Projektarbeit

Bildverarbeitung auf FPGA Board

Author:

Christoph Paa

Betreuer:

Prof. Dr.-Ing. Buerkle

Sommersemester 2012

Inhaltsverzeichnis

1 Über das Projekt

1.1 Zielsetzung

Ziel des Projektes ist es, dem von Aleksej Wolf erstellten (M)JPEG decoder[1] Möglichkeiten der Bildverarbeitung hinzuzufügen.

1.2 Zusammenfassung der Veränderungen

Zunächst wurde das Projekt so verändert, dass es anstatt 2 Entwicklungsboards und einer FTDI Platine auf einem einzelnen Terasic DE0 Board läuft. Hierzu wurde die Taktrate auf 100Mhz erhöht sowie die Ansteuerung des SDRAMs von Herrn Ehemann komplett neu entwickelt. Zur Übertragung der Daten wird die auf dem Board vorhandene RS-232 UART Schnittstelle verwendet. Hierzu ist es lediglich notwendig eine Buchse am Board anzulöten, die Notwendigen Signalpegel sind bereits vorhanden. Auf dem Board wurde ein frei verfügbarer RS-232 UART core verwendet.[2] Als Frontend wird hierfür ein Matlab GUI verwendet. Diese Übernimmt das übermitteln der Bilddaten an das Board sowie das Konfigurieren der Filtereinstellungen.

2 Matlab GUI

Da die Grafikkarte nun einige Funktionen bekommen soll, welche Parameter erfordern ist eine Kommunikation zwischen dem Board und einem PC notwendig. Um die Übermittlung von Befehlen und Daten möglichst Benutzerfreundlich zu gestalten wurde entschieden, diese Kommunikation auf Seiten des PCs mit einem GUI zu steuern. Als Basis für das Serielle Kontroll- GUI wurde Mathworks Matlab gewählt, da es damit erstens einfach ist mithilfe des mitgelieferten GUIDE Tools GUIs zu erstellen und es vorgefertigte Funktionen zur seriellen Verbindung gibt. Zunächst wurde eine Klasse namens *serial_con* entwickelt welche die Serielle Verbindung kontrolliert und die den aktuellen Status des Boards überwacht. Diese Klasse regelt wann welche Schreib und Leseoperationen am Board durchgeführt werden dürfen bzw. müssen. Die Graphische Benutzeroberfläche bietet die Möglichkeit Bilder an das Board zu übertragen und die Parameter der Bildverarbeitungsfilter anzupassen.

2.1 Communication setup

Das *communication setup* Panel dient dazu zu Beginn der Kommunikation den RS-232 Port des PCs einzustellen. Er übermittelt Einstellungen an das *serial_con* Objekt und steuert das Öffnen und schließen des Ports. Außerdem steuert der Zustand des Ports welche Elemente des GUI aktiv sind. Ist der Port noch geschlossen sieht das *communication setup* Panel folgendermaßen aus:

Sobald man die Einstellungen für *COM port* sowie *Baud rate* getroffen hat kann man mit *open port* den RS-232 Port öffnen. Im unteren Teil des Panels wird der Aktuelle

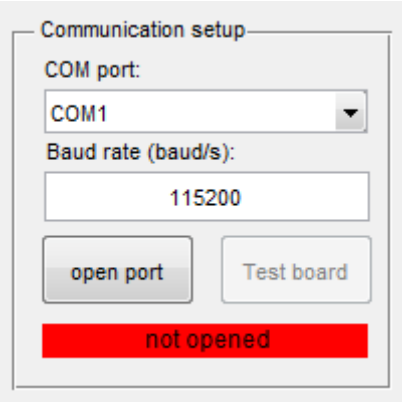

Abbildung 2.1: RS-232 Status rot

Verbindungsstatus Farbcodiert dargestellt. Ist das öffnen der Schnittstelle erfolgreich wechselt das Panel in den Nächsten Zustand:

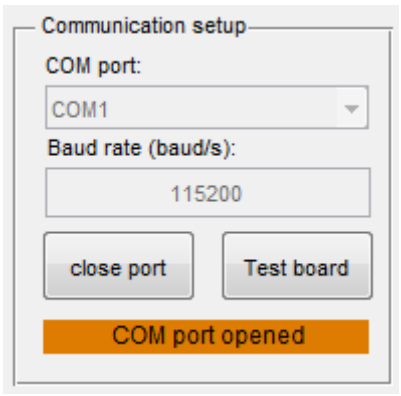

Abbildung 2.2: RS-232 Status orange

Da der Port nun geöffnet ist ist es nun nicht mehr möglich die *COM port* und *Baud rate* Einstellungen zu manipulieren. Der Status im unteren Teil des Panels zeigt an, dass der Port erfolgreich geöffnet wurde. Außerdem wurde nun der Button zum testen der Verbindung zwischen PC und Board aktiviert. Wenn dieser Test nun ausgeführt wird und das Board auf die Anfrage antwortet sieht das GUI die Verbindung als hergestellt an, aktiviert die anderen Panels und aktualisiert den Status. Das Panel sieht dann folgendermaßen aus:

Es ist natürlich jederzeit möglich die Schnittstelle wieder zu schließen und die Verbin-

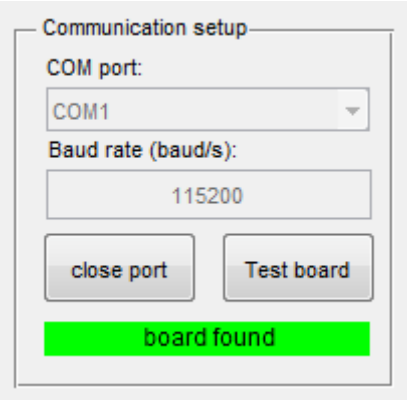

Abbildung 2.3: RS-232 Status grün

dung mit geänderten Einstellungen neu aufzubauen. Hierbei ist jedoch zu Beachten, dass der RS-232 core auf dem Board für geänderte *Baud rate* Einstellungen neu konfiguriert und kompiliert werden muss.

2.2 write file

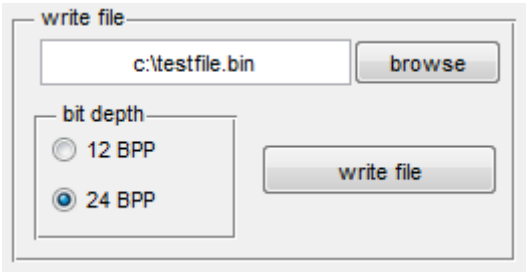

Abbildung 2.4: *Write file* Panel

Dieses Panel steuert die Übertragung von Bilddaten an das Board. Hier wird die zu Übertragende Datei gewählt sowie die Bittiefe eingestellt. Momentan unterstützt die Implementation lediglich Binärdateien im 24 Bit Format. Bilder können entweder durch direkte Eingabe des Pfades oder durch einen File Dialog, welcher über die *browse* Schaltfläche geöffnet wird, ausgewählt werden:

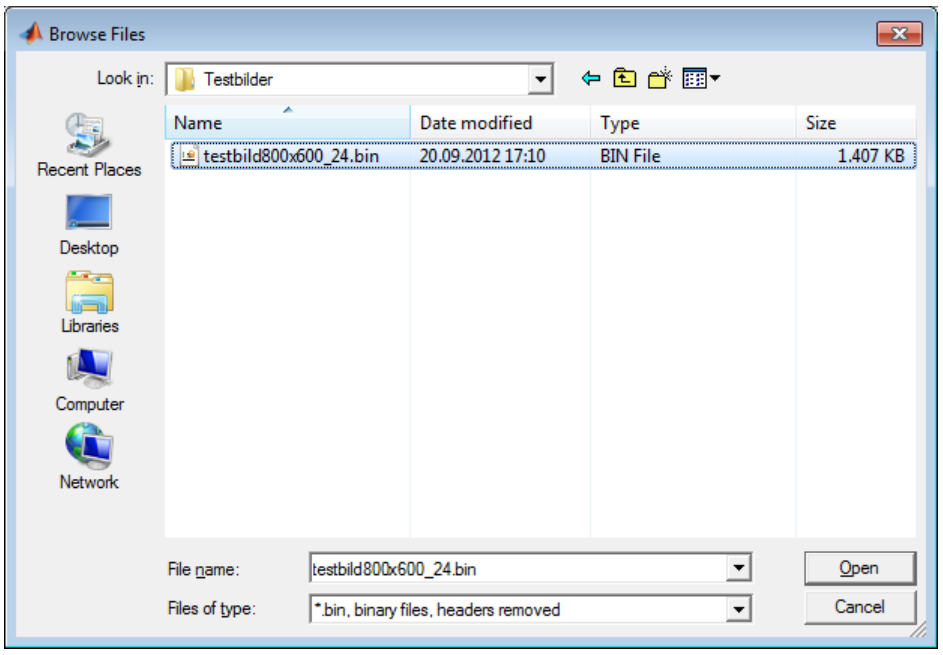

Abbildung 2.5: *Browse file* Dialog

2.3 filter options

Hier werden die Einstellungen für die Filter vorgenommen und diese an das Board übertragen. Die beiden Panels in Abbildung 2.6 und 2.7 dienen zur Übertragung der Einstellungen an die Filter auf dem Board. Wenn *apply filter* betätigt wird werden die in Kapitel 4.2 bzw. 4.3 beschriebenen Filter Gestartet. Die dabei übertragenen Befehle sind in auf Seite 9 in Tabelle 3.1 beschrieben.

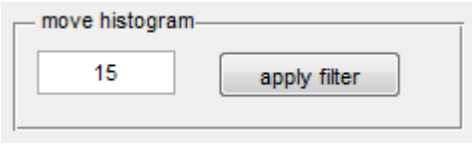

Abbildung 2.6: Hist. verschieben Panel

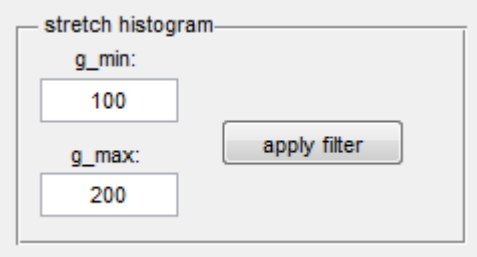

Abbildung 2.7: Hist. spreizen Panel

2.4 debug options

Das GUI beinhaltet ein zusätzliches Fenster welches nützliche Funktionen zum Debuggen von Änderungen an der Hardware beinhaltet. Hiermit ist es möglich bestimmte Datenmengen über die RS-232 Schnittstelle zu senden und Antworten des Boards zu empfangen. Dieser Teil des GUI wurde während des Entwicklungsprozesses ständig nach Bedarf verändert und er ist nicht immer mit der Aktuellen Version des FPGA Designs synchron.

3 RS-232 Kommando Empfänger

Die Grafikkarte mit Bildverarbeitung wird von einer Statemachine gesteuert, welche von RS-232 Kommandos beeinflusst wird. Als Kommandos werden hier ASCII Steuerzeichen verwendet. Folgende Befehle stehen zur Verfügung:

Hexadezimal	Steuerzeichen	Richtung	Beschreibung
0x05	ENQ	PC —> Board	Anfrage and das Board, wird automatisch mit ACK beantwortet
0x06	ACK	PC <— Board	Antwort auf ENQ Anfrage
0x02	STX	PC —> Board	Start der Bildübertragung, 3072 pages mit jeh 256 x 12/24 Bit
0x17	ETB	PC <— Board	Ende eines 256 x 12/24 Bit Blocks erreicht, Daten verarbeitet
0x11	DC1	PC —> Board	Filter Histogrammverschiebung, gefolgt von einem signed 8 Bit Integer
0x12	DC2	PC —> Board	Filter Histogrammspreizung, gefolgt von zwei 8 Bit unsigned Integern (g_{min}, g_{max})

Tabelle 3.1: Tabelle der RS-232 Befehle

Nachdem ein Datum vom RS-232 Empfänger empfangen wurde wird überprüft, welchem Befehlt es entspricht. Dies Geschieht im unten gelisteten Code. Hierbei wird mit

Hilfe der *rx_busy* und *rx_busy_last* Signale der Flankenwechsel am Befehlsempfänger synchron zum Systemtakt festgestellt. Das Empfangene Kommando wird mit der Funktion *get_rx_command* dekodiert. Außerdem wird das senden des *ACK* Steuerzeichens als Antwort auf *ENQ* direkt hier verarbeitet. Um beim Empfang von Bilddaten diese nicht versehentlich zu interpretieren wird der Befehlsdecoder während des Bildempfangs deaktiviert indem der Zustand *s_wait_for_com* nicht aufgerufen wird.

```
1  when s_wait_for_com =>
2      pic_received <= '0';
3      data_out     <= x"00";
4      TX_start     <= '0';
5      if rx_busy = '1' then
6          rx_busy_last <= '1';
7      elsif rx_busy = '0' and rx_busy_last = '1' then
8          rx_busy_last <= '0';
9          rx_cmd_var := get_rx_command(data_in);
10         rx_cmd <= rx_cmd_var;
11         if rx_cmd_var = check_com then
12             tx_cmd <= board_ack;
13         else
14             tx_cmd <= unidentified;
15         end if;
16     end if;
```

3.1 Veränderung der Hardware

Zunächst war geplant, die Datenübertragung über den USB-Anschluss des Boards zu implementierten. Dieser ist über einen FTDI-Chip auf dem Board mit dem als USB-Blaster konfigurierten Altera MAX II Baustein verbunden. Diese Art der Kommunikation wird auch von dem von Terasic zu Verfügung gestellten GUI verwendet. Jedoch stellte sich heraus, dass die Schnittstelle nicht in den Unterlagen zum Board dokumentiert ist. Auf Anfrage nach Unterlagen über diese Schnittstelle hat Terasic Support mir jedoch geraten aufgrund der Komplexen Anbindung des USB Anschlusses an den FPGA eine andere

Schnittstelle, wie z.B. das RS-232 Interface zu wählen. Das Datenblatt des auf dem Board vorhandenen Texas Instruments MAX232 Driver/Receiver ICs ist in den Unterlagen zum DE0 Board mitgeliefert. Die Verwendung der Schnittstelle erforderte jedoch zunächst das Anbringen eines DE-9 Steckers an die dafür vorgesehenen Lötpunkte des DE0 Boards. Hierbei ist bei der Pinbelegung darauf zu Achten, ob man einen Stecker oder eine Buchse verwendet, denn dies beeinflusst welche Art von Kabel man verwendet. Kabel mit einer Stecker-Stecker Belegung sind meist intern Gedreht, wohingegen Stecker-Buchse Kabel keine Drehung der Anschlüsse aufweisen.

3.2 Datenübertragung

Die Bilder sind auf dem PC als 24 bit *.bmp Datei mit einer Auflösung von 1024 x 768 Pixeln und entferntem Header gespeichert. Für die Datenübertragung werden sie in Matlab eingelesen und in Sektionen von jeweils 256 Pixeln unterteilt. Diese $\frac{256*24}{8} = 768$ Byte werden dann dem RS-232 Buffer übergeben und gesendet. Das Board empfängt diese Sektion in einem auf dem FPGA befindlichen Puffer. Dieser ist als Altera Cyclone III M9K Single-Port Ram implementiert. Wenn der komplette Block auf dem FPGA gepuffert ist wird er in den SDRAM geschrieben. Wenn die Schreiboperationen abgeschlossen sind signalisiert das Board dem PC, dass es bereit ist das nächste Datensegment zu empfangen. Die Datenübertragung erfolgt mit 115200 Baud/s, die Übertragung eines Vollständigen Bildes dauert hierbei etwa 4 Minuten.

4 Bildverarbeitung

Die beiden Bildverarbeitungsoperatoren die auf dem FPGA implementiert sind werden jeweils angewandt wenn sich das Bild vollständig im SDRAM befindet. Das Quellbild befindet sich hierbei auf der Ersten, das Ergebnisbild auf der zweiten Speicherbank des SDRAM. Die Operatoren arbeiten mit 24 Bit Farb- bzw. Grauwerten.

4.1 Punktoperatoren

Beide vorhandenen Operatoren zählen zur Klasse der Pixelverarbeitende Operatoren. Diese zeichnen sich dadurch aus, dass zu ihrer Berechnung jeweils nur *ein* Pixel verändert werden muss und diese Veränderung nicht von umliegenden Pixeln beeinflusst wird. So wird bei der Histogrammverschiebung jedem Pixel ein Fester Wert addiert bzw. von ihm subtrahiert. Manche Pixelverarbeitende Operatoren setzen auch eine vorherige Analyse des Bilder voraus, so z.B. die Histogrammspreizung. Hierbei wird anhand des Histogramms die Grauwertverteilung beurteilt und der Filter dann entsprechend eingestellt.

4.2 Histogrammverschiebung

Durch das Gleichförmige Verschieben des Histogramms in eine Richtung wird die *mittlere Helligkeit* des Bildes verändert. Diese Operation ist nicht Umkehrbar, da Helligkeitswerte am oberen bzw. unteren Rand des Histogramms abgeschnitten werden. In den Beispielabbildungen 4.1 und 4.2 Ist das Histogramm eines Bildes vor und nach der Transformation mit einem Hisogrammverschiebungsoperator zu sehen. In 4.1 ist zusätzlich die

Transformationskennlinie in rot zu sehen. Die Verschiebung beträgt im Beispiel etwa 63 Grauwertstufen nach Links.

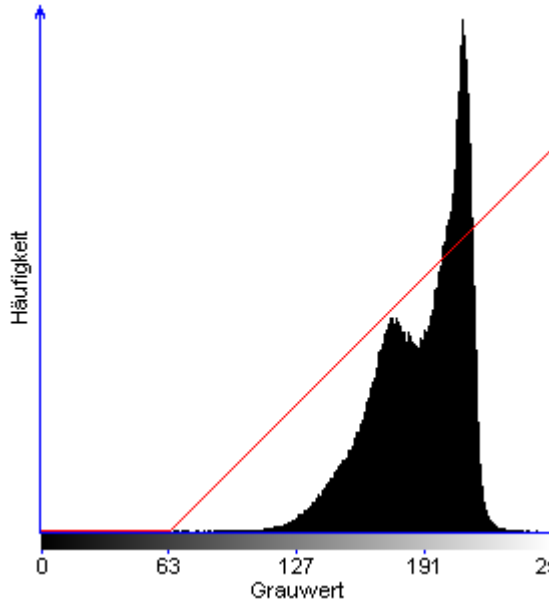

Abbildung 4.1: Hist. vorher[3]

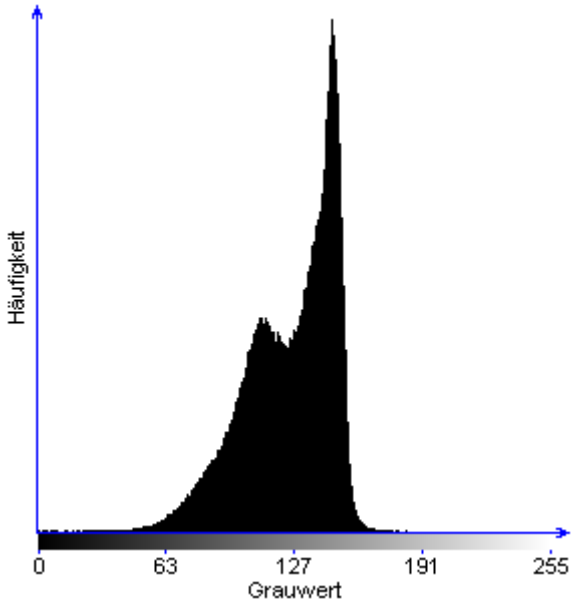

Abbildung 4.2: Hist. nachher[4]

4.2.1 Implementation

Die Histogrammverschiebung wird mit einem zweistufigen RS-232 Befehlt aufgerufen. Hierbei wird zuerst der Befehlscode für "Histogrammverschiebungünd dann der Wert übergeben. Der Wert ist ein 8 Bit signed integer, was eine Verschiebung von -128 bis +127 Stufen erlaubt. Wenn beide Teile des Befehls empfangen sind werden die Werte des Bildes in 256 Pixel Blöcken aus dem SDRAM ausgelesen. Die einzelnen Farbkanäle jedes Pixels werden dann mithilfe von folgender Funktion mit dem übergebenen Wert Addiert:

```
1 function capped_add(sum1 : unsigned(7 downto 0);
2                     sum2 :   signed(7 downto 0)
3                     ) return unsigned is
4     --adds / subtracts sum2 from unsigned sum1,
5     --returns result in 0...255 range.
6     variable erg : signed(9 downto 0) := (others => '0');
7     begin
```

```
8           erg := signed("00" & sum1) + sum2;
9           if erg > 255 then
10              return to_unsigned(255,8);
11          elsif erg < 0 then
12              return to_unsigned(0,8);
13          else
14              return unsigned(erg(7 downto 0));
15          end if;
16  end function capped_add;
```

Hierbei werden die von ieee.numeric_std eingeführten *signed* und *unsigned* Datentypen verwendet um korrekte Ergebnisse sicher zu stellen. Diese Funktion überprüft darüber hinaus ob der neue Helligkeitswert den Wertebereich 0...255 überschreitet. Falls dies der Fall ist wird er auf den Maximal bzw. Minimalwert gesetzt, dies wird in der Bildverarbeitung als *Clamping* bezeichnet. Die berechneten Werte werden dann in der zweiten Bank des SDRAM abgespeichert. Wenn das Komplette Bild bearbeitet ist wird die Darstellung von Bank 1 auf Bank 2 gewechselt und somit das verarbeitete Bild dargestellt.

4.3 Histogrammspreizung

Bei der Histogrammspreizung wird ein Bild mit schwacher Kontrast dadurch aufgewertet, dass der Verfügbare Helligkeitsraum im Histogramm besser ausgenutzt wird. Hierzu wird ein vorhandener schmaler Bereich an Helligkeitswerten wie in Abb. 4.3 auf einen möglichst großen Bereich Abgebildet (Abb. 4.4).

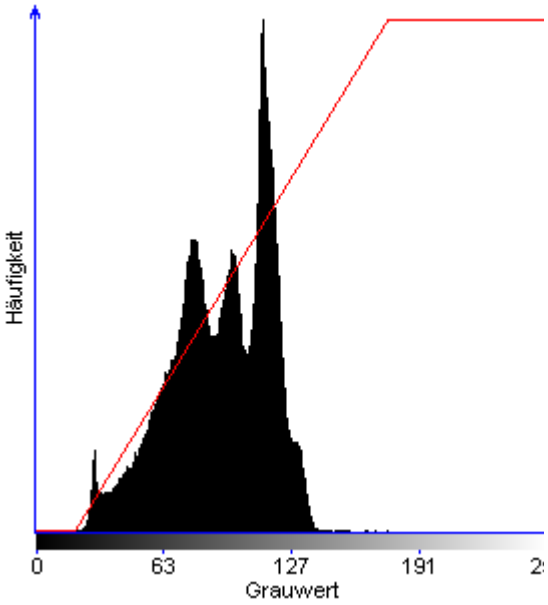

Abbildung 4.3: Hist. vorher[5]

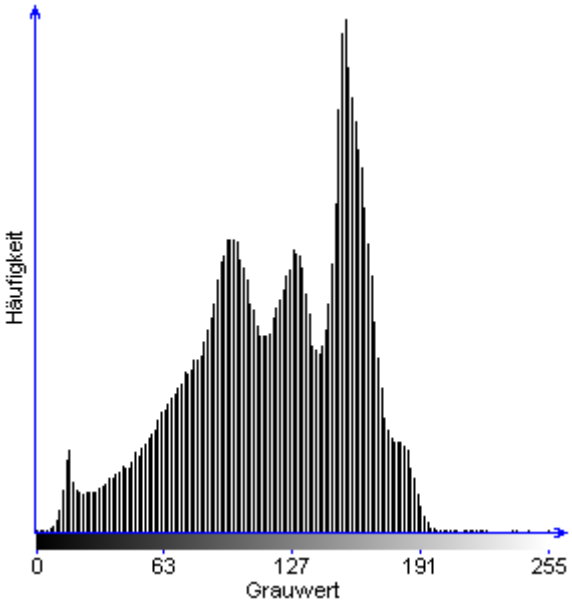

Abbildung 4.4: Hist. nachher[6]

4.3.1 Implementation

Die Histogrammspreizung besteht grob aus zwei Operationen: Dem Erstellen einer Lookup-Table (LUT) und dem Ersetzen der Farbwerte. Die LUT wird Erstellt, damit die Berechnung der Gleichungen 4.1 und 4.2 nur 256 mal ausgeführt werden muss anstatt $1024 * 768 * 3 = 2359296$ mal. Beim Ersetzen der Farbwerte ist es dann lediglich notwendig den Korrekten Wert aus der Wertetabelle auszulesen. Zur Histogrammspreizung wird zunächst eine eine Lookup-Table erstellt. Dafür werden die Anfangs und Endwerte der Transformationskennlinie via RS-232 übertragen. Die Transformationskennlinie $T_{stretch}(g)$ wird dann vom FPGA mit Hilfe der Gleichungen 4.1 und 4.2 berechnet und in einem M9K Single Port Ram mit 8 Bit Adressbreite sowie 8 Bit Datenbreite gespeichert.

$$T_{trans}(g) = \begin{cases} 0 & \text{if } g \leq g_{min}, \\ T_{stretch}(g) & \text{if } g_{min} \leq g < g_{max}, \\ 255 & \text{if } g_{max} \leq g. \end{cases} \tag{4.1}$$

$$T_{stretch}(g) = 256 * \frac{g - g_{min}}{g_{max} - g_{min}} \tag{4.2}$$

Die Berechnung der Gleichungen 4.1 und 4.2 wird von der Funktion *hist_stretch_calc* im Paket *graka_pack.vhd* Übernommen:

```
1  function hist_stretch_calc(
2      g       : unsigned(7 downto 0);
3      g_min   : unsigned(7 downto 0);
4      g_max   : unsigned(7 downto 0)
5      ) return unsigned is
6
7      variable gi : signed(9 downto 0);
8      variable gi_min : signed(9 downto 0);
9      variable gi_max : signed(9 downto 0);
10     variable erg : signed(17 downto 0);
11
12     variable div1 : signed(17 downto 0);
13     variable div2 : signed(17 downto 0);
14     begin
15         gi := signed("00" & g);
16         gi_min := signed("00" & g_min);
17         gi_max := signed("00" & g_max);
18
19         if gi > gi_max then
20             return to_unsigned(255,8);
21         elsif gi > gi_min then
22             div1 := signed((gi - gi_min) & "00000000");
23             div2 := signed("00000000" & (gi_max - gi_min));
24             erg := div1 / div2;
25             return unsigned(erg(7 downto 0));
26         else
27             return to_unsigned(0, 8);
28         end if;
29 end function hist_stretch_calc;
```

Hierbei werden zunächst in Zeile 15-17 die Signale gg_{min}, g_{max} in den Datentyp *signed* überführt, um bei der Berechnung von Zähler $g - g_{min}$ und Nenner $g_{max} - g_{min}$ sicher zu stellen, dass keine Überläuft stattfinden. Die in Gleichung 4.1 beschriebenen

Wertebereiche werden von den *if, elsif, else* Statements in Zeilen 19, 21 und 26 behandelt. Gilt $g_{min} \leq g < g_{max}$ werden die beiden Dividenden *div1* und *div2* erstellt. Da der Faktor 256 in Binärer Arithmetik einer einfachen Verschiebung entspricht wird er direkt in den Nenner Multipliziert indem dieser in Zeile 22 um 8 Stellen nach Links verschoben wird. Hierdurch kann die Division in Zeile 24 ohne die Verwendung von Fest- oder Gleitkommazahlen durchgeführt werden, was zusätzliche Libraries erfordert hätte.
Die Berechnete Transformationskennlinie wird angewendet, indem 256 Pixel aus dem SDRAM ausgelesen werden und jeder Farbwert durch den Wert in der LUT ersetzt wird. Dies wird dadurch vereinfacht, dass der 8 Bit Farbwert aus dem SDRAM direkt die 8 Bit Adresse des neuen Farbwertes in der LUT Darstellt.

5 Zusätzliche Informationen zum Projekt

Das Projekt wurde zusammen mit dem Projekt *24 Bit Grafikkarte* von Herrn Christoph Ehemann entwickelt. Zur einfacheren Kollaboration kam das open-source Versionsverwaltungssystem git[1] zum Einsatz. Als Repository Host wurde github[2] eingesetzt. Sämtliche Versionen des Quellcodes sind deshalb in der git Repository `https://github.com/youRFate/graka-projekt.git` verfügbar. Dieses Dokument wurde mit LaTeX erstellt[3].

[1] http://git-scm.com/
[2] https://github.com/
[3] http://www.latex-project.org/

Quellenverzeichnis

[1] Aleksej Wolf: *Implementation und Test eines (M)JPEG Decoders auf einem FPGA* Bachelor Thesis HTW-Aalen 20.10.2011

[2] Lothar Miller: *Implementierung einer RS232 Schnittstelle* Implementierung einer RS232 Schnittstelle 8.7.2009

[3] Wikimedia commons: *Erechtheum Acropolis 1853 histogram tkl trans.png* `http://commons.wikimedia.org/wiki/File:Erechtheum_Acropolis_1853_histogram_tkl_trans.png`

[4] Wikimedia commons: *Erechtheum Acropolis 1853 trans histogram.png* `http://commons.wikimedia.org/wiki/File:Erechtheum_Acropolis_1853_trans_histogram.png`

[5] Wikimedia commons: *Bike sapa29 clipping histogramm tkl.png* `http://commons.wikimedia.org/wiki/File:Bike_sapa29_clipping_histogramm_tkl.png`

[6] Wikimedia commons: *Bike sapa29 clipping stretch histogram.png* `http://commons.wikimedia.org/wiki/File:Bike_sapa29_clipping_stretch_histogram.png`