel, die aus dem Funktionsblock ausgegeben werden, auf die Ausgangssignale `data_out_pos_x` und `data_out_pos_y` gelegt. Das Lesen der Daten vom SDRAM, dessen Schreiben in den FIFO-Buffer und das Auslesen dieser Daten aus diesem erfolgt weiterhin über die beiden Zustandsmaschinen `rd_state` mit dem Takt `clk_pll` am Eingang `sdram_clk_rd` (Abbildung 30 a) und `wr_state` mit dem Takt `vga_pixel_clk` am Eingang `clk_wr` (Abbildung 30 b), wobei hier einige Namensanpassungen von Signalen vorgenommen wurden. Über die Zustandsmaschine `rd_state` werden die Pixeldaten einer Zeile eingelesen und in den FIFO-Buffer geschrieben, sobald über das Eingangssignal `rd_req_in` eine Anfrage zum Lesen einer Zeile erfolgt. Nach dem

Einlesen der Daten ist es möglich, die Daten wieder aus dem FIFO-Buffer auszulesen und pro Takt die Daten eines Pixels auf den Ausgang `data_out[7..0]` zu legen. Dies geschieht, sobald am Eingangssignal `wr_en` eine '1' anliegt. Dabei werden, wie oben erwähnt, zusätzlich die x- und y-Koordinaten der Pixel ausgegeben. Die Bedeutung der Parameter ist in Abbildung 29 b beschrieben, wobei der Wert von `DATA_BYTES_IN` auf den Wert einzustellen ist, der der Ausgabe der Bytes pro Takt aus dem SDRAM entspricht.

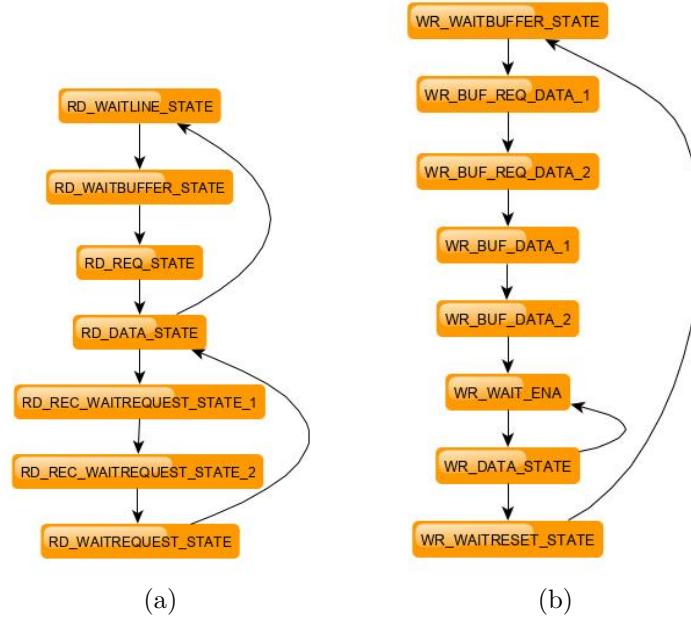


Abbildung 30: Zustandsmaschinen `rd_state` zum Lesen der Daten aus dem SDRAM (a) und dem Puffern dieser in einem FIFO-Buffer; Zustandsmaschine `er_state` zum Auslesen des FIFO-Buffers und das Schreiben der Daten auf den Ausgang (b)

Um die Darstellung dieser einzelnen Bildzeile auf Monitor auch in der entsprechenden Monitorzeile zu ermöglichen, war noch eine äußere Beschaltung notwendig, damit der `SINGLE_LINE_READ_BUFFER` die richtige Zeile puffert und diese Zeile zum richtigen Zeitpunkt wieder ausgibt. Zur Bestimmung, welche Bildzeile eingelesen werden soll, wurde der Funktionsblock `CTU_RTU_LINE` geschrieben. Dieser Funktionsblock gibt als Ausgabewert die auszugebende Zeile aus, die defaultmäßig auf die erste Bildzeile (entspricht y-Koordinate 0) gestellt ist. Durch das Betätigen des Tasters `KEY[2]` wird dieser Wert jeweils um eins erhöht und über `RESET` kann er wieder zurückgesetzt werden. Zur Bestimmung des Zeitpunkts zum Einlesen der Daten in den FIFO-Buffer des `SINGLE_LINE_READ_Buffers` und deren Ausgabe wurden die Signale `h_sync`, `v_sync` und `disp_ena` des `vga_controllers` genutzt. Mittels des Flankenwechsels von '0' auf '1' des invertierten `v_sync` Signals wird das Einlesen der Bildzeile gestartet. Zur Ausgabe der Bildzeile werden die Signale `h_sync` und `disp_ena` genutzt. Allerdings gab es anfänglich Schwierigkeiten, das Eingangssignal `wr_en` des `SINGLE_LINE_READ_BUFFERS` zum richtigen Zeitpunkt auf '1' zu setzen, damit die Ausgabe der Bildzeile erfolgt. Zuerst wurde versucht, über einen selbgeschriebenen Zählerbaustein `CTU_VGA` die Zeile zu bestimmen, die gerade am VGA-Ausgang ausgegeben wurde. Der Zeilenwert wurde um 1 erhöht, sobald das invertierte Signal `h_sync` des `vga_controllers` einen Flankenwechsel von '0' auf '1' aufwies und zurückgesetzt, durch den Flankenwechsel des invertierten

v\_sync-Signals von '0' auf '1'. Mit Hilfe eines geschriebenen Vergleichers wurde das wr\_en-Signal auf '1' gesetzt, sobald die gezählte Zeilennummer des CTU\_VGA der Zeilennummer des CTU\_RD\_LINE entsprach. Diese Beschaltung führte dazu, dass ein x- sowie y-Versatz in der Bildausgabe dieser Zeile vorhanden war. So wurden ca. die ersten 100 Pixel der Zeile nicht ausgegeben und die Ausgabe von der Bildzeile mit der y-Koordinate 0 erfolgte in der letzten Monitorzeile.

Als Grund des x-Versatzes stellte sich heraus, dass das wr\_en Signal schon auf '1' gesetzt wurde und somit der Funktionsblock SINGLE\_LINE\_READ\_BUFFER die Pixeldaten der Bildzeile ausgab, bevor die Ausgabe über den VGA-Ausgang begann. Die Ausgabe einer Bildzeile über VGA erfolgt nur in der Zeit, in der das Signal disp\_vga auf '1' gesetzt ist. Deshalb wurde zwischen dem Ausgang des Vergleichers und der Eingangsschnittstelle wr\_en des SINGLE\_LINE\_READ\_Buffers ein UND-Glied geschaltet, das mit dem Ausgangssignal des Vergleichers und dem Singal disp\_ena des vga\_controllers beschaltet wurde. Jetzt erfolgte die Ausgabe erst, wenn beide Signale auf '1' standen, wodurch es zu keinem x-Versatz mehr bei der Ausgabe der Pixel kam.

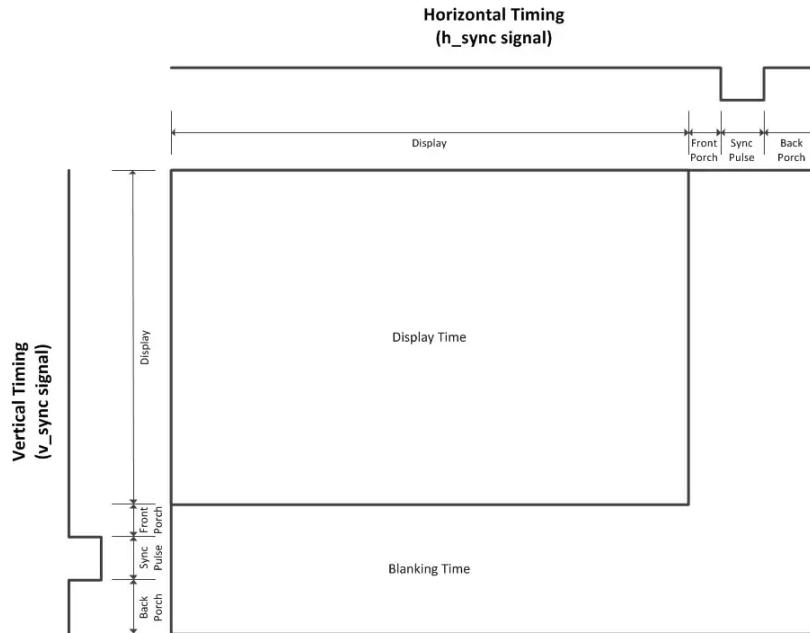


Abbildung 31: Verhalten der Signale des vga\_controllers [Larson, 2018]

Um das Problem mit dem y-Versatz zu lösen, wurde auf Vorschlag von Herrn Prof. Dr. Gick das Verhalten der Signale des vga\_controllers nochmals genauer geprüft, welches in Abbildung 31 dargestellt ist. Zusätzlich wurde das Verhalten der Signale h\_sync und v\_sync mit einem Oszilloskop der Firma Tektronix, dem TDS 3014C 100MHz, aufgezeichnet. Dieses aufgezeichnete Verhalten bestätigte das in Abbildung 31 gezeigte Verhalten dieser Signale. Die Signale h\_sync und v\_sync sind low aktiv, wie es auch der Parameterbeschreibung in 5.8 vga\_controller in Abbildung 28 b des vga\_controllers zu entnehmen ist. Die Parameter h\_pol und v\_pol sind auf '0' gestellt, wodurch die Signale h\_sync und v\_sync mit negativer Logik arbeiten. Das Signal h\_sync ist für die Zeit der Ausgabe der Pixel einer Zeile und den Abschnitt Front Porch auf high. Anschließend ist

es für den Abschnitt Sync Pulse auf low, danach wird es im Abschnitt Back Porch wieder auf high gesetzt. Das Signal v\_sync ist für den Abschnitt Display und Front Porch auf high gesetzt. Danach wird es für den Abschnitt Sync Pulse auf low und abschließend im Abschnitt Back Porch wieder auf high gesetzt. In der Abbildung 32 wird gezeigt, wie viele Takte bzw. Zeilen die jeweiligen Abschnitte bei einer Bildauflösung von 640x480 Pixeln und einer Taktfrequenz von 25,175 MHz auf high oder low gesetzt sind. Zudem lässt sich aus der Abbildung 31 herauslesen, dass das Signal disp\_ena im Abschnitt Display auf high gesetzt und ansonsten low ist.

Resolution (pixels)	Refresh Rate (Hz)	Pixel Clock (MHz)	Horizontal (pixel clocks)				Vertical (rows)				h_sync Polarity	v_sync Polarity
			Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch		
640x350	70	25.175	640	16	96	48	350	37	2	60	p	n
640x350	85	31.5	640	32	64	96	350	32	3	60	p	n
640x400	70	25.175	640	16	96	48	400	12	2	35	n	p
640x400	85	31.5	640	32	64	96	400	1	3	41	n	p
640x480	60	25.175	640	16	96	48	480	10	2	33	n	n

Abbildung 32: Anzahl der Takte der Abschnitte der Signale des vga\_controllers ([Larson, 2018])

Anhand der gewonnenen Erkenntnisse des Verhaltens der Signale des vga\_controllers wurden Anpassungen an der Erzeugung der Signale wr\_en und rd\_line\_next für den SINGLE\_LINE\_READ\_BUFFER vorgenommen. So wurde ein neuer Zähler CTU\_VGA\_V2 entworfen, um die Zeile, in der sich die VGA-Ausgabe befindet, zu zählen. Dieser berücksichtigt zusätzlich als Parameter die Zeiten Front Porch, Sync Pulse und Back Porch des v\_sync Signals. Dadurch war es nun möglich, eine im SINGLE\_LINE\_READ\_BUFFER gepufferte Bildzeile zum richtigen Zeitpunkt auszugeben, sodass sie auch in der entsprechenden Monitorzeile ausgegeben wurde, wie es in Abbildung 33 zu sehen ist.



Abbildung 33: Ausgabe einer Zeile über VGA

Des Weiteren wurde, nach Hinweis von Prof. Dr. Gick asynchrone Signale einzusynchronisieren und das Zählen der Zeile der VGA-Ausgabe über eine Zustandsmaschine zu realisieren, das asynchrone Eingangssignal des Tasters KEY[2] mit Hilfe des Bausteins SYNCHRONIZATION einsynchronisiert und der Funktionsblock CTU\_VGA\_V2 in den Funktionsblock CTU\_VGA\_V3\_state\_machine umgewandelt, bei dem das Zählen der

Zeile der VGA-Ausgabe nun über eine Zustandsmaschine erfolgt. In Abbildung 34 sind die genutzten Funktionsblöcke und die logischen Verbindungen für die beiden Signale wr\_en und rd\_line\_next dargestellt. Diese sind über die Verbindungen mit den Label SLRD\_wr\_en und SLRD\_rd\_line\_next mit den entsprechenden Eingangsschnittstellen des SINGLE\_LINE\_READ\_BUFFERS verbunden.

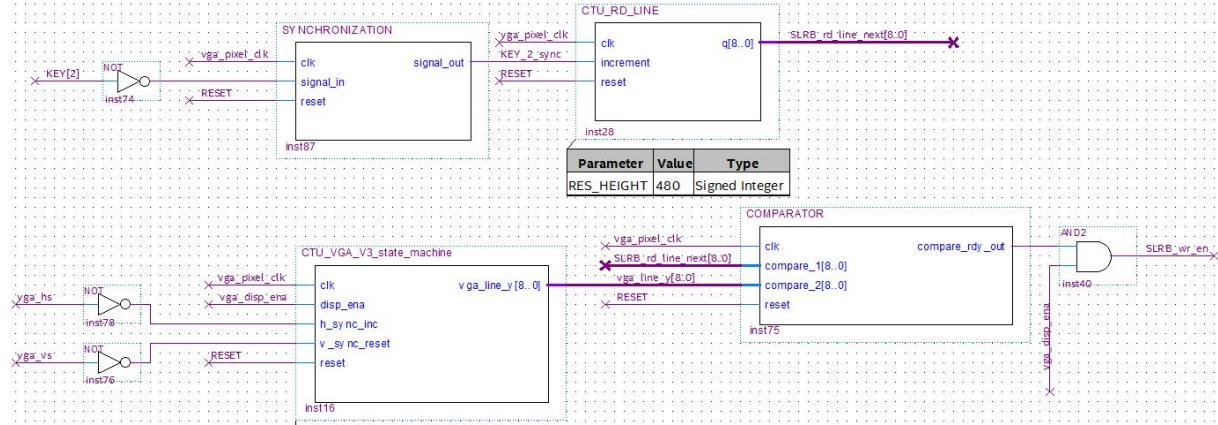


Abbildung 34: Erzeugung der Eingangssignale rd\_line\_next und wr\_en des SINGLE\_LINE\_READ\_BUFFERS

Zusätzlich wurde der in Abbildung 35 a gezeigte Funktionsblock GREY\_LEVEL\_GENERATOR entworfen, um die Farbintensität der Zeilen, die nicht vom SINGLE\_LINE\_READ\_BUFFER über VGA ausgegeben werden, stufenweise zu verändern (Abbildung 35). Dies wurde genutzt, um die extrahierte Zeile auf dem Monitor besser zu erkennen, da das aufgenommene Kamerabild anfangs einen dunklen Hintergrund besaß, da noch nicht mit einem weißen Blatt Papier als Standardhintergrund gearbeitet wurde.



Abbildung 35: Funktionsblock GREY\_LEVEL\_Generator (a) und mit dem GREY\_LEVEL\_GENERATOR bearbeitete Ausgabe (b)

## 6.2 Eindimensionale Positionserkennung eines isolierten Punktes

Basierend auf der Extraktion einer Bildzeile und deren Ausgabe mittels des SINGLE\_LINE\_READ\_BUFFERS über VGA erfolgte die Implementierung einer eindimensionalen Objekterkennung für einen isolierten Punkt. Dabei beruht das Konzept der Positionserkennung auf dem Mitlesen der Pixelwerte der isolierten Zeile und der Suche nach einem isolierten Punkt mittels mathematischer Bildoperationen.

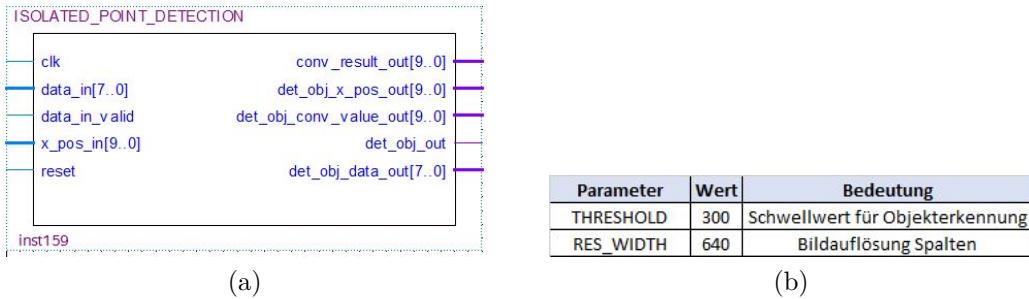


Abbildung 36: Funktionsblock ISOLATED\_POINT\_DETECTION (a) mit Parameterbeschreibung (b)

Die Umsetzung der Positionserkennung erfolgte durch den Entwurf des Funktionsblocks ISOLATED\_POINT\_DETECTION, der in Abbildung 36 a dargestellt ist. Die Berechnung und Suche nach einem isolierten Punkt erfolgt mit Hilfe der in Kapitel 2.3 beschriebenen Faltung mit der aus der 2. Ableitung der eindimensionalen Bildfunktion  $f(x)$  resultierenden Gewichtungsmatrix. Anhand der Ergebnisse der Faltungssumme an jeder Stelle einer Zeile kann darauf geschlossen werden, ob ein isolierter Punkt vorliegt oder nicht. Dafür müssen die Werte Pixel eingelesen und die Faltung durchgeführt werden, was über die in Abbildung 37 gezeigte Zustandsmaschine erfolgt.

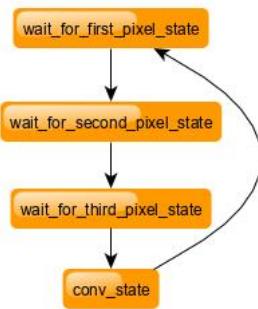


Abbildung 37: Zustandsmaschine ISOLATED\_POINT\_DETECTION

Im ersten Zustand **wait\_for\_first\_pixel\_state** wird der Pixelwert, der am Eingang **data\_[7..0]** anliegt, eingelesen, sobald das Signal am Eingang **data\_in\_valid** von '0' auf '1' wechselt. Dieser Wert entspricht dem Pixel an Stelle  $f(x-1)$ . Zudem wird die x-Koordinate des Pixles am Eingang **x\_pos\_in[9..0]** aufgenommen, damit bei einem gefundenen isolierten Punkt die Position ausgegeben werden kann. Zudem erfolgt der

Wechsel in den zweiten Zustand `wait_for_second_pixel_data_state`. In diesem wird der Wert und die x-Koordinate des zweiten Pixels eingelesen, der dem Wert an Stelle  $f(x)$  entspricht und in den dritten Zustand `wait_for_third_pixel_state` gewechselt, in dem der Wert für das dritte Pixel ( $f(x+1)$ ) sowie dessen x-Koordinate aufgenommen werden. Anschließend erfolgt der Wechsel in den Zustand `conv_state`.

Im `conv_state` Zustand wird das Ergebnis der Faltungssumme für die ersten drei eingelesenen Pixeldaten mit der Gewichtungsmatrix der 2. Ableitung berechnet. Die Berechnung ist in Abbildung 38 dargestellt. Da das Faltungsergebnis auch ein negatives Ergebnis sein kann, muss der Datentyp `std_logic_vector` der Pixeldaten in einen `signed`-Datentyp umgewandelt werden und zusätzlich muss die Länge der Daten angepasst werden. Dies geschieht über die selbstgeschriebene Funktion `std_to_sig_resize`, in der zunächst der Datentyp `std_logic_vector` in den `unsigned`-Datentyp umgewandelt, anschließend auf eine vorgegebene Größe verlängert und dann in den `signed`-Datentyp gewandelt wird. Falls das Ergebnis der Faltung einen Schwellwert, der über den in Abbildung 36 b beschriebenen Parameter `THRESHOLD` eingestellt wird, überschreitet, ist ein isolierter Punkt gefunden. Dann wird dies am Ausgang `det_obj_out` durch einen Signalwechsel von '0' auf '1' signalisiert. Zudem wird die x-Koordinate des gefundenen Punktes über den Ausgang `det_obj_x_pos_out` ausgegeben. Wird der Schwellwert nicht überschritten, werden die Daten für das nächste Pixel eingelesen. Dadurch wird das Pixel  $f(x)$  zum Pixel  $f(x-1)$ , das Pixel  $f(x+1)$  zum Pixel  $f(x)$  und das neu aufgenommene zum Pixel  $f(x+1)$ . Das vorherige Pixel  $f(x-1)$  fällt heraus, da es zur Berechnung nicht mehr benötigt wird. Anschließend wird im nächsten Takt erneut die Faltungssumme berechnet. Dieser Ablauf wird `conv_state` solange wiederholt, bis ein isolierter Punkt gefunden oder das Zeilenende erreicht wurde, das durch den Parameter `RES_WIDTH` bestimmt wird, welcher der Breite einer Bildzeile entspricht.

```
conv_result_var := std_to_sig_resize(data_in_ff_1, 10)
                  - std_to_sig_resize(data_in_ff_2, 10)
                  - std_to_sig_resize(data_in_ff_2, 10)
                  + std_to_sig_resize(data_in_ff_3, 10);
```

Abbildung 38: Programmausschnitt zur Berechnung der Faltung

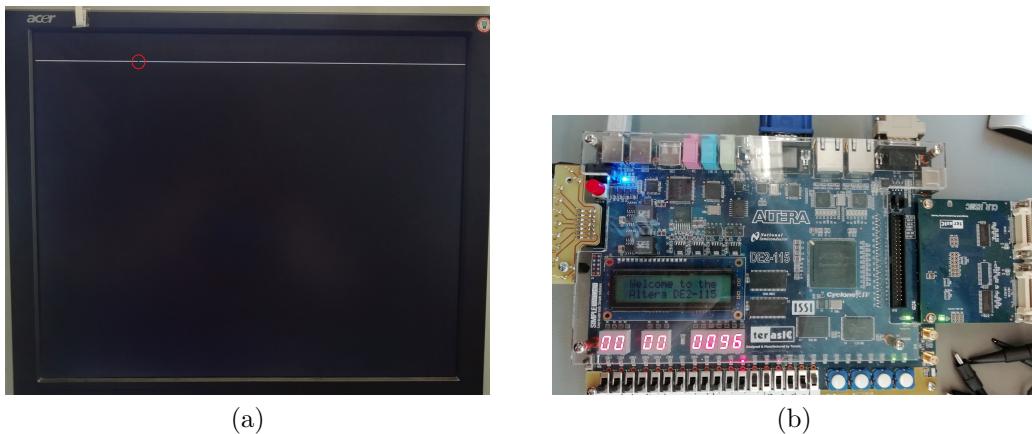


Abbildung 39: Darstellung eines isolierten Punktes in einer Bildzeile (a) und Ausgabe der x-Koordinate auf dem FPGA-Board (b)

Um die Funktion des ISOLATED\_POINT\_DETECTION Funktionsblocks zu testen, wurde der in Abbildung 40 a gezeigte Funktionsblock ISOLATED\_POINT\_GENERATOR geschrieben. Dieser erzeugt in einer Zeile an einer vorgegebenen x-Koordinate einen Punkt mit einer einstellbaren Intensität, wie es in Abbildung 39 a zu sehen ist. Beides ist über die in Abbildung 40 b beschriebenen Parameter einstellbar. Als Hintergrund für das von der Kamera aufgenommene Bild wurde ein weißes Blatt Papier genutzt. Mithilfe des ISOLATED\_POINT\_GENERATOR wurde dann in der Zeile, die der SINGLE\_LINE\_READ\_BUFFER extrahiert, eine schwarzer Punkt an der x-Koordinate 150 erzeugt. Dieser schwarze Punkt wurde als isolierter Punkt erkannt und die eingestellte Position korrekterweise als x-Koordinate auf der Siebensegmentanzeige des FPGA-Boards hexadezimal dargestellt (Abbildung 39 b). Ist anstatt eines schwarzen Punktes ein Punkt mit geringerer Intensität erzeugt worden, ist dieser nicht als isolierter Punkt erkannt worden, wenn das Ergebnis der Faltung nicht den eingestellten Schwellwert im ISOLATED\_POINT\_DETECTION-Funktionsblock überschritten hat.

Des Weiteren wurde getestet, ob der Funktionsblock eine Folge von schwarzen Pixeln auf weißem Hintergrund bzw. die Kante der Linie nicht als isolierten Punkt erkennt. Dafür wurde auf das weiße Papier eine schwarze Linie gezeichnet, die der Funktionsblock wie gewünscht nicht als isolierten Punkt erkennt. War der Schwellwert allerdings niedriger als 256 eingestellt, wurde teilweise die Kante einer Linie als isolierter Punkt erkannt.

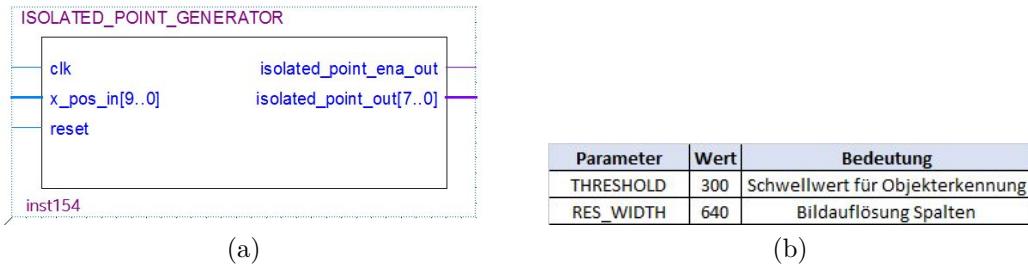


Abbildung 40: Funktionsblock ISOLATED\_POINT\_GENERATOR (a) mit Parameterbeschreibung (b)

### 6.3 Eindimensionale Positionserkennung mehrerer isolierter Punkte

Nach der Positionserkennung eines isolierten Punktes wurde die Erkennung so erweitert, dass mehrere isolierte Punkte als Objekte in einer Zeile erkannt werden können. Dazu wurde der Funktionsblock ISOLATED\_POINT\_DETECTION\_2 (Abbildung 41 a) entworfen. Das Grundkonzept dieses Funktionsblocks ist gleich geblieben. So werden weiterhin die Daten der Zeile, die der SINGLE\_LINE\_READ\_BUFFER für die Ausgabe über VGA ausgibt, mitgelesen und über eine Zustandsmaschine aufgenommen und ausgewertet.

Dieser Funktionsblock arbeitet grundsätzlich mit der gleichen Zustandsmaschine wie der ISOLATED\_POINT\_DETECTION-Funktionsblock, die in Abbildung 42 zu sehen ist. In den Zuständen `wait_for_first_pixel_state`, `wait_for_second_pixel_state` und `wait_for_third_pixel_state` werden die Daten der ersten drei Pixel einer Zeile sowie deren x-Koordinate eingelesen.

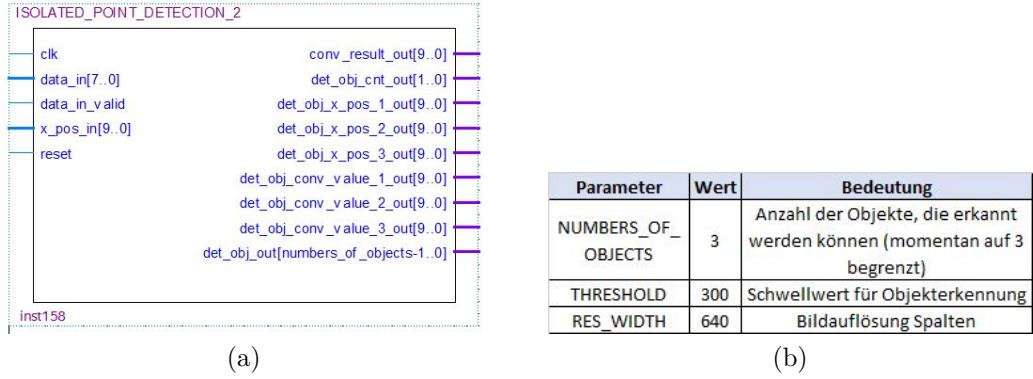


Abbildung 41: Funktionsblock `ISOLATED_POINT_DETECTION_2` (a) mit Parameterbeschreibung (b)

Im Zustand `conv_state` wird die Faltungssumme für das Pixel an der Stelle  $f(x)$  aus den Intensitätswerten der Pixel und der in Kapitel 2.3 beschriebenen Gewichtungsmatrix der 2. Ableitung auf die gleiche Weise wie beim Funktionsblock `ISOLATED_POINT_DETECTION` in Kapitel 6.2 berechnet. Dabei entspricht das im Zustand `wait_for_first_pixel_state` eingelesene Pixel dem Pixel an der Stelle  $f(x-1)$ , das aus Zustand `wait_for_second_pixel_state` dem Pixel an Stelle  $f(x)$  und das im Zustand `wait_for_third_pixel_state` eingelesene dem Pixel an Stelle  $f(x+1)$ . Überschreitet das Ergebnis der Faltung an dieser Stelle den über den Parameter `THRESHOLD` eingestellten Schwellwert, ist ein isolierter Punkt gefunden worden. Ist dies der Fall, wird der Zähler `det_obj_cnt_out` um eins erhöht und zudem signalisiert der Ausgang `det_obj_out[0]` durch einen Signalwechsel von '0' auf '1', dass ein Objekt gefunden wurde. Wurde kein isolierter Punkt gefunden oder sind noch nicht die über den Parameter `NUMBERS_OF_OBJECTS` eingestellte Anzahl an Objekten, die auf momentan 3 begrenzt ist, erreicht, wird der Wert des nächsten Pixels eingelesen und mit den drei zuletzt eingelesenen Pixelwerten die Faltungssumme des mittleren Pixels dieser drei berechnet. Dies wird solange wiederholt, bis die eingestellte Anzahl an Objekten gefunden oder das Zeilenende erreicht wurde, das durch den in Abbildung 41 b beschriebenen Parameter `RES_WIDTH` festgelegt wird. Wird ein zweiter isolierter Punkt gefunden, wird dies durch den Ausgang `det_obj_out[1]` signalisiert und für einen dritten durch den Ausgang `det_obj_out[2]`.

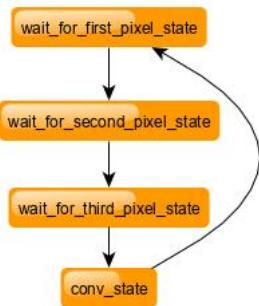


Abbildung 42: Zustandsmaschine `ISOLATED_POINT_DETECTION_2`

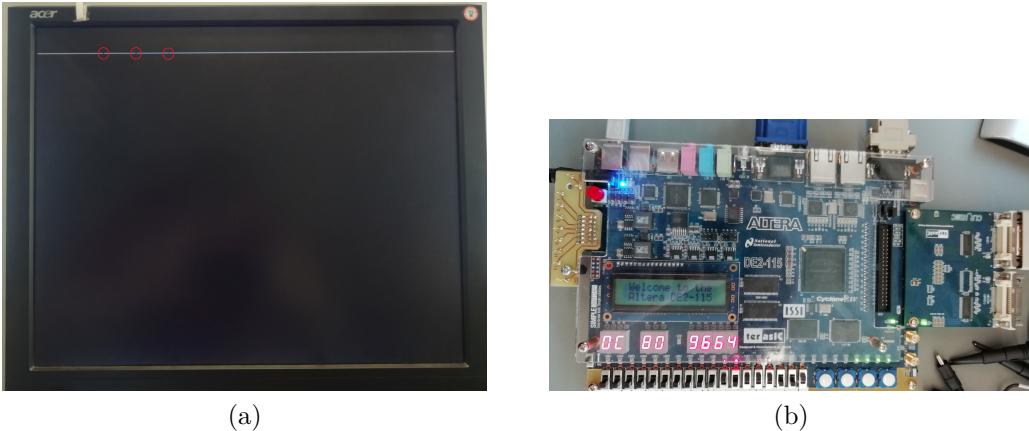


Abbildung 43: Darstellung mehrerer isolierter Punkte in einer Bildzeile (a) und Ausgabe deren x-Koordinate auf dem FPGA-Board (b)

Zum Testen dieses Funktionsblocks wurden mithilfe des in Abbildung 44 a gezeigten Funktionsblocks ISOLATED\_POINT\_GENERATOR\_2 isolierte Punkte erzeugt, die gefunden werden sollten. Dabei erzeugt der ISOLATED\_POINT\_GENERATOR\_2 im Gegensatz zum ISOLATED\_POINT\_GENERATOR drei isolierte Punkte, deren x-Position und Intensität über die beschriebenen Parameter in Abbildung 44 b eingestellt werden. So wurden drei schwarze Punkte an den x-Koordinaten 100, 150 und 200 erzeugt (Abbildung 43 a). Diese eingestellten Punkte wurden auch als isolierte Punkte durch den ISOLATED\_POINT\_DETECION\_2-Funktionsblock erkannt, die entsprechenden x-Koordinaten herausgegeben und auf den Siebensegmentanzeigen des FPGA-Boards hexadezimal dargestellt (Abbildung 43 b). Zudem wurden die Punkte wie gewünscht nicht erkannt, wenn sie zu hell waren und sich so nicht stark genug vom weißen Hintergrund abhoben, da der Intensitätsunterschied nicht mehr ausreichte, um den Schwellwert bei der Berechnung der Faltungssumme an dieser Stelle zu überschreiten. Wie beim ISOLATED\_POINT\_DETECTION-Funktionsblock wird die Kante einer Linie als isolierter Punkt erkannt, wenn ein niedriger Schwellwert eingestellt ist.

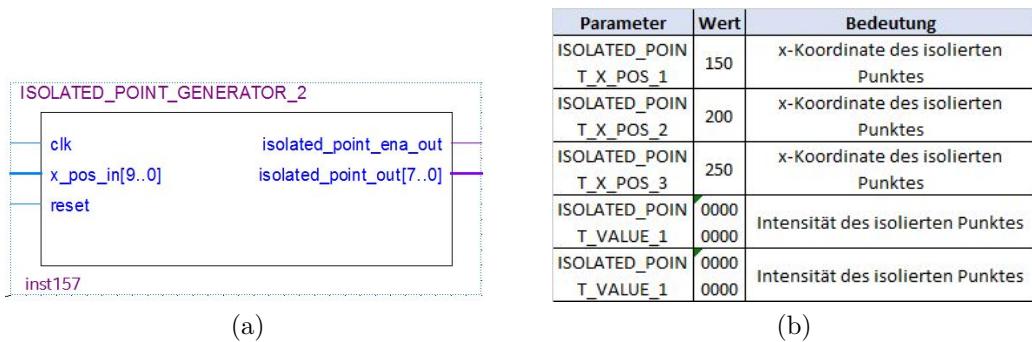


Abbildung 44: Funktionsblock ISOLATED\_POINT\_GENERATOR\_2 (a) mit Parameterbeschreibung (b)

## 6.4 Eindimensionale Positionserkennung einer Linie

Nach der Suche von isolierten Punkten in einer Zeile erfolgte die Objekt- und Positions-erkennung einer Linie in einem eindimensionalen Bild. Eine Linie bzw. ein Linienobjekt

ist eine Folge von Pixeln mit gleicher oder sehr ähnlicher Intensität, die sich in ihrer Intensität vom Hintergrund abhebt. Diese Erkennung baut wie die Erkennung eines isolierten Punktes auf den in Kapitel 2.3 beschriebenen mathematischen Operationen auf. Allerdings wird hier im Gegensatz zur Suche eines isolierten Punktes die 1. Ableitung der Funktion  $f(x)$  und der daraus gewonnenen Gewichtungsmatrix anstatt der 2. Ableitung und deren Matrix genutzt.

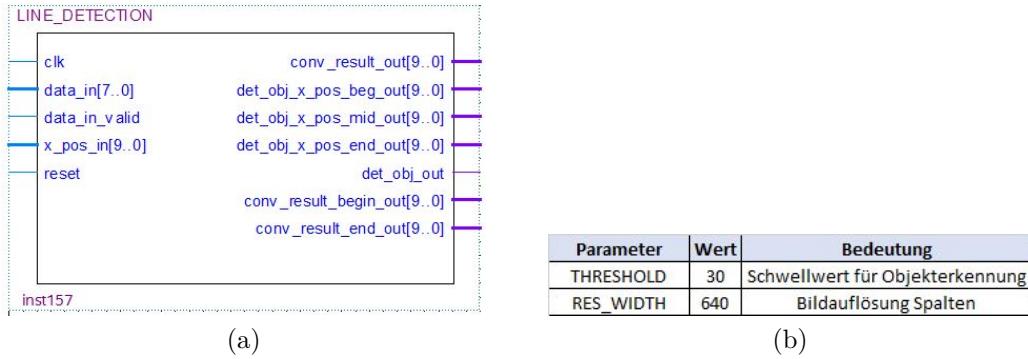


Abbildung 45: Funktionsblock **LINE\_DETECTION** (a) mit Parameterbeschreibung (b)  
**ISOLATED\_POINT\_GENERATOR\_2** (a) mit Parameterbeschreibung (b)

Zur Umsetzung der Objekt- und Positionserkennung wurde der Funktionsblock **LINE\_DETECTION** (Abbildung 45 a) entworfen. Dieser sucht mittels einer Zustandsmaschine nach einem Linienobjekt in einer Zeile. Dabei wurde davon ausgegangen, dass der Hintergrund weiß ist und die Linie schwarz, bzw. dunkler ist als der Hintergrund. Mittels der in Abbildung 46 dargestellten Zustandsmaschine erfolgt die Objekt- und Positionserkennung einer schwarzen Linie in einer Bildzeile. Im ersten Zustand `wait_for_first_pixel_state` wird der Wert des ersten Pixels einer Zeile, der am Eingang `data_in[7..0]` anliegt sowie die x-Koordinate dieses Pixels am Eingang `x_pos_in[9..0]` eingelesen, sobald das Signal am Eingang `data_in_valid` von '0' auf '1' wechselt. Zudem erfolgt der Wechsel in den zweiten Zustand `wait_for_second_pixel_state`. In diesem Zustand werden die Daten des zweiten Pixels und dessen x-Koordinate aufgenommen und es wird in den dritten Zustand `detect_start_of_line_state` gewechselt. Hier wird die Faltung für jedes Pixel mit der Gewichtungsmatrix der 1. Ableitung berechnet, beginnend mit den beiden bereits eingelesenen Pixelwerten. Dabei entspricht das im Zustand `wait_for_first_pixel_state` aufgenommene Pixel dem Pixel an der Stelle  $f(x)$  und das im Zustand `wait_for_second_pixel_state` dem Pixel an Stelle  $f(x+1)$ . Unterschreitet die Faltungssumme den negativen Wert des über den Parameter `THRESHOLD` eingestellten Schwellwerts, ist der Anfang einer Linie gefunden worden. Es wird in den Zustand `detect_end_of_line_state` gewechselt und die Daten des nächsten Pixels werden eingelesen. Wird im Zustand `detect_start_of_line_state` der negative Schwellwert nicht unterschritten, erfolgt das Einlesen des nächsten Pixelwertes und im nächsten Takt wird die Faltungssumme für die beiden zuletzt aufgenommenen Pixelwerte berechnet. Dies wird solange wiederholt, bis ein Linienanfang gefunden oder das Zeilenende erreicht wurde. Im Zustand `detect_end_of_line` erfolgt die Berechnung der Faltungssumme für jedes Pixel, bis das Ende eines Linienobjektes gefunden wurde. Dies ist der Fall, wenn das Ergebnis der Faltung an einer Stelle den Schwellwert überschreitet. Sollte allerdings bis zum Zeilenende kein Ende der Linie gefunden worden sein, ist auch kein Linienobjekt detektiert worden.

Ist der Anfang einer Linie gefunden worden, wird dessen x-Koordinate zwischengespeichert. Wird anschließend das Linienende noch detektiert, wird auch dessen x-Koordinate gespeichert. Aus diesen beiden Werten wird der Mittelpunkt des Linienobjektes berechnet und es werden alle drei x-Koordinaten über die Ausgänge `det_obj_x_pos_beg_out[9..0]`, `det_obj_x_pos_mid_out[9..0]` und `det_obj_x_pos_end_out[9..0]` ausgegeben.

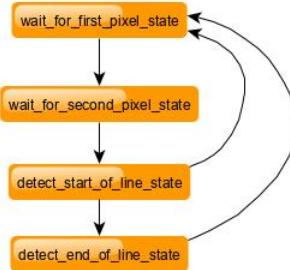


Abbildung 46: Zustandsmaschine LINE\_DETECTION

Zum Test dieses Blocks wurde auf ein weißes Papier ein schwarzer Strich gezeichnet, der als Linie in einem eindimensionalen Bild erkannt werden soll. Dieser schwarze Strich wurde auch als Linienobjekt erkannt und die x-Koordinate des Objektbeginns auf den Siebensegmentanzeige angezeigt (Abbildung 47a und b). Über die Schalter SW[8] und SW[9] des FPGA-Boards kann eingestellt werden, ob der Start-, Mittel- oder Endpunkt des gefundenen Objektes ausgegeben werden soll. Wurden mehrere schwarze Striche in eine Zeile gezeichnet wurde nur die erste schwarze Linie als ein Objekt erkannt, wie auch angedacht war. Zur Überprüfung, ob die x-Koordinaten auch korrekt berechnet und ausgegeben werden, wurde mit Hilfe des in Kapitel 6.3 beschriebenen Funktionsblocks ISOLATED\_POINT\_GENERATOR\_2 drei schwarze Punkte an drei aufeinanderfolgenden x-Koordinaten erzeugt, die auf diese Weise eine schwarze Linie mit bekannten x-Koordinaten erzeugt. Diese Linie wurde beim Testen als Linienobjekt erkannt und dessen x-Koordinaten auch korrekt auf den Siebensegmentanzeigen ausgegeben.

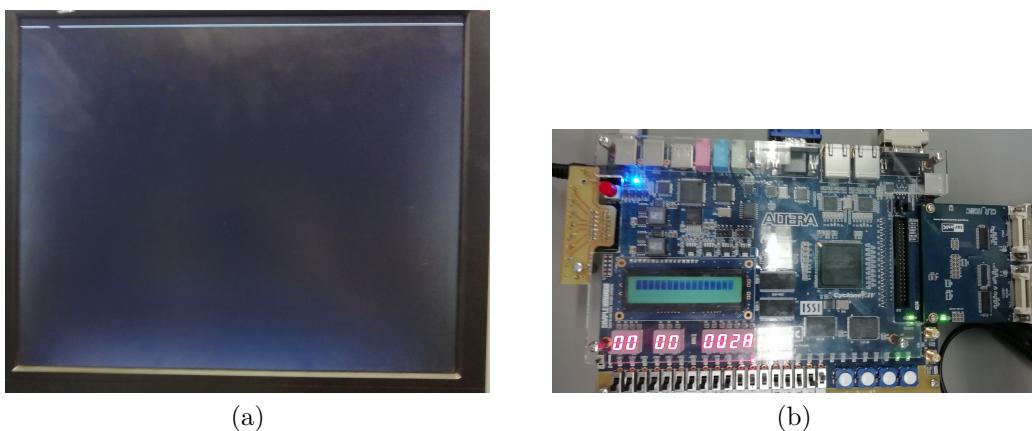


Abbildung 47: Darstellung eines Linienobjekts in einer Bildzeile (a) und Ausgabe der x-Koordinaten des Objektbeginns auf dem FPGA-Board (b)

## 6.5 Eindimensionale Positionserkennung mehrerer Linien

Aufbauend auf der Suche einer Linie in einem eindimensionalen Bild erfolgt die Ausweitung der Suche auf mehrere Linien in einer Bildzeile. Dazu wurde der in Abbildung 48 a gezeigte Funktionsblock LINE\_DETECTION\_2 geschrieben, der im Grunde auf der selben Zustandsmaschine und der selben Berechnung zur Erkennung des Anfangs und des Endes einer Linie beruht wie der in Kapitel 6.4 beschriebene Funktionsblock LINE\_DETECTION. Diese beiden Funktionsblöcke unterscheiden sich lediglich darin, dass der LINE\_DETECTION\_2 mehrere Linien erkennen kann und dadurch der Ablauf in den Zuständen detect\_start\_of\_line\_state und detect\_end\_of\_line\_state etwas abweicht.

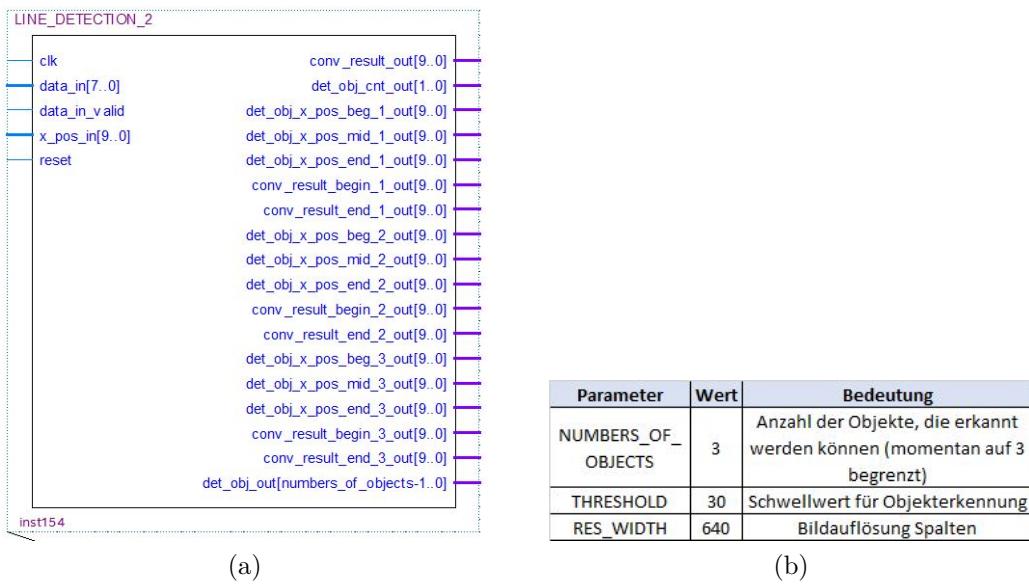


Abbildung 48: Funktionsblock LINE\_DETECTION\_2 (a) mit Parameterbeschreibung (b)

In der in Abbildung 49 dargestellten Zustandsmaschine des LINE\_DETECTION\_2 Funktionsblocks wird im ersten Zustand wait\_for\_first\_pixel\_state der Wert des ersten Pixels einer Zeile am Eingang data\_in[7..0] eingelesen, wenn der Signalwechsel am Eingang data\_in\_valid von '0' auf '1' erfolgt. Zudem wird die x-Koordinate des Pixels über den Eingang x\_pos\_in[9..0] aufgenommen. Anschließend erfolgt der Wechsel in den Zustand wait\_for\_second\_pixel\_state, in dem beim nächsten Takt der Wert des nächsten Pixels und dessen x-Koordinate aufgenommen werden. Daraufhin wird in den Zustand detect\_start\_of\_line\_state gewechselt. In diesem Zustand wird die Faltung der Pixelwerte mit der Gewichtungsmatrix der 1. Ableitung durchgeführt. Zuerst wird die Faltungssumme für die in den Zuständen wait\_for\_first\_pixel\_state und wait\_for\_second\_pixel\_state eingelesenen Pixelwerte berechnet, wobei das im Zustand wait\_for\_first\_pixel\_state aufgenommene Pixel dem Pixel an der Stelle  $f(x)$  und das im Zustand wait\_for\_second\_pixel\_state dem Pixel an der Stelle  $f(x+1)$  entspricht. Ist das Ergebnis der Faltung an dieser Stelle kleiner als der negative Schwellwert, dessen Wert über den Parameter THRESHOLD eingestellt wird, ist der Anfang einer Linie gefunden worden. Dann wird die x-Koordinate des Linienanfangs zwischengespeichert, der Wert des nächsten Pixels eingelesen und in den Zustand detect\_end\_of\_line\_state

gewechselt. Wird der negative Schwellwert nicht unterschritten, erfolgt das Einlesen des Wert des nächsten Pixels. Die Faltung für die Pixel einer Zeile wird solange berechnet, bis der Anfang eines Linienobjektes gefunden oder das Ende der Bildzeile erreicht wurde, welches durch den in Abbildung 48 b gezeigten Parameter RES\_WIDTH bestimmt wird. Im Zustand detect\_end\_of\_line\_state wird die Faltung fortgesetzt, bis das Ende des Linienobjektes gefunden wurde. Das Ende der Linie wurde gefunden, wenn das Ergebnis der Faltung an einer Stelle den Schwellwert überschreitet. Dann wird die x-Koordinate gespeichert und der Mittelpunkt der Linie berechnet und alle drei Werte für das erste gefundene Objekt über die Ausgänge det\_obj\_x\_pox\_beg\_1\_out[9..0], det\_obj\_x\_pox\_mid\_1\_out[9..0] und det\_obj\_x\_pox\_end\_1\_out[9..0] ausgegeben. Zudem wird der Zähler det\_obj\_out um 1 erhöht, der die Anzahl gefundener Objekte anzeigt. Anschließend wird in den Zustand det\_start\_of\_line\_state gewechselt und der Beginn des nächsten Linienobjekts gesucht. Dieser Ablauf wird solange wiederholt, bis die über den Parameter NUMBERS\_OF\_OBJECTES eingestellte Anzahl an Objekten, die auf drei begrenzt wurde, gefunden oder das Zeilenende erreicht wurde.

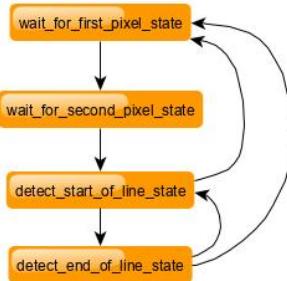


Abbildung 49: Zustandsmaschine LINE\_DETECTION\_2

Um die Funktion zu überprüfen, wurde ein weißes Blatt Papier als Hintergrund genutzt und dort mehrere schwarze Striche eingezeichnet, die als Linienobjekte in einer Zeile gefunden werden sollen. Zudem wurde die Ausgangslogik der Siebensegmentanzeigen so angepasst, dass die x-Koordinaten der gefundenen Objekte hexadezimal ausgegeben werden. Dabei kann über die Schalter SW[8] und SW[9] des FPGA-Boards eingestellt werden, ob die Start-, Mittel- oder Endpunkte der gefundenen Objekte ausgegeben werden sollen. Der Test ergab, dass der Funktionsblock LINE\_DETECTION\_2 bei einer eingestellten Anzahl von drei zu findenden Objekten in der Lage ist, diese zu finden und deren x-Koordinaten auszugeben, wie es in Abbildung 50 a und b gezeigt wird. Zur Überprüfung der Korrektheit der x-Koordinaten wurde wie im Kapitel 6.4 mithilfe des Funktionsblocks ISOLATED\_POINT\_GENERATOR\_2 eine Linie mit bekannten x-Koordinaten erzeugt. Diese wurden vom LINE\_DETECTION\_2 Funktionsblock korrekt erkannt und ausgegeben.

## 6.6 Funktionen von Tastern und Schaltern

In den vorherigen Unterkapiteln wurde teilweise auf die Funktion bzw. Bedeutung der Taster des FPGA-Boards eingegangen. Diese sind hier nochmals für die Taster KEY[0] bis KEY[3] in der Tabelle 1 beschrieben.

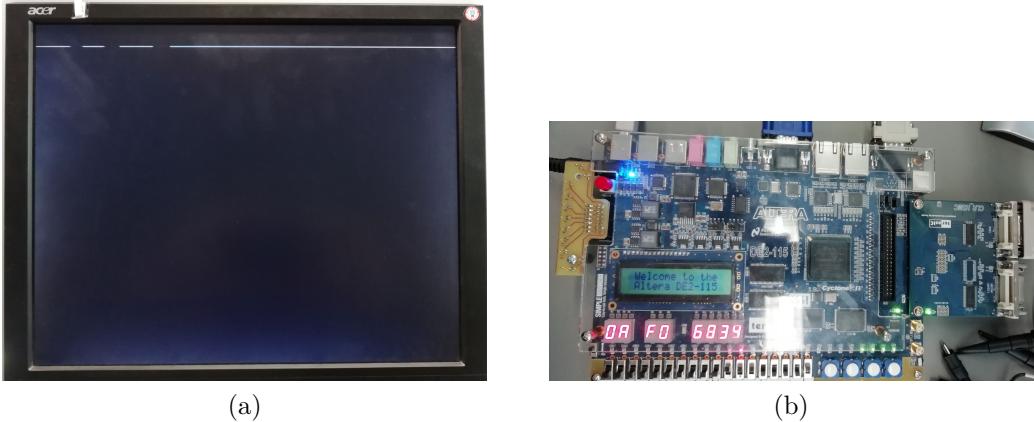


Abbildung 50: Darstellung mehrerer Linienobjekte in einer Bildzeile (a) und Ausgabe der x-Koordinaten des Beginns der Objekte auf dem FPGA-Board (b)

Tabelle 1: Funktionsbeschreibung der Taster des FPGA-Boards

Taster	Funktion
KEY[0]	Reset
KEY[1]	Einzelaufnahme camera_data_mux_gen
KEY[2]	Zeile inkrementieren, die der SINGLE_LINE_READ_BUFFER aus dem SDRAM lesen soll
KEY[3]	Änderung der Pixelintensität GREY_LEVEL_GENERATORS

Die beschriebenen Funktionsblöcke in den vorherigen Unterkapiteln haben alle die Funktion, die x-Koordinaten ihrer gefundenen Objekte sowie das Faltungsergebnis auszugeben. Deshalb wurde eine Ausgangslogik entworfen, die es ermöglicht, über die Schalter des FPGA-Boards auszuwählen, welche Daten auf den Siebensegmentanzeigen dargestellt werden sollen. In Abbildung 51 ist ein Teil dieser Ausgangslogik dargestellt. Zudem wird in der Tabelle 2 beschrieben, welche Funktion die einzelnen Schalter haben.

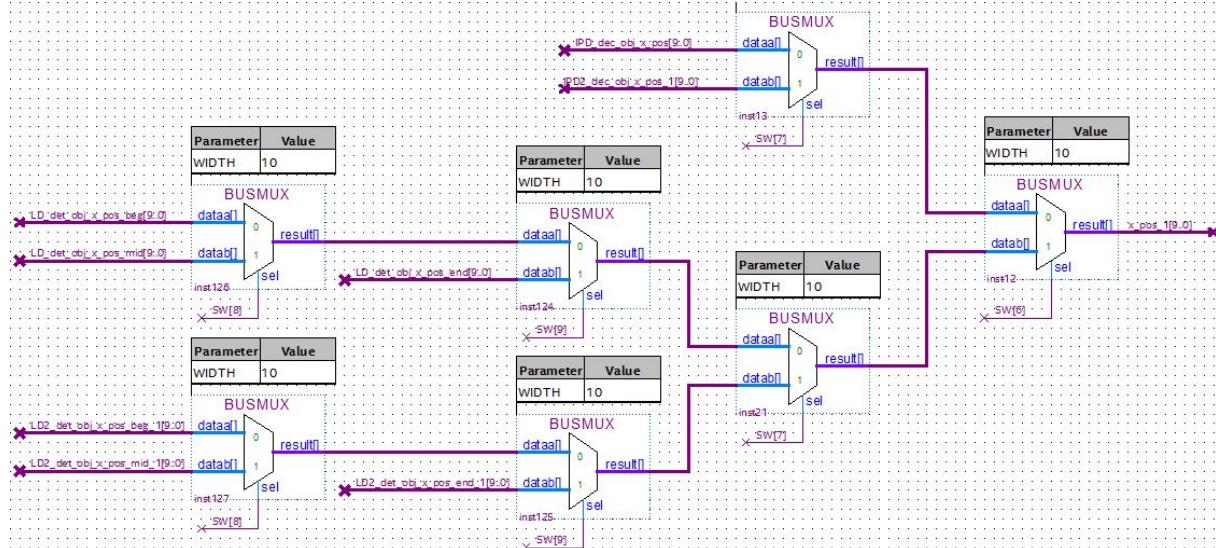


Abbildung 51: Teilausschnitt der Logik für die Ausgabe über die Siebensegmentanzeigen

Tabelle 2: Beschreibung der Einstellmöglichkeiten über die Schalter des FPGA-Boards

Schaltern	Funktion
SW[0]	Wahl, ob Einzelaufnahme (1) oder kontinuierliche Verarbeitung von Kamerabildern (0) (camera_data_mux_gen)
SW[1]	—
SW[2]	Wahl, ob ein isolierter Punkt(0) oder drei isolierte Punkte (1)
SW[3]	Wahl, ob Ergebnis für 1 (0) oder 3 (1) Objekte auf Siebensegmentanzeige dargestellt werden soll
SW[4]	Wahl, ob originale extrahierte Zeile (0) oder ob manipulierte extrahierte Zeile (mit ISOLATED_POINT_GENERATOR) (1) über VGA ausgegeben werden soll
SW[5]	Wahl, ob ganzes Bild (0) extrahierte Zeile (1) über VGA ausgegeben werden soll
SW[6]	Wahl, ob Daten für isolierte Punkte (0) oder für Linienobjekte (1) ausgegeben werden sollen
SW[7]	Wahl, ob Daten von Funktionsblöcken, die nur ein Objekt finden können, (0) oder ob Daten von Funktionsblöcken, die mehrere Objekte finden können, (1) ausgegeben werden sollen
SW[8]	bei Linienobjekten wählen, ob x-Koordinate des Objektbeginns (0) oder des Objektmittelpunktes (1) ausgegeben werden soll
SW[9]	bei Linienobjekten wählen, ob x-Koordinate des Objektbeginns/-mittelpunktes (über SW[8] einstellbar) (0) oder des Objektendpunktes (1) ausgegeben werden soll
SW[10]	Wahl, ob x-Koordinate (0) oder Faltungsergebnis (1) auf den Siebensegmentanzeigen ausgegeben werden soll
SW[11]	—
SW[12]	—
SW[13]	—
SW[14]	—
SW[15]	—
SW[16]	—
SW[17]	—

## 7 Bildwiederholungsrate

Nach der eindimensionalen Positionserkennung von Objekten folgte die Erhöhung der Bildwiederholungsrate. Hierbei war es das Ziel, durch die Erhöhung der Wiederholungsrate mehr Bilder pro Sekunde zu generieren, wodurch der Positions- und Objekterkennung mehr Bilder bei der Verarbeitung zur Verfügung stehen.

Dabei wurde als erstes mittels des Programms CLCrl2 die Zeilenanzahl eines Bildes verringert, um die Wiederholungsrate zu erhöhen. Anschließend erfolgte die Anpassung der Bildausgabe über VGA.

### 7.1 Maximale Bildwiederholungsrate

Damit die Erhöhung der Bildwiederholungsrate möglich ist, muss die Kamera umparametert werden. Dies geschieht über das Programm CLCrl2, mit dem es möglich ist, die eingestellten Parameter der Kamera zu lesen und zu verändern.

Über den Button *Read all* auf der Startseite des Programms können alle Parameter der Kamera, die im Electrically Erasable Programmable Read Only Memory (EEPROM) gespeichert sind, abgerufen werden. Über den in Abbildung 52 gezeigten Writer ist es möglich, die Anzahl der auszugebenden Zeilen anzupassen sowie mehrere Teilbilder von einer bestimmten bis zu einer weiteren vorgebbaren Zeile auszugeben.

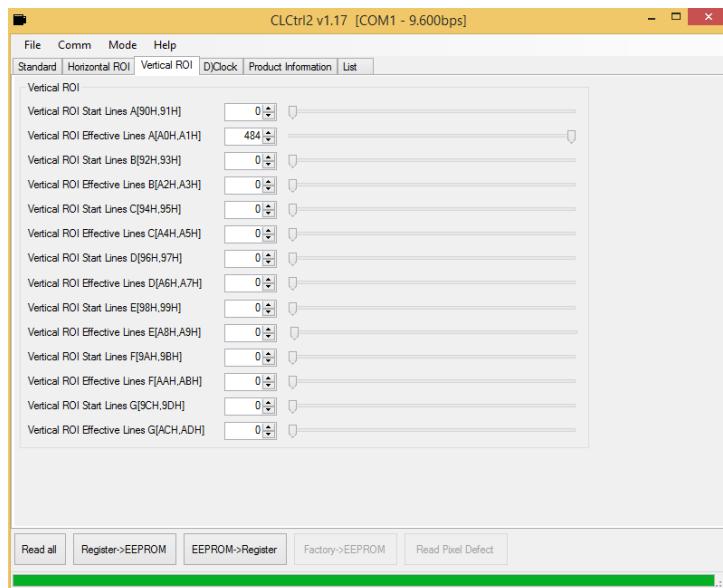


Abbildung 52: CLCrl2\_Vertical\_ROI

Um eine Bildausgabe von der ersten Bildzeile bis zu einer bestimmten Zeile zu erhalten, muss die Eingabe bei Vertical ROI Start Lines A[90H,91H] 0 lauten, damit das ausgebene Bild in der ersten Zeile des von der Kamera aufgenommenen Bildes startet. Die Zeile, bis zu der das Kamerabild ausgegeben werden soll, wird über die Eingabe bei Vertical ROI Effectiv Lines A[A0H,A1H] vorgegeben. In Tabelle 3 sind für verschiedene Anzahl an Zeilen die dazugehörigen Bildwiederholungsfrequenzen und FPS angegeben. Die Ermittlung der Frequenz erfolgte durch das Oszilloskop TDS 3014C 100MHz der Firma Tektronix, mit dem das Signal FVAL der Kamera aufgenommen und dessen

Frequenz ermittelt wurde.

Tabelle 3: Erhöhung der Bildwiederholungsrate mit der Anzahl der Bildzeilen und der dazugehörigen Bildwiederholungsfrequenz

Blindzeilen	Bildzeilen	Frequenz [kHz]	FPS
14	484	0,48	480
14	242	0,841	841
14	201	1,002	1002
14	94	1,995	1995
14	58	2,992	2992
14	40	3,990	3990
14	29	5,010	5010
14	22	5,983	5983
14	17	6,948	6948
14	13	7,979	7979
14	10	8,975	8975
14	8	9,790	9790
14	4	11,97	11970
14	3	12,67	21670

Das Standardbild mit 484 Bildzeilen hat eine Bildwiederholungsrate von 0,48 kHz, was 480 FPS entspricht. Eine Halbierung der Bildzeilen auf 242 Zeilen hat eine Erhöhung auf eine Frequenz von 0,841 kHz (841 FPS) zur Folge. Eine Erhöhung der Bildwiederholungsrate ist bis auf 12,67 kHz möglich, bei der noch drei Bildzeilen ausgegeben werden.

Durch die Erhöhung der Bildwiederholungsrate werden zwar mehr Bilder aufgenommen, allerdings kommt es dadurch zu Bilddopplungen bzw. Mehrfachausgabe eines Bildes über die VGA-Ausgabe auf dem Monitor, da die Hardware nicht daran angepasst ist. Zudem kommt es zu dunkleren Bildern, da die Beleuchtungszeit kürzer wird.

## 7.2 Bilddopplung

Durch die beschriebene Erhöhung der Bildwiederholungsrate kommt es zu einer Doppelung bzw. Mehrfachausgabe eines Bildes, wie es in Abbildung 53 a und b gezeigt wird. Aufgrund dessen muss eine Anpassung an den Parametern der Funktionsblöcke vorgenommen werden, damit es zu keiner Bilddopplung mehr kommt.

Der Grund der Bilddopplung bzw. Mehrfachausgabe durch eine erhöhte Bildwiederholungsrate liegt darin, dass die meisten Funktionsblöcke mit Parametern für die Größe eines Bildes arbeiten, die auf die Größe des Monitors mit 640x480 Pixeln ausgelegt sind, so auch die Funktionsblöcke `camara_data_mux_gen` und `SDRAM_Write_Buffer_gen`. Deren Parameter `RES_WIDTH` und `RES_HEIGHT` sind auf die Werte 640 (`RES_WIDTH`) und 480 (`RES_HEIGHT`) festgelegt. Dadurch erfolgt die Speicherung eines Bildes mit 242 Bildzeilen als Bildhöhe doppelt, was in etwa einer verdoppelten Bildwiederholungsrate entspricht, weil das erste Bild in den berechneten Speicheradressen für die ersten 242 Bildzeilen eines 480 Zeilen großen Bildes erfolgt. Anschließend werden die Daten der 14



Abbildung 53: Bilddopplung (a) bzw. Mehrfachausgabe eines Bildes (a) durch Erhöhung der Bildwiederholungsrate auf 0,841 kHz und 1,995 kHz



Abbildung 54: Erhöhung der Bildwiederholungsrate auf 0,841 kHz und 1,995 kHz ohne Bilddopplung (a) bzw. Mehrfachausgabe (b)

Blindzeilen zwischen zwei Kamerabildern in den Speicheradressen für die nächsten 14 Bildzeilen abgelegt. Danach wird das nächste aufgenommene Bild in den Speicheradressen für die Zeilen 257 bis 480 des 480 Bildzeilen großen Bildes gespeichert, wobei die letzten 16 Zeilen des 242 Zeilenbildes aufgrund der 14 gespeicherten Blindzeilen verloren gehen, da sie nicht mehr gespeichert werden. Erst bei Erreichen der Speicheradresse für die Bildzeile 480 und der Aufnahme des nächsten Bildes wird dieses wieder beginnend an der Speicheradresse für Zeile 0 abgelegt. Dadurch wird beim Auslesen des Speichers und der Ausgabe über VGA das Bild, wie es in Abbildung 53 a zu sehen ist, ausgegeben. Wird die Bildwiederholungsrate noch weiter erhöht, z.B. so, dass ein Bild nur noch 94 Zeilen hat, werden mehrere Bilder in aufeinanderfolgenden Speicheradressen abgelegt, statt dass ein neues Bild das alte überschreibt, wodurch es zur in Abbildung 53 b dargestellten Mehrfachausgabe eines Bildes kommt.

Um das Problem zu lösen, wurde an allen Funktionsblöcken, die mit dem Parameter der *RES\_HEIGHT* arbeiten (außer dem Funktionsblock *vga\_controller*) dieser Parameter immer auf den Wert eingestellt, mit dem auch das Kamerabild parametriert wurde. Bei einem Kamerabild mit 240 Bildzeilen wurde der Parameter *RES\_HEIGHT*

somit auf den Wert 240 gesetzt; bei einer Bildhöhe von 94 auf den Wert 94. Dadurch wird ein neues Bild immer beginnend an der Speicheradresse gespeichert und auch nur bis zur Speicheradresse der letzten Bildzeile ausgelesen und es kommt zu keinen Bilddopplungen bei der VGA-Ausgabe mehr, wie es in Abbildung 54 a und b zu sehen ist. Die Zeilen am Monitor, die nicht mehr zum Bild gehören, werden schwarz ausgegeben.

### 7.3 Belichtungszeit

Durch die Erhöhung der Bildwiederholungsrate kommt es neben dem Effekt der Bilddopplung noch zu einem weiteren Effekt. Durch die höhere Bildfrequenz wird die Belichtungszeit für die einzelnen Bilder kürzer. Dies hat zur Folge, dass die Bilder dunkler werden, wodurch die Positions- und Objekterkennung erschwert wird, da die Intensitätsunterschiede geringer werden.

Dass die Kamerabilder bei einer erhöhten Bildwiederholungsrate dunkler werden, ist schon beim Vergleich der Bilder in Abbildung 54 a und b zu erkennen. Dabei ist Bild a mit einer Bildfrequenz von 0,841 kHz und Bild b mit einer Bildfrequenz von 1,995 kHz aufgenommen worden. Mithilfe einer zusätzlichen Lichtquelle kann diesem Problem entgegengewirkt werden. Diese zusätzliche Lichtquelle hat zur Folge, dass die aufgenommenen Kamerabilder heller werden, was für eine bessere Positions- und Objekterkennung sorgt.

## 8 Ergebnis und Diskussion

Im nachfolgenden Teil erfolgt die Beschreibung der erzielten Ergebnisse und die Diskussion dieser. Dabei wird zuerst auf die Extraktion und Ausgabe einer Bildzeile über den VGA-Ausgang auf einen Monitor eingegangen. Anschließend wird die Erkennung isolierter Punkte in einem eindimensionalen Bild, die Erkennung von Linien in einem eindimensionalen Bild und die Erhöhung der Bildwiederholungsrate betrachtet.

### 8.1 Ausgabe einer Bildzeile

Um die eindimensionale Positionserkennung zu ermöglichen, musste zuerst ein eindimensionales Bild erzeugt werden. Dies wurde über den in Kapitel 6.1 entworfenen Funktionsblock SINGEL\_LINE\_READ\_BUFFER ermöglicht. Mithilfe dieses Funktionsblockes, angepasster Beschaltung der Eingänge und der Ausgangslogik für die VGA-Ausgabe ist es möglich, die gewünschte Zeile aus einem Bild zu extrahieren und damit ein eindimensionales Bild zu erzeugen, das über VGA auf einen angeschlossenen Monitor ausgegeben werden kann.

Da der Funktionsblock SINGLE\_LINE\_READ\_BUFFER auf dem bereits existierenden Funktionsblock SDRAM\_Read\_Buffer\_gen beruht, wurden an diesem Funktionsblock lediglich kleine Anpassungen vorgenommen. So wurden ein paar Signalnamen geändert und die Zustandsmaschine rd\_state zum Lesen der Pixeldaten der zu extrahierenden Bildzeile leicht abgeändert.

Die Ausgabe einer extrahierten Bildzeile auf einen Monitor über VGA funktioniert zuverlässig und die Zeile wird auch in der richtigen Zeile des Bildmonitors ausgegeben, d.h. wenn die vierte Bildzeile des Kamerabildes ausgegeben werden soll, wird diese auch in der vierten Zeile des Monitors dargestellt. Die Beschaltung des Einganges wr\_en des Funktionsblocks, über den die Ausgabe der Bildzeile gestartet wird, ist recht aufwändig realisiert worden. So wird über die Signale h\_sync und v\_sync des vga\_controllers im Funktionsblock CTU\_VGA\_V3\_state\_machine die Zeile mitgezählt, in der sich die VGA-Ausgabe gerade befindet. Über den Funktionsblock COMPARATOR erfolgt der Vergleich zwischen der Zeile, in der sich die VGA-Ausgabe befindet, mit der Zeile, die extrahiert werden soll. Wenn die VGA-Ausgabe die extrahierte Zeile erreicht, wird diese Zeile ausgegeben, sobald das Signal disp\_ena des vga\_controllers '1' ist. Wie bereits erwähnt, ist dieser Lösungsweg ein sehr aufwändiger Weg, funktioniert aber gut und liefert das gewünschte Ergebnis.

### 8.2 Erkennung isolierter Punkte in einem eindimensionalen Bild

Die Erkennung von isolierten Punkten in einer Zeile erfolgt über die Funktionsblöcke ISOLATED\_POINT\_DETECTION und ISOLATED\_POINT\_DETECTION\_2, wobei der Unterschied dieser beiden Funktionsblöcke in der Anzahl der Objekte, die gefunden werden können, liegt. Die Positions- und Objekterkennung erfolgt über das Mitlesen der extrahierten Bildzeile des SINGLE\_LINE\_READ\_BUFFERS, die über VGA ausgegeben wird. Diesen Daten werden über eine Zustandsmaschine aufgenommen und mittels der 2. Ableitung der Funktion  $f(x)$  wird die Faltungssumme für jedes Pixel berechnet, wie es in Kapitel 2.3 beschrieben ist. Über einen Schwellwertvergleich wird entschieden, ob ein isolierter Punkt vorliegt oder nicht und bei einem Treffer wird die x-Koordinate des isolierten Punktes ausgegeben.

Das Einlesen der Daten, die Berechnung der Faltungssumme, die Auswertung über einen Schwellwert zur Objekterkennung als auch die Ausgabe der x-Koordinate bei einem gefundenen Objekt funktioniert wie erwünscht. So wird ein Pixel als isolierter Punkt erkannt, wenn die Faltungssumme den Schwellwert überschreitet und die x-Koordinate wird korrekt ausgegeben, wie es in den Kapiteln 6.2 und 6.3 beschrieben ist.

Als problematisch hat sich herausgestellt, dass diese beiden Funktionsblöcke den Anfang bzw. die Kanten einer Linie als isolierten Punkt erkennen können, wenn der Schwellwert unter einen Wert von 256 eingestellt wird und die Pixel der Linie beispielsweise schwarz sind, also den Wert 0 aufweisen, und die Pixel des Hintergrundes weiß sind, also den Wert 255 haben. Dann ergibt sich bei der Berechnung der Faltungssumme an dieser Stelle der Wert 255 und die Kante der Linie wird als Objekt erkannt. Dies ist allerdings kein isolierter Punkt, da sich das Pixel in seiner Intensität nicht von den beiden benachbarten Pixeln abhebt, weshalb es nicht als isolierter Punkt erkannt werden sollte. Um dies auszuschließen, darf der Schwellwert nicht unter 256 eingestellt werden. Alternativ müsste über die Zustandsmaschine eine weitere Auswertung erfolgen, die erkennt, ob wirklich ein isolierter Punkt vorliegt oder ob es sich um eine Kante einer Linie bzw. einen langsamen Intensitätswechsel handelt.

Da sowohl in dem Funktionsblock ISOLATED\_POINT\_DETECTION als auch im Funktionsblock ISOLATED\_POINT\_DETECTION\_2 die Faltung jedes Pixels einer Bildzeile mit der Gewichtungsmatrix, die aus der 2. Ableitung ableitbar ist, berechnet wird, wäre es eine Möglichkeit, die Berechnung in einem eigenen Funktionsblock statt dies in beiden Blöcken durchzuführen. Damit stünde das Ergebnis der Faltung für alle Funktionsblöcke zur Verfügung, würde nicht doppelt bzw. mehrfach berechnet, wenn diese Funktionsblöcke öfters eingesetzt und in der selben Zeile nach Objekten mit unterschiedlichen Schwellwerten suchen. Außerdem könnten Hardwareressourcen im FPGA gespart werden.

### 8.3 Erkennung von Linienobjekten in einem eindimensionalen Bild

Zur Realisierung der Positions- und Objekterkennung von Linien in einem eindimensionalen Bild wurden die beiden Funktionsblöcke LINE\_DETECTION und LINE\_DETECTION\_2 entworfen. Beide haben die gleiche Aufgabe, unterscheiden sich aber in der Anzahl an Objekten, die sie finden können. So kann der Line\_DETECTION-Funktionsblock nur die erste Linie in einer Zeile erkennen, der LINE\_DETECTION\_2 hingegen kann bis zu drei Linienobjekte in einer Bildzeile erkennen. Beide Funktionsblöcke lesen dabei die Werte der Pixel über eine Zustandsmaschine mit, die der SINGEL\_LINE\_READ\_BUFFER für die Ausgabe über VGA ausgibt. Zudem wird die Faltungssumme wie in Kapitel 2.3 beschrieben für jedes Pixel mithilfe der 1. Ableitung der Funktion  $f(x)$  berechnet und über einen Vergleich der Faltungssumme mit einem vorgegebenen Schwellwert der Anfang und das Ende einer Linie gesucht. Wenn eine Linie gefunden wurde, werden die x-Koordinaten für den Anfangs-, Mittel- und Endpunkt der Linie ausgegeben.

Die Erkennung von Linien in einer Zeile funktioniert, wobei in Abhängigkeit vom eingesetzten Schwellwert sich die Linie in ihrer Intensität stärker vom Hintergrund abheben muss oder nicht. Momentan können nur Linien erkannt werden, wenn der Hintergrund hell ist und Linien eine dunklere Farbe aufweisen, die ausreicht, um bei der Berechnung der Faltungssumme und dem anschließenden Vergleich mit dem Schwellwert diesen zu über-

schreiten. Um Linien allgemein besser erkennen zu können, also helle Linien auf einen dunklen Hintergrund, müsste die Zustandsmaschine dementsprechend erweitert und angepasst werden.

Zudem werden isolierte Punkte als Linienobjekte erkannt, was nicht unbedingt erwünscht ist, da dies keine Linien sind. Dies geschieht, da der Intensitätswechsel vom weißen Hintergrund zu einem schwarzen isolierten Punkt als Anfang einer Linie und der Wechsel von einem schwarzen isolierten Punkt zum weißen Hintergrund als Ende einer Linie erkannt werden. Damit dies nicht geschieht, müsste in der Zustandsmaschine eine weitere Überprüfung stattfinden, ob es sich um den Beginn einer Linie oder einen isolierten Punkt handelt.

Eine Maßnahme zur besseren Ausnutzung der Hardwareressourcen, wäre die Durchführung der Berechnung der Faltung einer Bildzeile mithilfe der ersten Ableitung in einem eigenen Funktionsblock. Dadurch erfolgt die Berechnung nicht doppelt oder mehrfach, wenn mehrere Funktionsblöcke LINE\_DETECTION oder LINE\_DETECTION\_2 genutzt werden, die in der selben Zeile nach Linienobjekten suchen. Zudem stünde das Faltungsergebnis allen Funktionsblöcken zur Verfügung.

## 8.4 Erhöhung der Bildwiederholungsrate

Die Erhöhung der Bildwiederholungsrate ist durch Änderungen der Parametrierung der Kamera möglich. Dafür wird in der Kamera der Parameter Vertical ROI Start Lines A[90H,91H] auf 0 gestellt und der Parameter Vertical ROI Effectiv Lines A[A0H,A1H] auf den gewünschten Wert, wie viele Zeilen ein Bild haben soll. Dadurch sind Bildwiederholungsraten bis zu 12,67 kHz möglich, was der einer Bildgröße von nur noch drei Zeilen entspricht.

Damit es durch die Erhöhung der Bildwiederholungsrate nicht zu dem in Kapitel 7.2 beschriebenen Effekt der Bilddopplung kommt, wurde eine recht simple Lösung gefunden. Bei allen Funktionsblöcken (außer dem vga\_controller), die mit der Höhe eines Bildes arbeiten, wird der Wert des Parameters RES\_HEIGHT auf den Wert eingestellt, der der Anzahl der Bildzeilen entspricht, die das Kamerabild aufweist. Dies ist wie bereits beschrieben eine recht einfache Lösung, allerdings muss jedesmal, wenn die Größe eines Bildes geändert wird, der Parameter RES\_HEIGHT an allen Funktionsblöcken geändert, die abgeänderte Hardwarebeschreibung neu compiliert und übertragen werden.

Zudem musste eine Lösung für das Problem der in Kapitel 7.3 gesunkenen Belichtungszeit bei erhöhter Bildwiederholungsrate und den daraus resultierenden dunkleren Bildern gefunden werden. Dieses Problem wurde mittels einer zusätzlichen Lichtquelle gelöst. Dabei ist allerdings zu beachten, dass eine Lichtquelle mit konstantem Lichtstrom genutzt wird. Dies ist nötig, da bei einer Lichtquelle, die keinen konstanten Lichtstrom aufweist, sich Hell- und Dunkelphasen abwechseln, wodurch die aufgenommenen Bilder unterschiedlich stark beleuchtet werden. Dadurch funktionieren die Funktionsblöcke zur Erkennung von isolierten Punkten und Linien nicht mehr ordnungsgemäß, da in den Dunkelphasen die Bilder eine geringe Intensität aufweisen, wodurch die Intensitätsunterschiede zwischen hellen und dunklen Stellen in einem Bild geringer werden. Durch diese geringeren Intensitätsunterschiede kann es dazu kommen, dass in den Hellphasen gefundene Objekte in den Dunkelphasen nicht mehr als Objekte erkannt werden.

## 9 Zusammenfassung und Ausblick

Ziel dieser Studienarbeit war es, aufbauend auf bereits vorhandenen Funktionen die Bilddatenverarbeitung in einem FGPA voranzutreiben. Dazu wurden zu der vorhandenen Funktion (Aufnahme, Speicherung und Ausgabe eines Kamerabildes über den VGA-Ausgang des FPGA-Boards) weitere Funktionen implementiert. Es erfolgte die Implementierung einer eindimensionalen Positionserkennung von Punkten und Linien sowie die Erhöhung der Bildwiederholungsrate. Genutzt wurden dabei die Kamera *STC-CMB33PCL* der Firma Semtech und das FPGA-Board *DE2-115* der Firma Terasic mit der Erweiterungskarte *CLR-HSMC Daughter Card*. Die eindimensionale Positions-erkennung beruht dabei auf Schwarzweißbildern, die von der Kamera aufgenommen werden. Die Erstellung dieser Funktionen erfolgte dabei mittels der Hardwarebeschreibungssprache VHDL und der Software Quartus Prime 18.1 Lite.

Um die eindimensionale Positions- und Objekterkennung zu implementieren, wurde ein Funktionsblock entwickelt, der eine bestimmte Zeile eines Bildes aus dem Speicher ausliest und über VGA auf einen Monitor ausgibt. Dazu wurde ein bereits existierender Funktionsblock, der die gespeicherten Bilddaten aus dem SDRAM des FPGA-Boards liest, in seiner Funktion an wenigen Stellen abgeändert, die Beschaltung der Eingangsschnittstellen angepasst, um die richtige Bildzeile aus dem Speicher zu lesen, und die Ausgangslogik für die VGA-Ausgabe überarbeitet, um zwischen der Ausgabe des gesamten Bildes und der Ausgabe einer Bildzeile umschalten zu können.

Danach wurde ein Funktionsblock entworfen, der in dieser Bildzeile nach einem isolierten Punkt sucht. Dazu wurde eine Zustandsmaschine genutzt, die die Intensitätswerte der Pixel einer Zeile einliest, mathematische Bildoperationen mittels Faltung der Pixelwerte mit einer Gewichtungsmatrix vornimmt und anhand der Ergebnisse der Bildoperationen über eine Auswertungslogik bestimmt, ob ein Objekt vorliegt. Ist ein Objekt gefunden worden, wird die Position dieses Objektes ausgegeben. Dieser Funktionsblock ist danach erweitert worden, sodass es möglich ist mehrere isolierte Punkte in einer Zeile zu finden. Anschließend wurde ein Funktionsblock geschrieben, der es ermöglicht, in einem eindimensionalen Bild eine Linie zu erkennen. Dabei werden über eine Zustandsmaschine die Pixelwerte einer Zeile eingelesen, mathematische Bildoperationen durchgeführt und das Ergebnis der Bildoperationen ausgewertet. Ist ein Objekt gefunden worden, wird die Position vom Anfangs-, Mittel- und Endpunkt des Linienobjektes ausgegeben. Danach wurde aufbauend auf diesem Funktionsblock ein weiterer entworfen, der es ermöglicht, mehr als ein Linienobjekt in einer Zeile zu erkennen.

Nach der Erkennung von Objekten in eindimensionalen Bildern erfolgte die Erhöhung der Bildwiederholungsrate. Um dies zu ermöglichen, mussten die Parameter der Kamera angepasst werden, die die Anzahl der Bildzeilen eines Kamerabildes festlegen. Durch eine Verringerung der Bildzeilenzahl wird ein Bild kleiner, wodurch sich die Frequenz der Bildwiederholung erhöht, da weniger Pixel pro Bild ausgegeben werden müssen. Ein Standardbild der Kamera hat 642 Spalten und 484 Zeilen und es werden dabei 480 Frames per second (FPS) ausgegeben. Wird nun die Anzahl der Bildzeilen verringert, erhöht sich die Bildwiederholungsrate. Ein Bild mit nur noch 242 Zeilen hat eine Bildwiederholungsfrequenz von 0,841 kHz (841 FPS). Die maximale Bildwiederholungsrate liegt bei 12,67 kHz (12670 FPS).

Aufbauend auf den erzielten Ergebnissen kann in weiteren Arbeiten die Entwicklung

der Bilddatenverarbeitung in einem FPGA vorangetrieben werden. So kann als nächste Schritte die Positions- und Objekterkennung in zweidimensionalen Bildern sowie die Verarbeitung von RGB-Farbbildern implementiert werden. Zudem kann anschließend eine Geschwindigkeit- und Beschleunigungsberechnung anhand der Positionsdaten von Objekten entwickelt werden.

## A Literatur

- [Gonzalez, R. C.; Woods, R. E., 2018] Gonzalez, Rafael C.; Woods, Richard E.: Digital Image Processing, 4. Auflage, Harlow: Pearson, 2018. -ISBN 13: 978-9-22304-9
- [Intel, 2019] Intel Corporation: Introducing the Intel® Quartus® Prime Pro and Standard Edition Software User Guides, URL: <https://www.intel.com/content/www/us/en/programmable/products/design-software/fpga-design/quartus-prime/user-guides.html>, Stand: 08.11.2019
- [Larson, 2018] Larson, Scott: VGA Controller (VHDL), 2018, URL: <https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278#space-menu-link-content>, Stand: 29.11.19
- [Mikrocontroller.net, o.J.] Mikrocontroller.net: VHDL, o.J., URL: <https://www.mikrocontroller.net/articles/VHDL>, Stand: 08.11.2019
- [microcontrollerslab.com, o.J.] microcontrollerslab.com: INTRODUCTION TO FIELD PROGRAMMABLE GATE ARRAYS (FPGA), o.J., URL: <https://microcontrollerslab.com/fpga-introduction-block-diagram/>, Stand: 08.12.19
- [Reichard, J.; Schwarz, B., 2015] Reichard, Jürgen; Schwarz, Bernd: VHDL-SYNTHESIZE. Entwurf digitaler Schaltungen und Systeme, 7. Auflage, Berlin: de Gruyter, 2015. -ISBN 13: 978-3-11-037505-3
- [Sentech, o.J.] Sentech Co., Ltd: CL CMOSOS High Speed VGA Product Specification, STC-CMB33PCL/CMC33PCL Specification and User guide, ohne Jahr
- [Stemmer, o.J.] STEMMER IMAGING AG: CameraLink, o.J., URL: <https://www.stemmer-imaging.com/de-de/grundlagen/cameralink/>, Stand: 04.01.2020
- [Terasic, 2010] Terasic Technologies: CLR-HSMC, User Manual Terasic CLR-HSMC Daugther Card, 2010
- [Terasic, 2012] Terasic Technologies: DE2-115 User Manuel, 2012
- [Tietze, U. et al., 2016] Tietze, Ulrich; Schenk, Christoph; Gamm, Eberhard: Halbleiter-schaltungstechnik, 15. Auflage, Berlin, Heidelberg: Springer Verlag, 2016. - ISBN-13: 978-3-662-48354-1
- [Upegui, 2006] Upegui, Andres: Dynamically Reconfigurable Bio-inspired, Lausanne, 2006, URL: [https://www.researchgate.net/figure/Programmable-Logic-Array-PLA\\_fig1\\_37437764](https://www.researchgate.net/figure/Programmable-Logic-Array-PLA_fig1_37437764) Stand 08.12.2019
- [Wikipedia, 2018] wikipedia.org: Camera Link, 2018, URL: [https://de.wikipedia.org/wiki/Camera\\_Link](https://de.wikipedia.org/wiki/Camera_Link), Stand : 04.01.2020