Übersicht Ist-Zustand

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

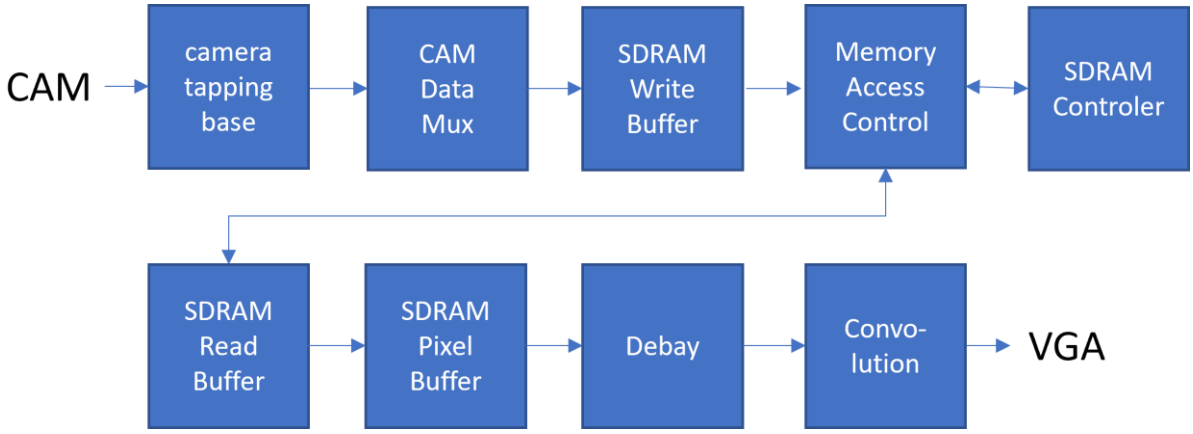


Abbildung 3: Übersicht der Module innerhalb des bisherigen Quartusprojekt

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

Abbildung 3 verfasst die Aufteilung der Module innerhalb des FPGA Bordes. Der Aufbau der vorgefundenen Hardware ist in Abbildung 4 zu sehen.

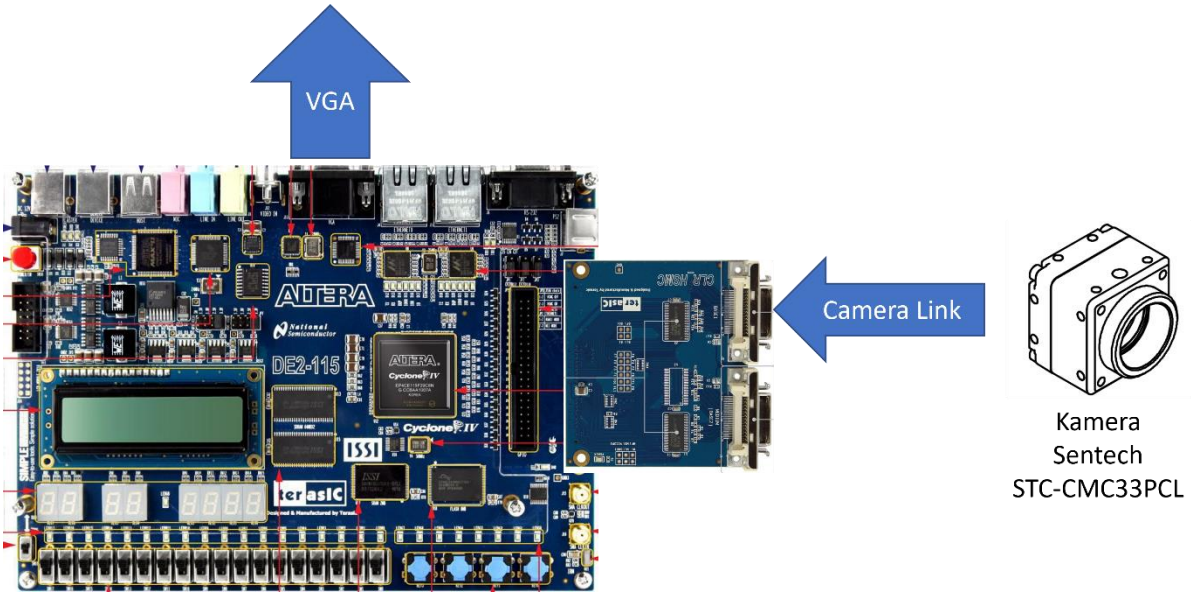


Abbildung 4: Übersicht verwendete Hardware

Modul: cameralink_tapping_base

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

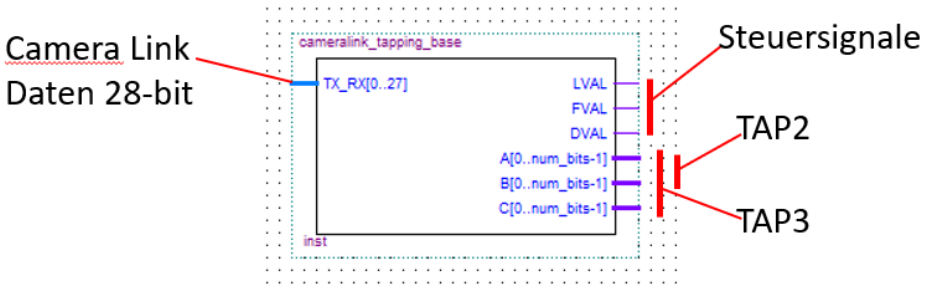


Abbildung 5: Blockschaltbild

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

Kameramodi: Taps, horizontales und vertikales Timing

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

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

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

Horizontales Timing 2 Tap

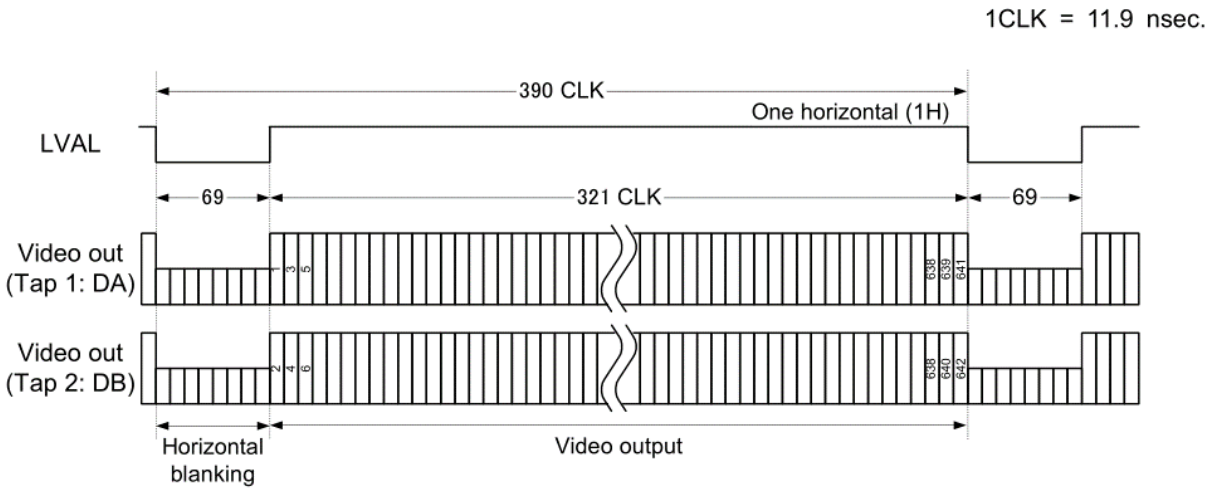


Abbildung 6: Übersicht Timing horizontal, Modi 2Tap, 642 Pixel horizontal

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

Diese Einstellung wird in der Kamera als „TAP Count“ eingestellt, oben beschrieben die Einstellung „TAP Count = 2Tap“. Darüber hinaus gibt es die Einstellung „3Tap“, welche die die Übertragung auf drei Kanäle aufteilt, jedoch nicht anders in der Funktionsweise zu „2Tap“ daher keine genauere Erläuterung dazu an dieser Stelle.

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

Vertikales Timing

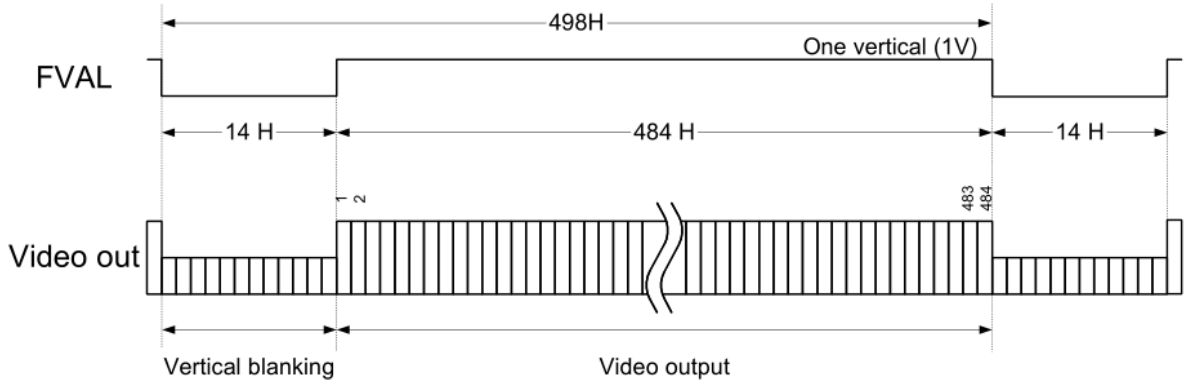


Abbildung 7: Übersicht Timing vertikal, 484 Zeilen vertikal

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

Modul: camera_data_mux_gen

Das Modul vereint die Signale „TAP1“ und „TAP2“ aus Abbildung 5: Blockschaltbild in einem 4 Byte Shift Register. Eine detaillierte Erklärung des Moduls kann in der Ausarbeit „Seite 19 camera_data_mux_gen, Bilddatenvorverarbeitung in einem FPGA, Lukas Herbst“ gefunden werden.

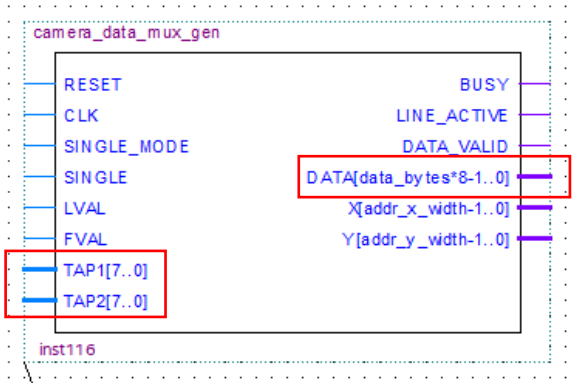


Abbildung 8: Blockschaltbild

Aufbau Intern: camera_data_mux_gen (Vereinfachte Version)

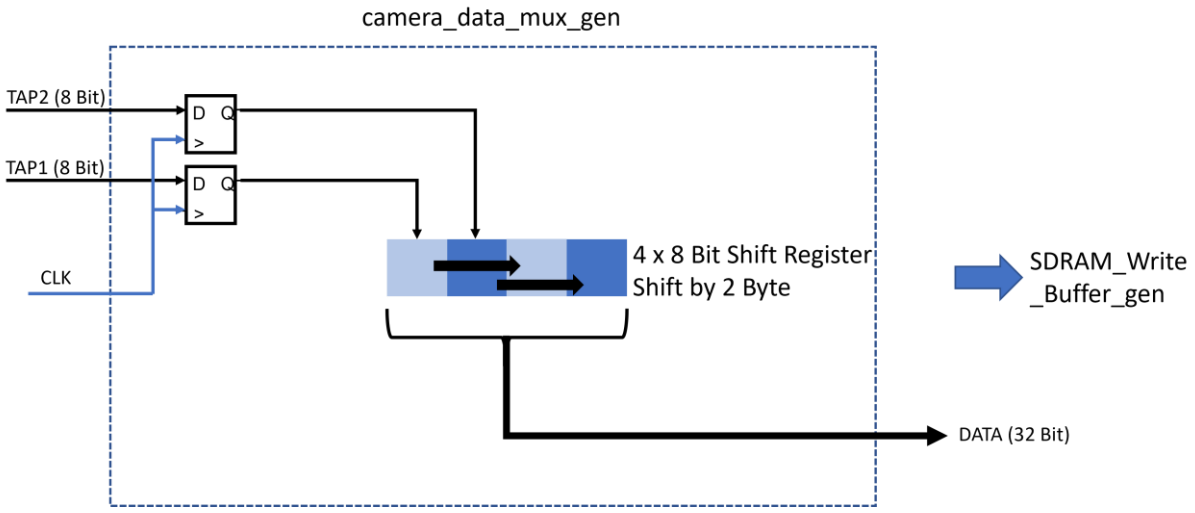


Abbildung 9: Vereinfachte Funktionsweise Modul „camera_data_mux_gen“

Modul: SDRAM_Write_Buffer_gen

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

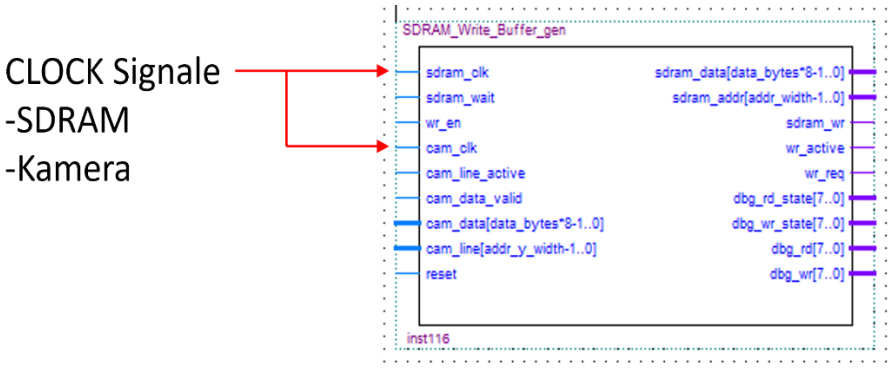


Abbildung 10: Blockschaltbild

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

Aufbau Intern: SDRAM_Write_Buffer_gen (Vereinfachte Version)

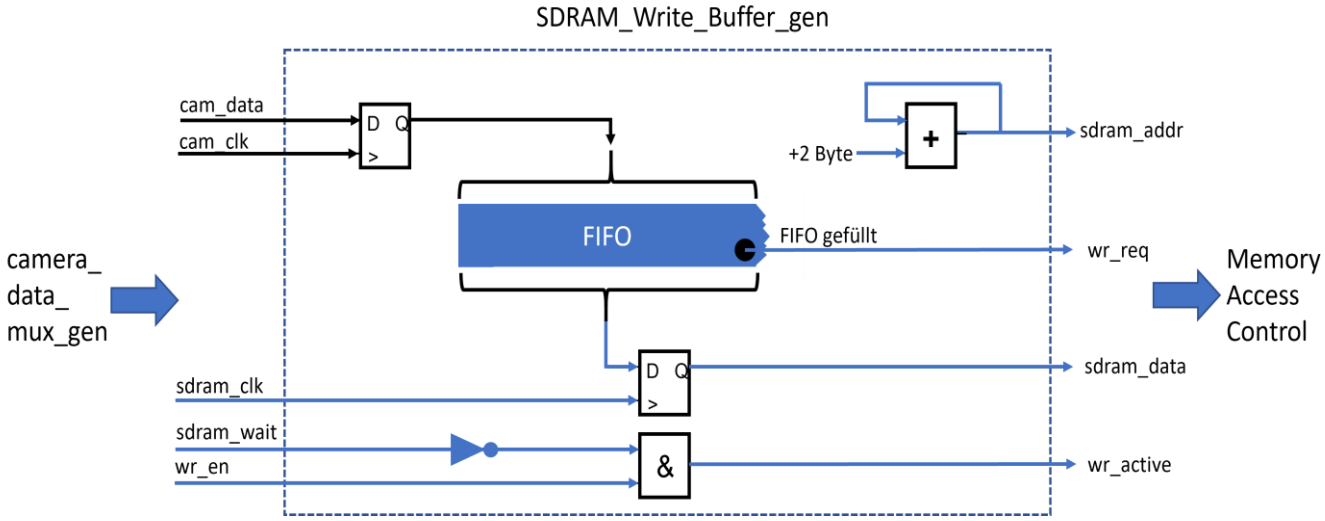
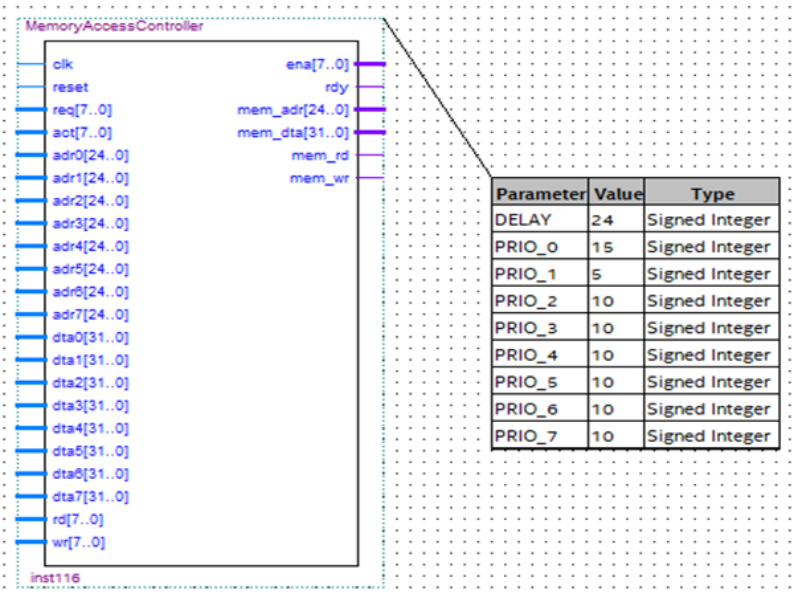


Abbildung 11: Vereinfachte Funktionsweise Modul „SDRAM_Write_Buffer_gen“

Modul: Memory Access Control

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



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

Abbildung 12: Blockschaltbild „MemoryAccessController“ mit Parametern

Das Interface selbst besteht aus den folgenden Signalen:

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

Funktionsweise Interface

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

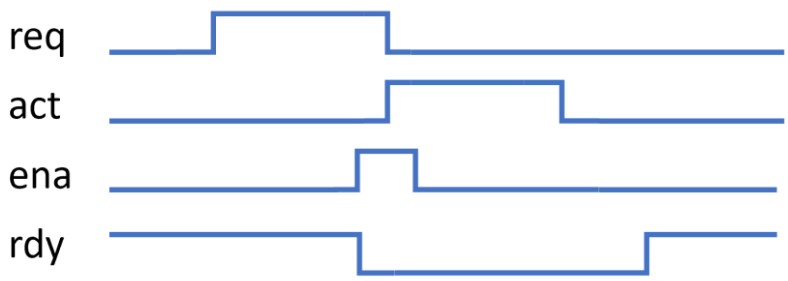


Abbildung 13: Funktionsweise Interface MemoryAccessController

Modul: SDRAM Controller

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

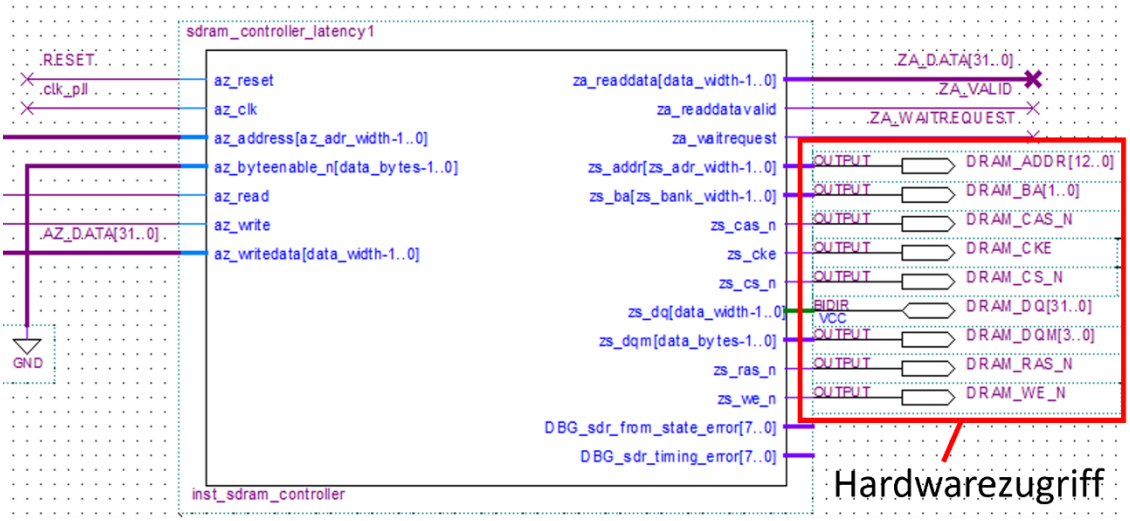


Abbildung 14: Blockschaltbild + Beschaltung

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

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

Modul: SDRAM_Read_Buffer_gen

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

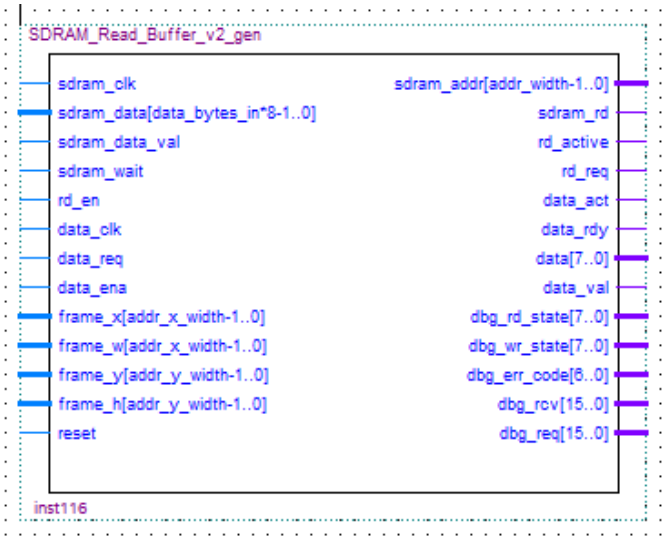


Abbildung 15: Blockschaltbild

Interne Buffer Struktur

Die SDRAM Daten werden in FIFO Buffern gespeichert. Um Daten den Datendurchsatz zu erhöhen werden Daten jeweils in einen Buffer gelesen und in den anderen geschrieben.

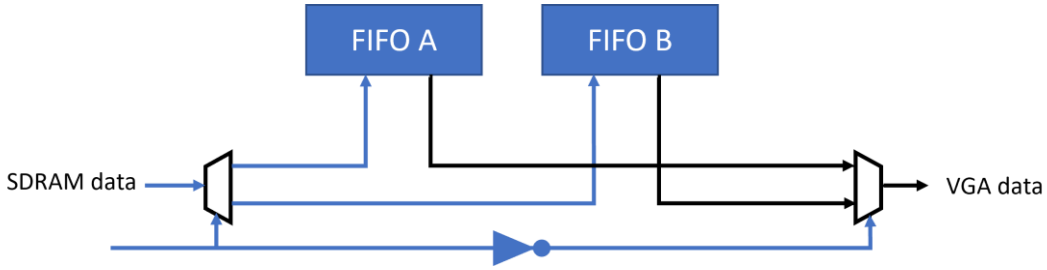


Abbildung 16: Interne Buffer Struktur des Moduls „SDRAM_Read_buffer_gen“

Modul: SDRAM_Pixelbuffer

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

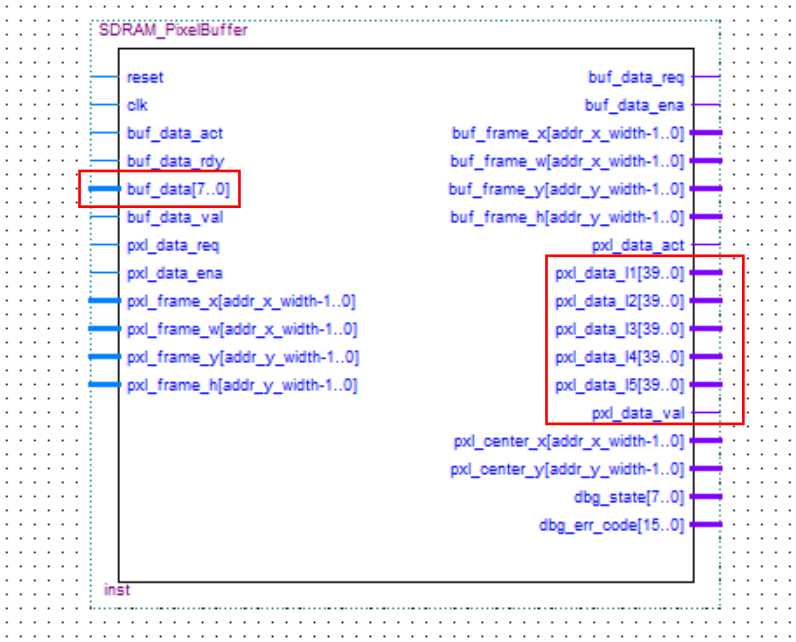


Abbildung 17: Blockschaltbild, rot markiert die wichtigsten IOs

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

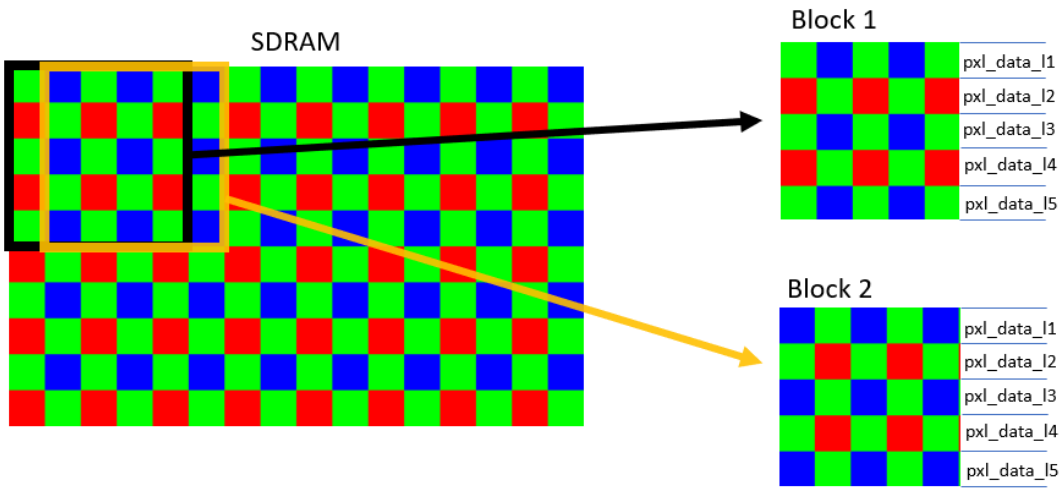


Abbildung 18: Schaubild Funktionsweise Pixelbuffer

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

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

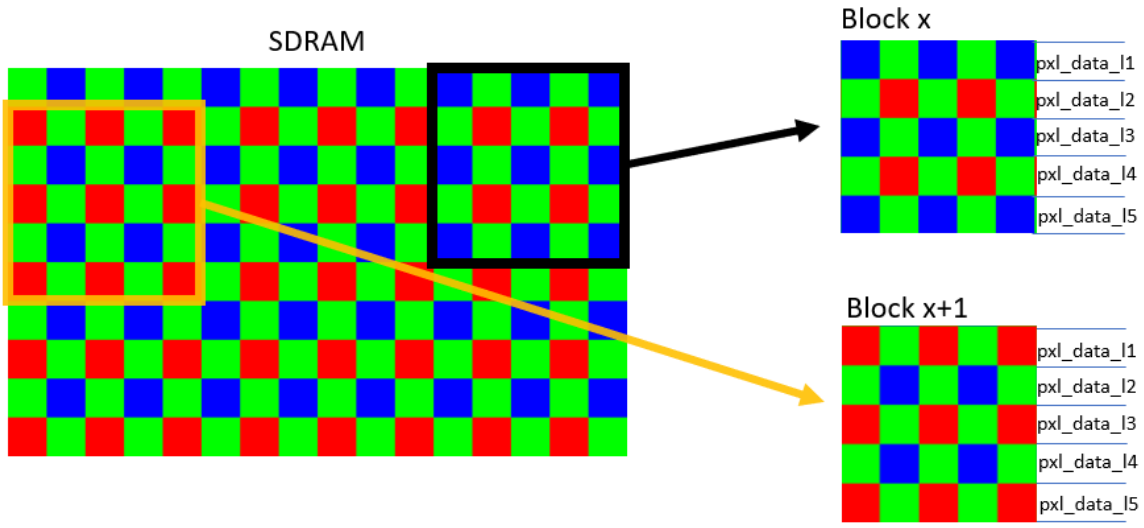


Abbildung 19 Schaubild Pixelbuffer Zeilenende

Aufbau Intern: SDRAM_Pixelbuffer(Vereinfachte Version)

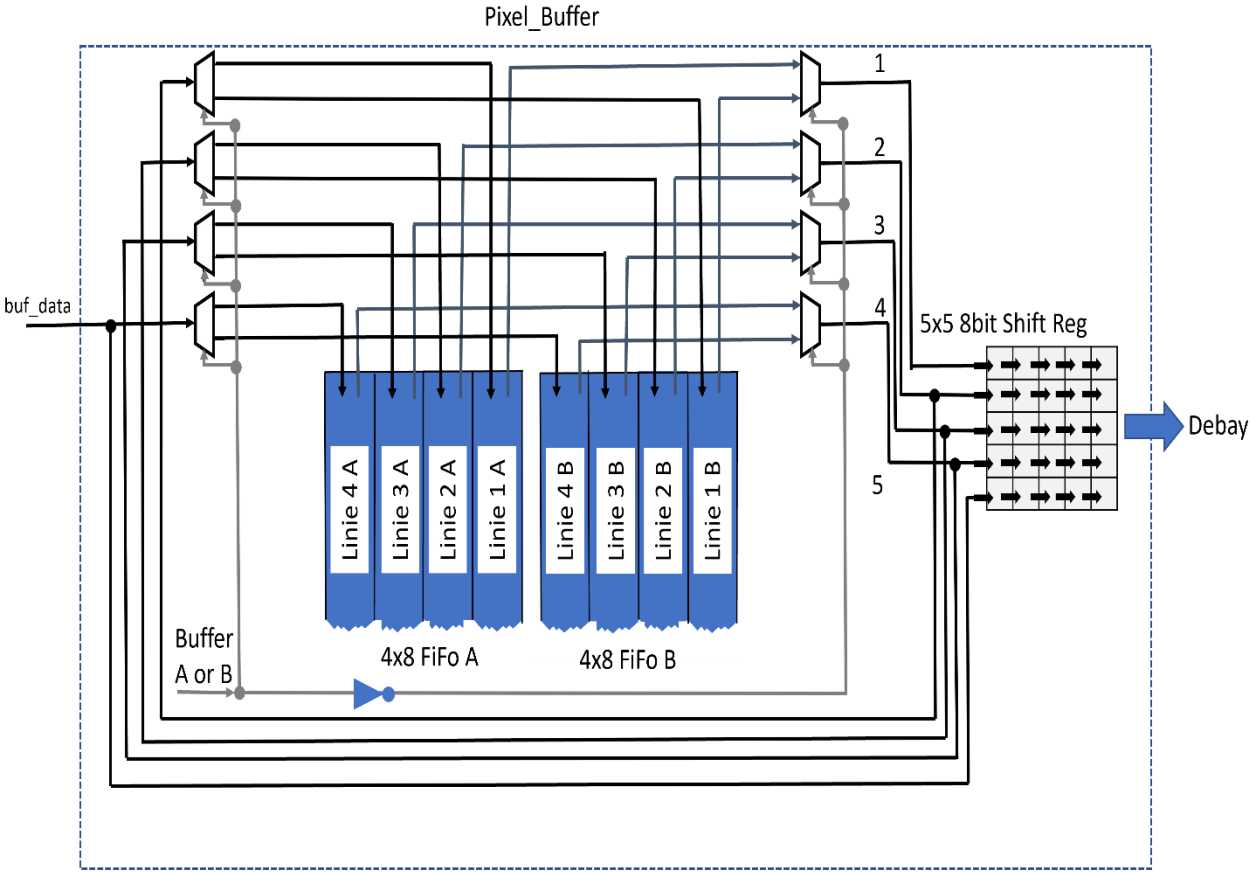


Abbildung 20 Schaubild interner Aufbau SDRAM_Pixelbuffer

Modul: Debay

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

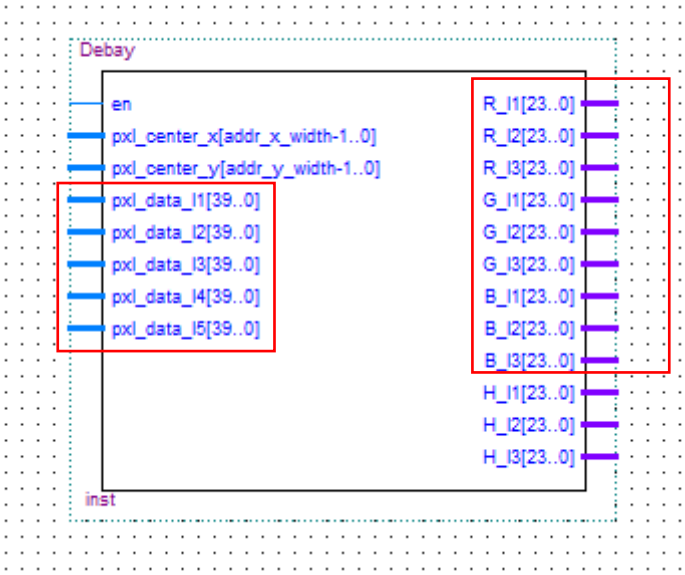


Abbildung 21 Blockschaltbild

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

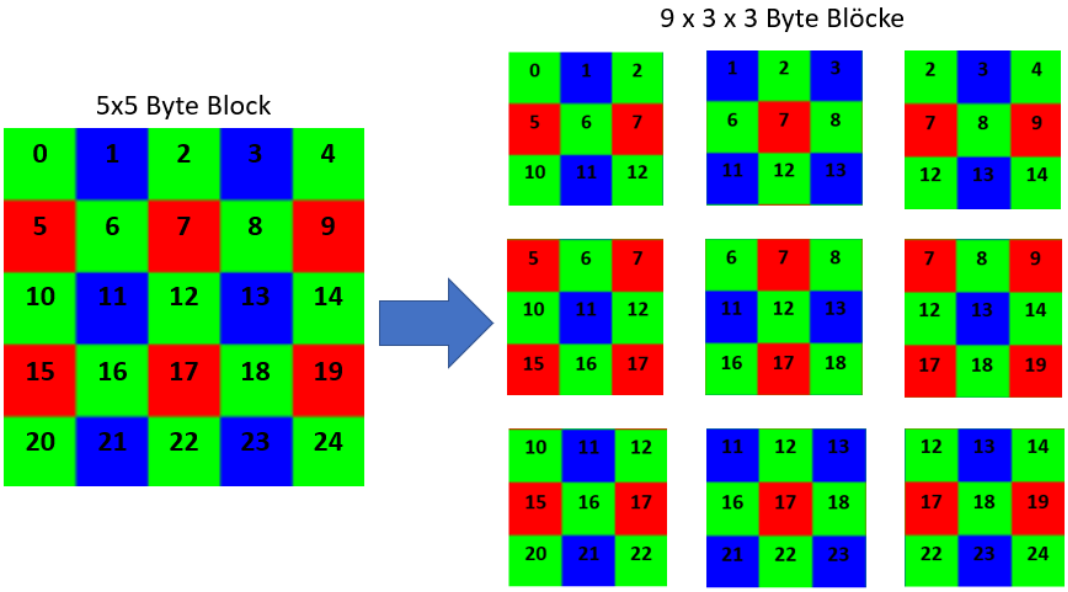


Abbildung 22 Aufteilung 5x5 Byte Block in kleinere 3x3 Blöcke

Durch die Aufteilung erhalten wir neun 3x3 Byte Blöcke. Aus jedem Block wird für alle drei Farbkanäle ein Byte extrahiert. Wie in Abbildung Debay 2 zu sehen, beinhaltet ein 3x3 Block mehr als nur eine Farbinformation für einen Kanal, so ist beispielsweise im ersten Block an Stelle 5 und 7 die Farbinformation für rot doppelt. Um aus zwei Byte ein zubekommen, nehmen wir das arithmetische Mittel der zwei Farbinformationen aus diesem Block. Zum besseren Verständnis dieser Vorgehensweise siehe Abbildung 23.

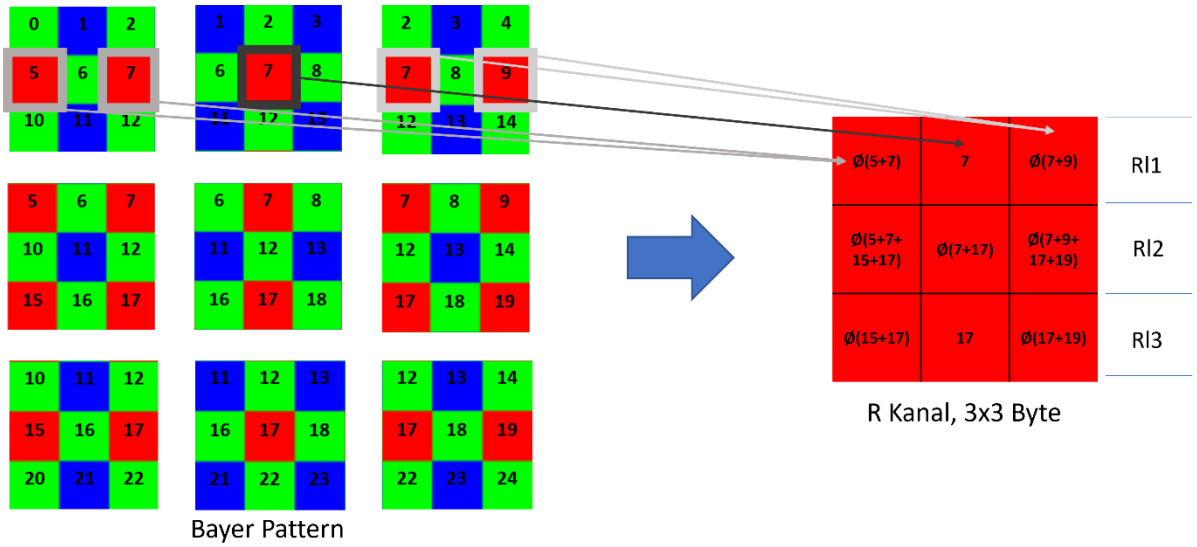
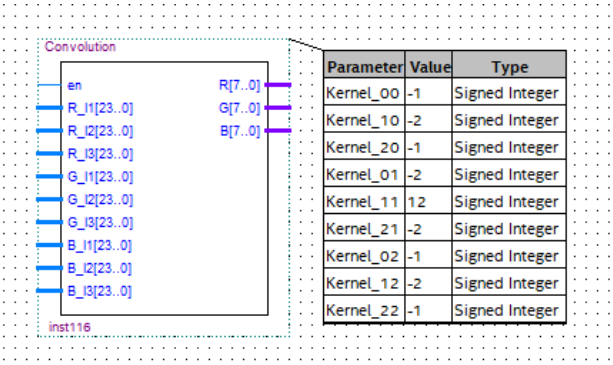


Abbildung 23: Zusammensetzung 3x3, Beispiel am Rotkanal, Rl1 = Output der ersten Zeile des Rotkanals

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

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

Modul: Convolution



Parameter	Value	Type
Kernel_00	-1	Signed Integer
Kernel_10	-2	Signed Integer
Kernel_20	-1	Signed Integer
Kernel_01	-2	Signed Integer
Kernel_11	12	Signed Integer
Kernel_21	-2	Signed Integer
Kernel_02	-1	Signed Integer
Kernel_12	-2	Signed Integer
Kernel_22	-1	Signed Integer

Abbildung 24: Blockschaltbild

„Convolution“ das bedeutet Faltung und ist der Hauptbestandteil der inneren Funktionsweise des Moduls in Abbildung 24. Die Eingangsgröße des Blocks ist die Ausgangsgröße zu sehen in Abbildung 23, es handelt sich um jeweils 3x3 Byte Blöcke für jeden Farbkanal (RGB). Aus diesem Byte Blöcken wird jeweils nur ein Farbwert gewonnen und auf die Ausgänge „R[7..0], G[7..0] und B[7..0]“ gelegt.

Das Ziel des Blocks ist es für jeden neuen Block einen RGB Pixelwert zu berechnen. Die Blöcke stammen aus dem ursprünglichen Bild und werden jeweils um **eine** Zeile in Ihrer Auswahl verschoben. Damit schlussendlich ein Bild mit der gleichen Anzahl an Pixel in Höhe und Breite zur Verfügung steht.

Für die Gewichtung der Farbanteile aus dem Byte Block stehen die Kernel Parameter 00-22 zur Verfügung.

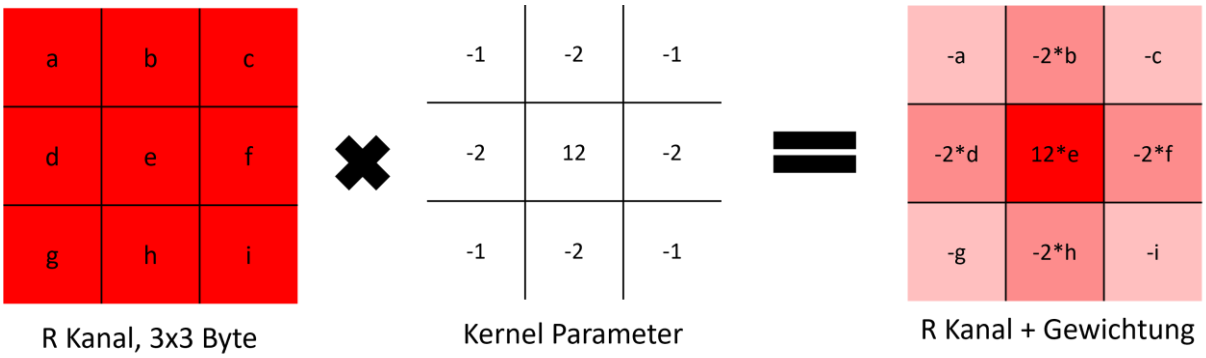


Abbildung 25: Gewichtung des 3x3 Rot Kanal für Berechnung Rot Kanal

Die Gewichtung führt dazu das in Abbildung 25 der Wert „e“ am stärksten betont wird, wobei die Nachbarwerte weniger gewichtet werden. Abbildung 26 zeigt, wie aus dem gewichteten R Kanal Block ein einziger Farbwert wird.

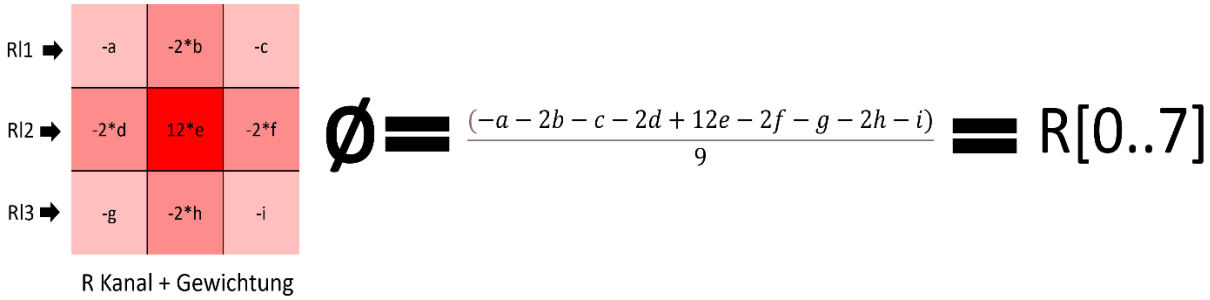


Abbildung 26: Berechnung Farbwert R

VGA Ausgabe

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

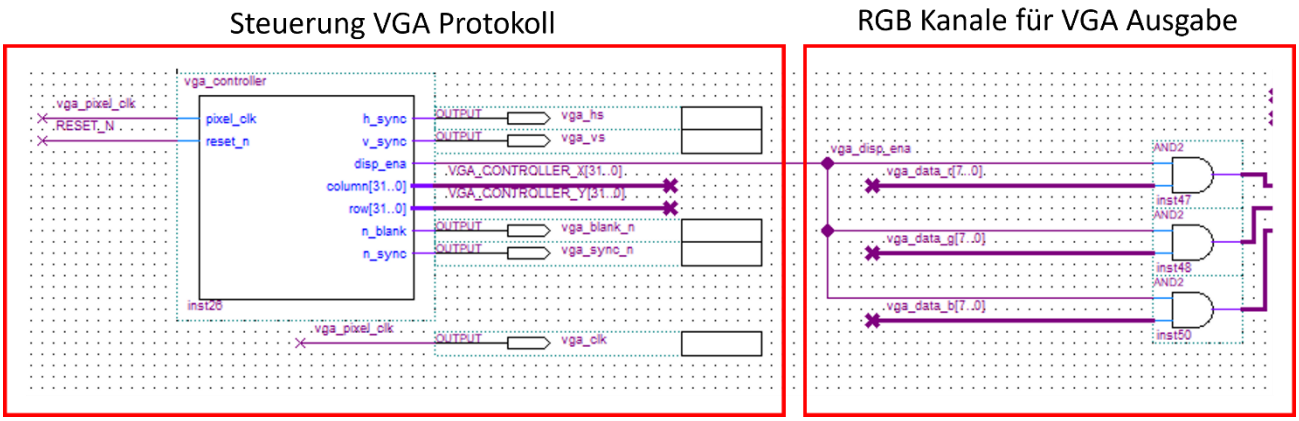


Abbildung 27: Schaltbild VGA Ausgabe

In Abbildung 27 zu sehen sind alle Steuersignale für die VGA Schnittstelle zu sehen. Im rechten Bereich finden wir die Farbkanäle „vga_data_r“, „vga_data_g“ und „vga_data_b“, die unsere Bilddaten darstellen. Diese Signale kommen aus dem „Convolution“