Hochschule Aalen

Projektarbeit

24-Bit Grafikkarte

Author:

Christoph Ehemann

Mat. Nr: 30035

christoph.ehemann@gmx.net

Betreuer:

Prof. Dr.-Ing. Buerkle

Sommersemester 2012

Inhaltsverzeichnis

1. Einleitung

An der Hochschule Aalen besteht ein breites Spektrum an Arbeiten welche sich mit Bild- und Videobearbeitung beschäftigen oder einer Monitorausgabe für andere Zwecke verwenden. Jedoch nutzen all dieses Projekte bestehende VGA-Schnittstellen mit einer maximalen Farbauflösung von 8-Bit, was eine Darstellung von $2^8 = 256$ Farben ermöglicht. Allerdings ist dies relativ wenig im Vergleich zum heutigen Standard von 24-Bit, also $2^{24} = 16777216$ Farben.

Des Weiteren sind derartige Darstellungsformate mit einem Datenaufwand verbunden, welcher die Kapazitäten eines Cyclone 3 übersteigt. Also muss auch eine Anbindung an einen externen Speicher entwickelt werden. Hierfür wurde der auf dem DE0-Board verbaute SDRAM verwendet.

Ziel dieser Arbeit ist es eine Grafikkarte mit einer Farbauflösung von 24-Bit zu entwickeln und diese in ein bestehendes Projekt einzuarbeiten. Hierfür wurde das Projekt von Herrn Christoph Paa verwendet, welcher zeitgleich ein Programm zur Bildbearbeitung auf Basis eines FPGA im Rahmen seiner Projektarbeit entwickelte.

2. VGA

2.1. VGA-Standard

VGA (lang: Video Graphics Array) ist ein analoger Grafikstandard, welcher 1957 von IBM eingeführt wurde. Er stellte damals den Nachfolger zu den bisherigen Standards von IBM - EGA(Enhanced Graphics Adapter) und CGA (Color Graphics Adapter) - dar und wird selbst heute noch von den meisten Geräten unterstützt.

2.2. Funktionsweise

Eine VGA-Schnittstelle setzt sich im wesentlichen aus folgenden Signalen zusammen:

- drei analoge Signale für jeweils Rot, Grün und Blau
- HSYNC, dem Signal zur horizontalen Synchronisation
- VSYNC, dem Signal zur vertikalen Synchronisation

Da VGA noch aus der Zeit von CRT-Monitoren stammt, ist die Funktionsweise auch damit zu erklären. Bei CRT-Monitoren läuft der Elektronenstrahl von links oben, zeilenweise bis nach rechts unten. Auf jeder Seite des Bildschirms durchläuft er Bereiche außerhalb der sichtbaren Bildschirmfläche. Wie auf Abbildung 2.1 zu sehen ist, wird zu Beginn einer jeden Zeile der sichtbare Bereich durchlaufen, während auf den RGB-Pins analog der Farbwert übertragen wird. Anschließend folgen die Bereiche „Front Porch“, H-Sync und „Back Porch“. Während diesen sollten keine Farbwerte übertragen werden, da das bei manchen Bildschirmen zu Fehlern führt. H-Sync ist ein Rechteck-Impuls an

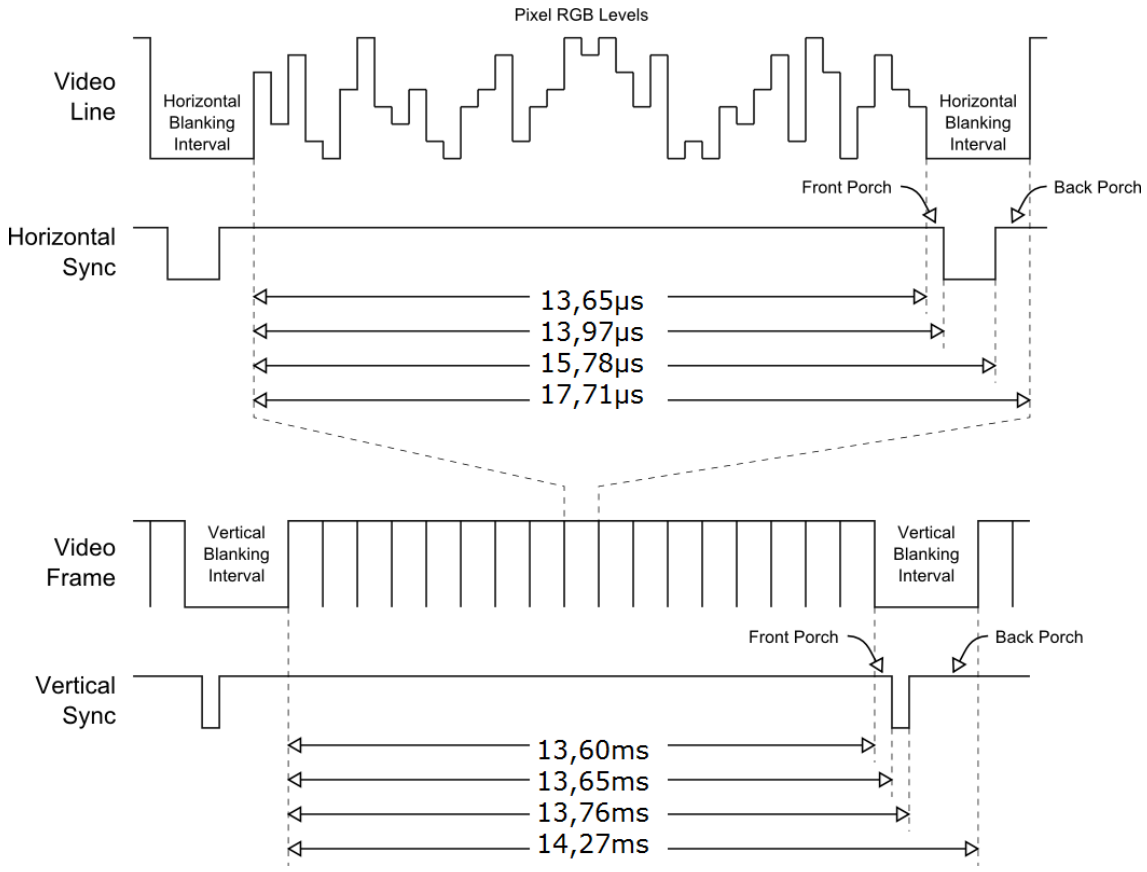


Abbildung 2.1.: VGA-Timings: VESA 1024*768 bei 70Hz [1].

dem sich der Bildschirm orientiert und erkennt, welche Auflösung verwendet wird. Nach einem Bild, also bspw. nach 768 Bildzeilen bei einer Auflösung von 1024*768 Pixeln, folgt in der Vertikalen ein solcher Bereich außerhalb der Anzeige, welcher ebenso aus „Front Porch“, H-Sync und „Back Porch“ besteht.

Durch das Timing der Pulse HSYNC und VSYNC, schließt der Monitor auf die verwendete Pixel-, Monitorfrequenz, etc. zurück und man muss nun zum richtigen Zeitpunkt den gewünschten Farbwert pro Pixel aus den RGB-Signalen zusammensetzen. Aufgrund einer einfacheren Speicheradressierung, wird hier eine Auflösung von 1024*768 verwendet. Die hierfür verwendeten Timings sind Tabelle 2.1 zu entnehmen.

Tabelle 2.1.: VGA-Timings: 1027x768 bei 60Hz

Generelle Timings	
Bildschirmwiederholungsrate	60Hz
Pixelfrequenz	65MHz

Horizontale Timings (Zeile)		
Zeilen-Abschnitt	Pixel	Zeit [μS]
Sichtbarer Bereich	1024	15,75
Front Porch	24	0,37
HSYNC Puls	136	2,09
Back porch	160	2,46
Gesamte Zeile	1344	20,68

Vertikale Timings (Bild)		
Bild-Abschnitt	Zeilen	Zeit [mS]
Sichtbarer Bereich	768	15,88
Front Porch	3	0,06
VSYNC Puls	6	0,12
Back Porch	29	0,60
Gesamtes Bild	806	16,67

2.3. VGA-Modul

Das entwickelte VGA-Modul ist verantwortlich für die Kommunikation mit dem Bildschirm und arbeitet mit den zuvor angesprochenen Timings. Es erhält die anzuzeigenden Farbwerte über den Eingangeingangsbus *pixel* und gibt seine aktuelle Bildschirmposition

an der SDRAM-Controller über die Signale *Vcnt* und *Hcnt* weiter.

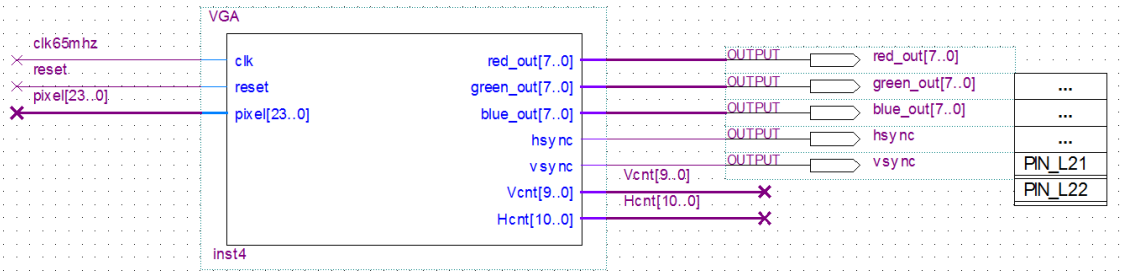


Abbildung 2.2.: VGA-Modul

3. SDRAM-Controller

Da die interne Speichergröße des verwendeten Cyclone 3 FPGA nur 516 kBit beträgt und ein 1024*768 Pixel großes Bild mit 24 Bit Farbauflösung auf eines Größe von $1024 * 768 * 24 = 18{,}87 MBit = 18 MiBit$ kommt, wird auf den verbauten 64MiBit großen Zentel SDRAM zurückgegriffen.

3.1. Synchronous Dynamic Random Access Memory

Synchronous Dynamic Random Access Memory (kurz: SDRAM) ist eine 1997 eingeführt Speichertechnologie, welche vor allem in PCs zur Anwendung kam. Sie verwendet Kondensatoren als speicherndes Element, was zu einem flüchtigen (volatile) Speichermedium führt. Das „Synchronous“ im Namen geht auf die Verwendung eines Taktes zur Synchronisierung des Speichers zurück. Da sich die Speicherkondensatoren über die Zeit entladen, benötigen die Speicherzellen in bestimmten Zeitabständen eine Erneuerung der gespeicherten Daten, daher das „Dynamic“ im Namen. „Random Access“ bedeutet, dass man auf zufällige Zellen im Speicher zugreifen kann, im Gegensatz zu Speichern mit sequenziellen Zugriffen (Vgl. FIFO).

3.2. Speicherorganisation

Der verwendete Speicher besteht aus vier Bänken, welche jeweils in 4096 Zeilen mit je 256 Spalten unterteilt sind. Jeder der derartig adressierten Speicherbereiche fasst 16 Bit, was zu einem Gesamtspeichervolumen von $4 * 4096 * 256 * 16 Bit = 67108864 Bit$ ($/2^{20} = 64 MiBit$). Verdeutlicht wird die nochmals durch Abbildung 3.1.

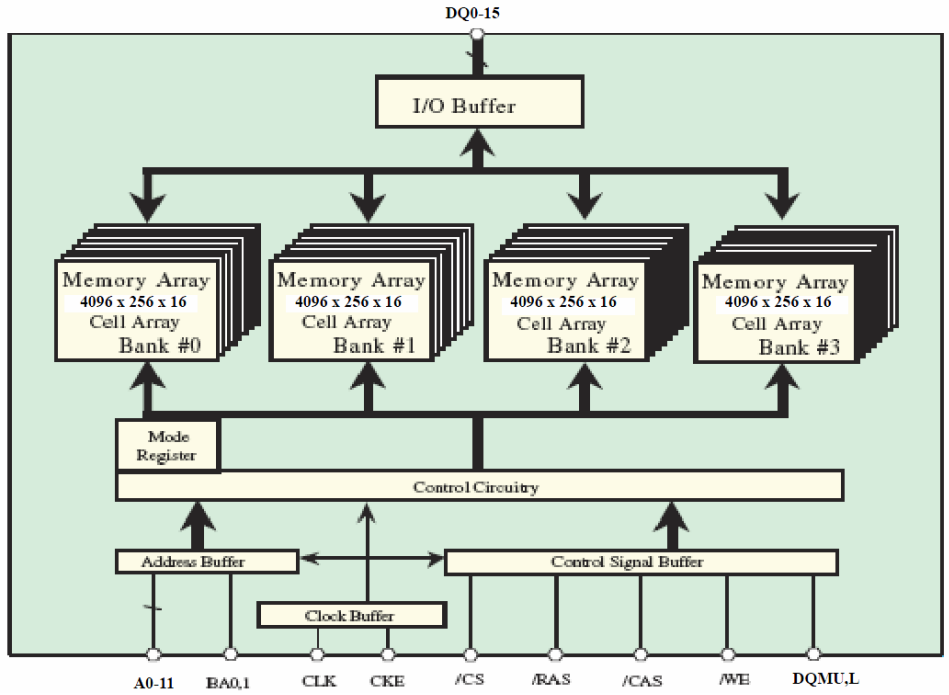


Abbildung 3.1.: SDRAM-Organisation

3.3. Speicherinterface

Die Kommunikation mit dem SDRAM läuft über die Pins in Tabelle 3.1 (Richtung aus Sicht des Controllers).

Tabelle 3.1.: SDRAM-Pins

Pin-Name	Richtung	Busbreite	Aufgabe
DQ	I/O	16	Daten Ein- und Ausgang.
ADDR	Ausgang	13	Gibt je nach Befehl Zeilen-(A0-A11) oder Spaltenadresse(A0-A7) an. Wärend einem Write- oder Read-Befehlt steuert A10 die auto-precharge-Option.
BA	Ausgang	2	Bank Address wählt die zu verwendende Bank aus.
DQM	Ausgang	2	Gibt die Möglichkeit die oberen oder unteren 8 Bit des Adress-Bus hochohmig zu schalten und kann damit zur Maskierung verwendet werden. DQM1 steuert dabei DQ15-DQ8 und DQM0 entsprechend DQ7-DQ0.
WE	Ausgang	1	Steuert den Kommando-Interpreter.
CAS	Ausgang	1	Steuert den Kommando-Interpreter.
RAS	Ausgang	1	Steuert den Kommando-Interpreter.
CS	Ausgang	1	Chip Select aktiviert (logisch '0') oder deaktiviert den Kommando-Interpreter (logisch '1')
CKE	Ausgang	1	Clock Enable aktiviert den Takt-Eingang des RAMs. Kommt auch zum Einsatz bei speziellen Betriebsmodi wie Power-Down, Sleep, etc.
RAM_CLK	Ausgang	1	RAM-Takt-Eingang.

3.4. Befehlssequenzen

Prinzipiell werden Befehle gestartet, indem bei korrekt angelegten Daten und Adressen das CS-Signal für einen Takt auf logisch '0' gesetzt wird und anschließend die geforderten Timings erfüllt werden.

Initialisierungssequenz

Sobald ein stabiler Takt am SDRAM anliegt, muss dieser zuerst initialisiert und konfiguriert werden. Diese Sequenz muss laut Datenblatt[2] aussehen wie in Tabelle 3.2.

Tabelle 3.2.: Initialisierungssequenz

ADDR	BA	DQM	RAS& CAS& WE	CS
x“0000”	“00”	“11”	“000”	’1’
Power-UP: 200μs Delay				
x“0400”	“00”	“11”	“010”	’0’
Precharge All: schließt alle Zeilen auf allen Bänken				
x“XXXX”	“00”	“11”	“XXX”	’1’
t_{RP}: 18ns Delay				
x“0000”	“00”	“11”	“001”	’0’
Refresh All: Erneuert den Speicherinhalt einer Zeile auf allen vier Bänken. Die Zeilen werden intern im SDRAM gezählt, daher müssen diese nicht adressiert werden. Nach Abschluss des Refreshs werden diese wieder geschlossen.				
x“XXXX”	“00”	“11”	“XXX”	’1’
t_{ARFC}: 60ns Delay				
x“0000”	“00”	“11”	“001”	’0’
Refresh All.				
x“XXXX”	“00”	“11”	“XXX”	’1’
t_{ARFC}: 60ns Delay				
x“0037”	“00”	“11”	“000”	’0’
Mode Register Set: Setzt die Grundeinstellungen des SDRAM per ADDR. • A2-A0: Burst-Länge, hier: Full Page • A3: Burst-Typ, hier: sequenziell • A6-A4: CAS-Latenz, hier: 3 • A9: Burst-Modus für Schreibzugriffe, hier: gleich wie Lesezugriff				
x“XXXX”	“00”	“11”	“XXX”	’1’
r_{MRD}: 2 Taktzyklen				

Lesesequenz

Unabhängig von Burst-Länge haben normale Lesezugriffe(Tab. 3.3) einen einheitlichen Ablauf. Es gibt auch Sonderfälle, die es einem beispielsweise ermöglichen auf die selbe Zeile in unterschiedlichen Bänken sehr schnell zuzugreifen, aber diese werden hier nicht verwendet.

Tabelle 3.3.: Lesesequenz

ADDR	BA	DQM	RAS&CAS&WE	CS
Speicherzeile	Bank	“11”	“011”	’0’
Bank Active: Aktiviert die angegebene Speicherzeile in der angegebenen Bank und ermöglicht so Lese- und Schreibzugriffe.				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{RCD}: 18ns Delay				
Speicherspalte	Bank	“00”	“101”	’0’
Read ohne Precharge(A10=’0’), der Burst beginnt in der angegebenen Spalte. Bei Lese-Zugriffen hat DQM ebenfalls eine CAS-Latenz, also muss auch DQM nun schon auf “00”gesetzt werden.				
x“XXXX”	“XX”	“00”	“XXX”	’1’
Einhaltung der angegebenen CAS-Latenz.				
x“XXXX”	“XX”	“00”	“XXX”	’1’
Jetzt folgen auf dem DQ-Bus bei jedem Takt die angeforderten Daten, so lange bis die angegebene Burst-Länge erreicht ist.				
x“XXXX”	Bank	“11”	“010”	’0’
Precharge selected bank: Schließt alle Zeilen auf der ausgewählten Bank.				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{RP}: 18ns Delay				

Schreibsequenz

Ebenso wie die Lesesequenzen sind Schreibsequenzen(Tab. 3.4) vom Ablauf her grundlegend gleich, es müssen nur abhängig von Burst-Länge genug Daten bereitgestellt werden.

Tabelle 3.4.: Schreibsequenz

ADDR	BA	DQM	RAS&CAS&WE	CS
Speicherzeile	Bank	“11”	“011”	’0’
Bank Active.				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{RCD}: 18ns Delay				
Speicherspalte	Bank	“00”	“100”	’0’
Write ohne Precharge(A10=’0’), der Burst beginnt in der angegebenen Spalte. Bei Schreib-Zugriffen besteht keine CAS-Latenz, also muss in diesem Takt schon die erste Speicherzelle beschrieben werden				
x“XXXX”	“XX”	“00”	“XXX”	’1’
Falls die Bust-Länge größer 1 ist, werden nun die folgenden Speicherzellen beschrieben.				
x“XXXX”	“XX”	“11”	“110”	’0’
Burst-Stop: Bei meiner Arbeit hat sich herausgestellt, dass der verwendete SDRAM bei einem Full Page Write nach einer vollen Zeile wieder zur Spalte 0 springt und dort nochmals 5 Zellen mit fehlerhaften Daten beschreibt, bis der Schreibzugriff tatsächlich endet. Also wird dieser per Burst-Stop abgebrochen.				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{RDL}: 2 Takte				
x“XXXX”	Bank	“11”	“010”	’0’
Precharge selected bank.				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{RP}: 18ns Delay				

Refresh

Damit während der Bildübertragung vom PC zum DE0-Board die Daten im SDRAM nicht verfälscht werden, folgt nach jedem Full-Page-Write ein Refresh des gesamten RAMs. Bei jedem „Auto Refresh All“ Befehl wird eine Zeile auf allen vier Bänken erneuert und danach sofort wieder geschlossen. Diese Zeile ist in einem RAM-internen Zähler hinterlegt.

Tabelle 3.5.: Auto Refresh

ADDR	BA	DQM	RAS&CAS&WE	CS
x“XXXX”	“XX”	“11”	“001”	’0’
Auto refresh				
x“XXXX”	“XX”	“11”	“XXX”	’1’
t_{ARFC}: 60ns Delay				

Da der Refresh-Befehl bei meinem RAM absolut nicht funktionierte, erneuere ich den Speicherinhalt, indem ich nach jeder empfangenen Zeile sämtliche bisher empfangenen Zeilen im RAM nacheinander aktiviere und anschließend wieder deaktiviere. Diese Methode dauert zwar länger, überzeugt jedoch durch Funktionalität.

3.5. SDRAM-Controller-Modul

Um sich die Entwicklung eines weiteren Kommunikationskanals zu ersparen, wurde das SDRAM-Controller in den Kommando-Decoder der RS232-Schnittstelle von Herrn Paa integriert. Die Namen der Schnittstellen zum SDRAM tragen den selben Namen wie im Kapitel Speicherinterface angeführt. Der Bus *pixel* gibt die per *Vcnt* und *Hcnt* von der VGA-Schnittstelle geforderten Farbwerte an diese zurück. Die restlichen Ein- und Ausgänge werden für die RS232-Kommunikation verwendet.

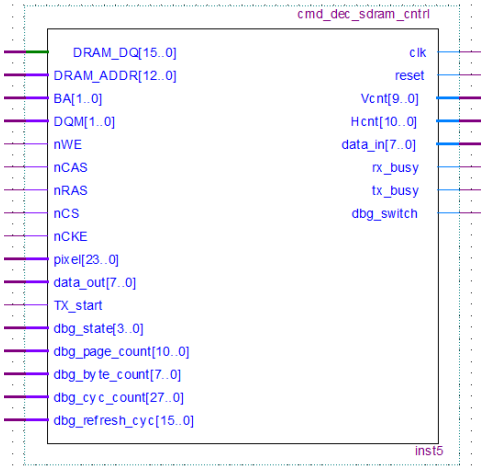


Abbildung 3.2.: SDRAM-Modul

4. 24-Bit Erweiterung

Da die Hardware auf dem DE0-Board nur eine Farbauflösung von 12-Bit ermöglicht, musste dieses über eine weitere Platine erweitert werden. Die digital-analog Wandlung für die Erzeugung der analogen Farbwerte des VGA-Anschluss erfolgt über ein Widerstandsnetzwerk (ähnlich einem R2R-Netzwerk), sowohl auf dem DE0-Board als auch auf dem Erweiterungsboard.

4.1. DAC per Widerstandsnetzwerk

Grundlegend funktionieren ein solcher DAC durch die parallele Verschaltung von Widerständen. Durch den 75Ω Eingangswiderstand des Monitors ergibt sich ein Spannungsteiler, welcher das Ausgangssignal im gewünschten Spannungsbereich zwischen 0V und 0,7V hält, trotz der 3,3V Ausgangsspannung des FPGAs. Würde man beispielsweise Bit0, Bit1 und Bit2 auf den logischen Wert '1' setzen, ergibt sich eine Parallelschaltung des 4kΩ, des 2kΩ und des 1kΩ Widerstands, also R=571,43Ω. Man erhält für $U_A = \frac{3,3V * R_L}{R + R_L} = \frac{3,3V * 75\Omega}{(571,43+75)\Omega} = 0,3829V$. Zu erwarten wären $\frac{7}{15} * 0,7V = 0,3267V$. Dies ist zwar nicht 100% genau, aber man kommt so zumindest bei Vollaussteuerung auf 0,7V.

Anhand dieser Formeln wurden nun Widerstandswerte für ein derartiges Netzwerk mit 8 Bit errechnet.

$$Bit0 : R_0 = \frac{3,3V * 75\Omega * 256}{0,7V} - 75\Omega = 90,44k\Omega \tag{4.1}$$

$$Bit1 : R_0 = \frac{3,3V * 75\Omega * 128}{0,7V} - 75\Omega = 45,18k\Omega \tag{4.2}$$

$$Bit2 : R_0 = \frac{3,3V * 75\Omega * 64}{0,7V} - 75\Omega = 22,55k\Omega \tag{4.3}$$

$$Bit3 : R_0 = \frac{3,3V * 75\Omega * 32}{0,7V} - 75\Omega = 11,24k\Omega \tag{4.4}$$

$$Bit4 : R_0 = \frac{3,3V * 75\Omega * 16}{0,7V} - 75\Omega = 5,58k\Omega \tag{4.5}$$

$$Bit5 : R_0 = \frac{3,3V * 75\Omega * 8}{0,7V} - 75\Omega = 2,75k\Omega \tag{4.6}$$

$$Bit6 : R_0 = \frac{3,3V * 75\Omega * 4}{0,7V} - 75\Omega = 1,34k\Omega \tag{4.7}$$

$$Bit7 : R_0 = \frac{3,3V * 75\Omega * 2}{0,7V} - 75\Omega = 632\Omega \tag{4.8}$$

Man erkennt, dass sich die Widerstandswerte ungefähr verdoppeln, also wurden nun passende Widerstände aus einer E-Reihe ausgewählt. Diese Auswahl wurde durch das Sortiment des Lieferanten etwas eingeschränkt, aber mit den Widerständen in Tabelle 4.1 konnten vernünftige Ergebnisse erzielt werden.

Tabelle 4.1.: R0-R7

R0	68,1k
R1	39,0k
R2	18,0k
R3	9,09k
R4	4,30k
R5	2,21k
R6	1,10k
R7	560Ω

4.2. Simulation

Die Schaltung wurde mit dem Simulationsprogramm LTspice der Firma Linear Technology simuliert. In der Simulation zählen Spannungsquellen digital von 0 bis 255 und liefern das Ergebnis in Abbildung 4.1. Die extremen Ausschläge nach unten hängen mit der Simulationsmethode zusammen und haben nichts mit dem realen Ergebnis zu tun. Ansonsten ist gut zu erkennen, dass der gesamte Spannungsbereich von 0,0V bis 0,7V durchlaufen wird. Größtenteils ergibt sich eine einheitliche Treppenform - bis auf ein paar vereinzelte Stellen, was auf die Widerstandswahl zurück gehen. Verwendet man einheitlich Vielfache von 560Ω, zeigt die Simulation absolut äquidistante Spannungspegel. Dies war leider bei dem verwendeten Lieferanten nicht möglich.

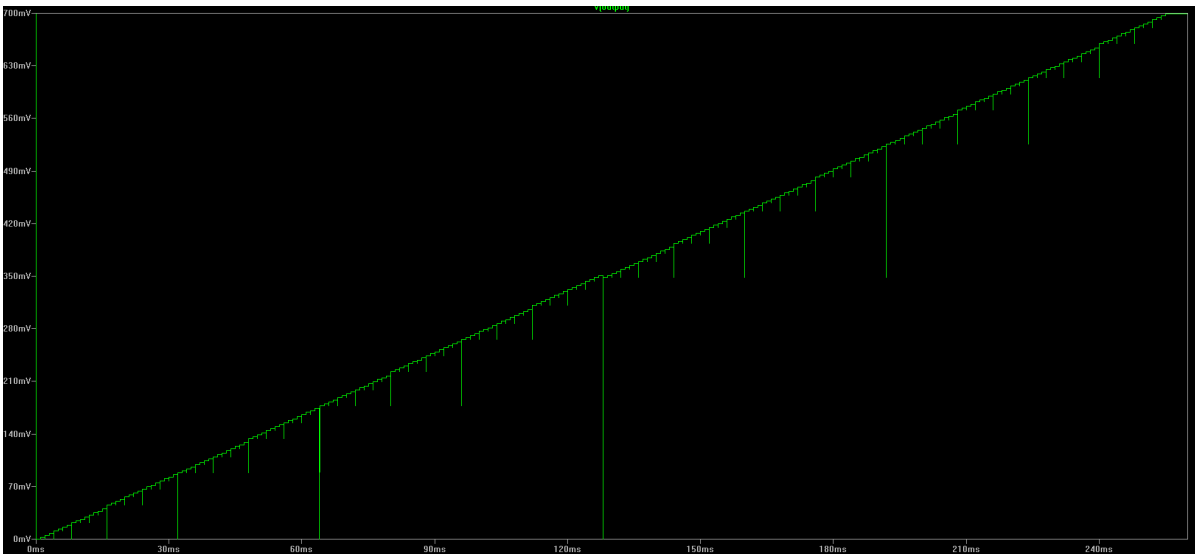
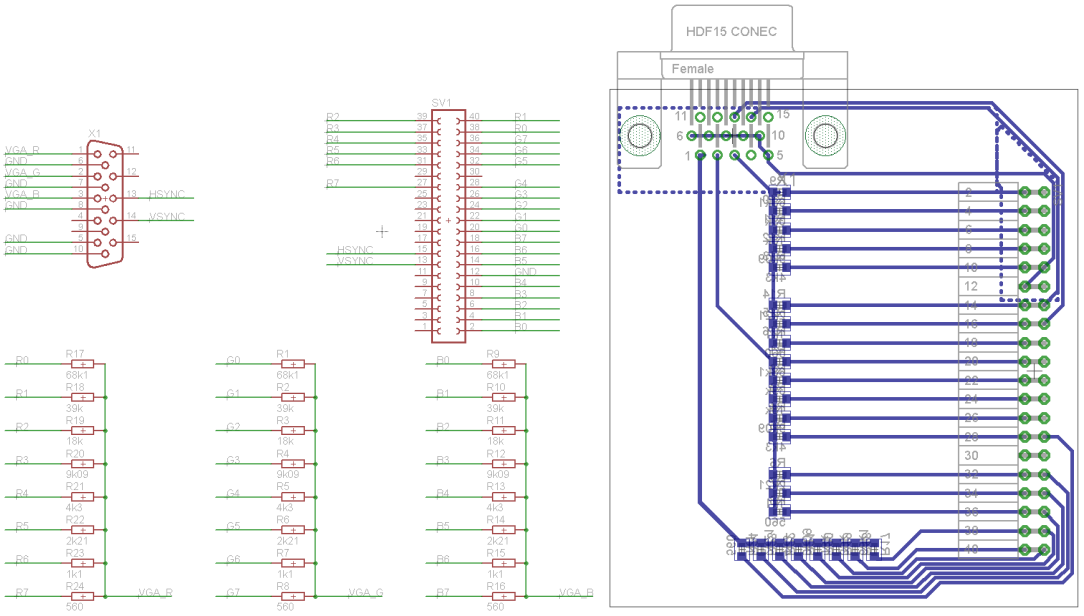


Abbildung 4.1.: Simulationsergebnis

4.3. Platinen-Design

Anhand der Simulationsergebnisse wurde nun die Schaltung mit dem Layout-Programm Eagle entworfen und ein Board-layout erstellt. Es ist so entworfen, dass es später senkrecht auf dem Board im 40-Pin-Header *GPIO1* steckt und so noch Platz für Erweiterungen im *GPIO0* bleibt.



(a) Schaltplan (b) Platinenlayout

Abbildung 4.2.: Pictures of animals

4.4. Messergebnisse

Leider ist die bestellte Platine nicht rechtzeitig bei mir eingetroffen und es sind so keine Messungen vor der Abgabe möglich. Die Platine wird nachgereicht.

5. Einbindung in Bildverarbeitung von C. Paa

Diese Arbeit wurde parallel bzw. als Grundlage für das Projekt von Herrn Paa entwickelt. Seine Arbeit beschäftigt sich mit der Implementierung von Bildbearbeitungsalgorithmen im FPGA und einem passenden Frontend dazu. Zur Kommunikation mit dem PC wurde dem DE0-Board ein 9-polige DSUB-Buchse aufgelötet und mit dem RS232-Port eines PCs verbunden. Als Frontend dient ein Matlab-GUI (Abb. **??**), über welches Bilder auf das Board übertragen und bearbeitet werden können.

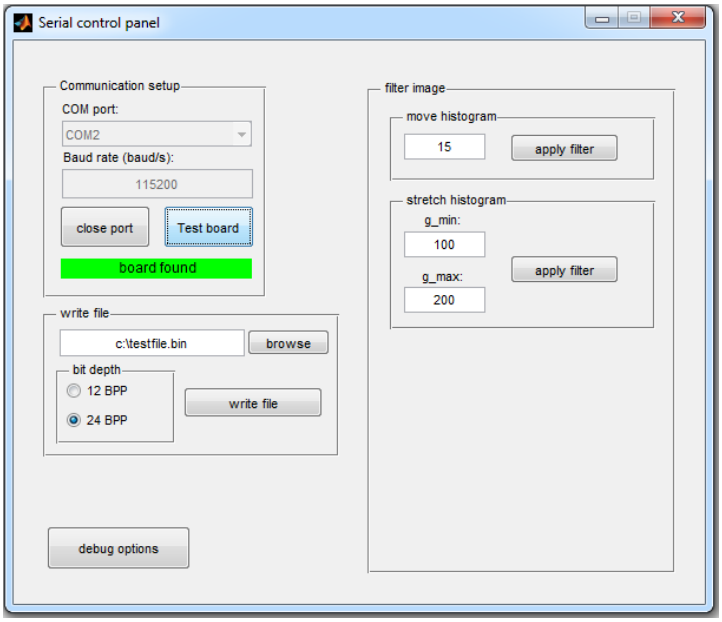


Abbildung 5.1.: Matlab-GUI

Meine Arbeit setzt an der Stelle ein, die übertragenen Daten im SDRAM abzuspei-

chern und die für die Bildverarbeitung benötigten Daten bereit zu stellen. Dafür haben wir die Module von Herrn Paa zur RS232-Kommandointerpretierung und Bildverarbeitung, sowie meinen SDRAM-Controller in einem Modul kombiniert, um uns einen weiteren Kommunikationskanal zwischen den Komponenten zu ersparen.

Das gesamt Projekt ist in der Lage eine Bitmap mit bereits entferntem Header im SDRAM des DE-Boards abzulegen und anschließend zu bearbeiten. Zur Verfügung stehen eine Option zur Verschiebung des Histogramms, was eine Veränderung der Helligkeit ermöglicht. Des Weiteren besteht die Möglichkeit der Spreizung des Histogramms, also die Festlegung eines neuen Maximum und Minimum in jedem Farbbereich und einer darauf folgenden Neuberechnung eines jeden Farbwertes. Dies führt zu einer ausgeglicheneren Farbverteilung, bei richtiger Anwendung.

6. Zusammenfassung

In meinem Teil des Projektes habe ich eine VGA-Schnittstelle mit einer Auflösung von 1024x768 Pixeln und einer Bildwiederholrate von 60 Hertz entwickelt. Des Weiteren wurde ein SDRAM-Steuerung für den Zentel SDRAM entwickelt, welcher auf dem Terasic DE0-Board verbaut ist. Sämtliche entwickelten Komponenten arbeiten mit einer Farbauflösung von 24 Bit und müssen nur an entsprechend auflösende DACs angeschlossen werden. Diese werden in Form einer Erweiterungsplatine noch nachgeliefert.

Quellenverzeichnis

[1] devbisme *VGA - the Rest of the Story* 11.06.2011 URL: ''http://devbisme.webfactional.com/sites/default/files/users/devbisme/blog/images/VGA/app001_2.png''

[2] Zentel Datasheet *A3V64S40ETP* Revision 1.2, March 2010 URL: ''http://www.zentel.com.tw/upload/Automotive/A3V64S40ETP_I%20ver_v.1.2_Zentel.zip''

A. Anhang

Inhalt der CD-ROM

/Datenblätter

DE0_Schaltplan.pdf	Schaltplan des Terasic DE0 Boards
DE0_User_manual.pdf	DE0 Datenblatt
Zentel_SDRAM.pdf	Zentel SDRAM Datenblatt

/Dokumentation

Dokumentation.tex	Latex Basis-File
Dokumentation.pdf	Dokumentation im .pdf-Format
/images/...	Unterordner für die in der Dokumentation verwendeten Bilder

/Eagle-Files

VGA_Erweiterung.brd	Board-Layout der Erweiterungsplatine
VGA_Erweiterung.sch	Schaltplan des Erweiterungsboards

/LTspiceSimulationen

4bit.asc	Simulation des DACs auf dem DE0 Board
8bit.asc	Simulation des entworfenen DACs
8bit_optimal.asc	Entwurf für den Fall einheitlicher Vielfacher von R7

/MatlabGUI

graka_gui.m	Matlab GUI

/LTspiceSimulationen

4bit.asc	Simulation des DACs auf dem DE0 Board
8bit.asc	Simulation des entworfenen DACs
8bit_optimal.asc	Entwurf für den Fall einheitlicher Vielfacher von R7

/Testbilder

futurama.bin	Futurama-Bitmap ohne header
futurama.bmp	Futurama-Bitmap
Schloss.bin	Schloss-Bitmap ohne header
Schloss.bmp	Schloss-Bitmap

/VHDL-Code

cmd_dec_sdram_cntrl.vhd	RS232-Komandodecoder und SDRAM-Controller
dbg_decoder.vhd	Debug-Modul
graka_pack.vhd	Funktionspackage
PLL.vhd	166MHz PLL
PLL65.vhd	65MHz PLL
RS232.vhd	RS232 Schnittstelle
single_port_ram.vhd	Single Port Ram Komponente
ss_decoder.vhd	Decoder für Siebensegment Anzeige
VGA.vhd	VGA-Schnittstelle
VHDL_graka.qpf	Quartus Projekt-File