TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

Institut für Technische Informatik – Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Beleg

zur Lehrveranstaltung

Schaltkreis- und Systementwurf

Demonstration graphischer Anwendungen auf FPGAs

Andrej Olunczek
andrej@olunczek.de
Matrikel-Nr.: 3066276

Betreuer:	Dipl.-Inf. Th. Preußer
Verantwortlicher Hochschullehrer:	Prof. Dr. R. G. Spallek
vorgelegt am:	26. April 2011

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabe

Aufgabe war es, eine einfache Anwendung auf einem FPGA zu implementieren. Ich habe hier eine einfache graphische Anwendung implementiert, die es erlaubt, auf einem VGA-kompatiblen Monitor mit Hilfe einer PS2-Maus verschiedene Objekte auf eine Zeichenebene zu zeichnen, ähnlich wie es zum Beispiel in der Anwendung 'paint' unter Windows funktioniert.

Das ganze dient zur Demonstration der Interaktion zwischen Maus-Eingabe, externem RAM und VGA-Ausgabe. Des Weiteren ist der Bresenham-Algorithmus in zwei Varianten implementiert, um beliebige Linien und Ellipsen zu rastern. Es ist auch ein einfaches dreiteiliges Maus-gesteuertes Menü realisiert, um zwischen verschiedenen Zeichenobjekten, wie Linien, Rechtecke oder Ellipsen, Farben und Zeichenstilen, beziehungsweise Pinselformen, zu wechseln.

Als Hardware dient ein Spartan-3E Starter Kit Board mit einem XC3S500E der Firma Xilinx als FPGA. Extern angeschlossen ist ein VGA-kompatibler Monitor und eine beliebige PS2-Maus. Außer dem FPGA werden auf dem Board lediglich der DDR-RAM-Baustein, ein Taster als Resetschalter und der 50-MHz-Taktgeber genutzt.

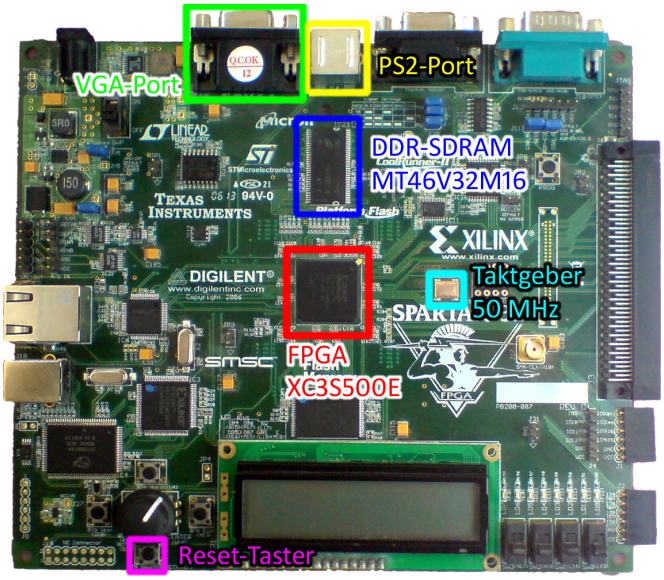


Abbildung 1: Das verwendete Xilinx FPGA Starter Kit Board

2 Aufbau

2.1 Allgemein

Zentraler Punkt der Anwendung ist der RAM Arbiter, der zuordnet, welches Modul zu welcher Zeit auf den Speicher zugreift. Dabei gibt es einen Leseport für das VGA-Modul, dass die Daten aus dem Speicher ausliest, um sie ohne Umwege auf dem Monitor graphisch darzustellen. Natürlich müssen dazu erst einmal die Daten in den Speicher kommen. Dazu gibt es drei Schreibports, einen für die Maus und zwei für die eigentliche Programmlogik. Das Mausmodul schreibt immer den aktuellen Mauszeiger in den Speicher. Die Programmlogik ist für den ganzen Rest zuständig.

Damit Objekte, die gerade gezeichnet werden und noch über die Zeichenfläche gezogen werden, den bestehenden Inhalt nicht wegwischen, werden diese, solange sie nicht fest sind, in eine temporäre Ebene gespeichert und erst im Anschluss in die permanente Ebene übernommen. Aus diesem Grund braucht die Programmlogik die beiden Schreibports.

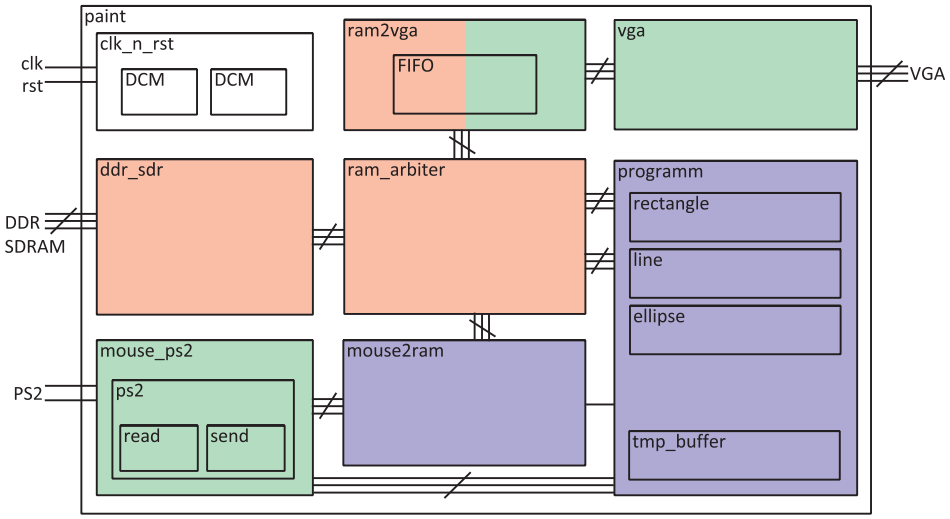


Abbildung 2: Aufbau der Schaltung; rot: 100MHz; blau: 50MHz; grün: 25MHz

2.2 Taktgenerierung

Es gibt ein zentrales Modul für die Taktgenerierung, dass gleichzeitig auch mit das Resetverhalten steuert. Die Anwendung arbeitet mit drei verschiedenen Taktraten, 25 MHz, 50 MHz und 100 MHz. Dabei arbeiten der RAM, der RAM-Controller und der RAM-Arbiter mit 100 MHz. Die eigentliche Programmlogik und die Steuerung der Mausdarstellung arbeiten mit 50 MHz, der PS2-Maus-Controller und die VGA-Ausgabe mit 25 MHz. Das ganze ist auch unter Abbildung 2 dargestellt.

Für die Taktgenerierung werden, wie in Abbildung 3 zu sehen, zwei DCMs (Digital Clock Manager) genutzt. Für die 100 MHz wird für die DDR-RAM-Ansteuerung auch noch ein zweites Taktsignal mit einer Phasenverschiebung von 270 Grad benötigt, dafür ist der zweite DCM nötig, da ein DCM bei der Taktverdoppelung des Eingangssignal nicht gleichzeitig eine Phasenverschiebung von 270 Grad realisieren kann.

Eine Resetlogik gibt das Resetsignal erst frei, wenn die DCMs stabile Taktsignale liefern. Dadurch wird verhindert, dass andere Module falsche Berechnungen anstellen oder ungültige Signale liefern, weil die Taktflanken unkontrolliert oder willkürlich auftreten. Dazu wird das locked-Signal der DCMs genutzt, die das Signal erst auf 'high' setzten, wenn die Taktsignale stabil und zueinander Phasensynchron sind.

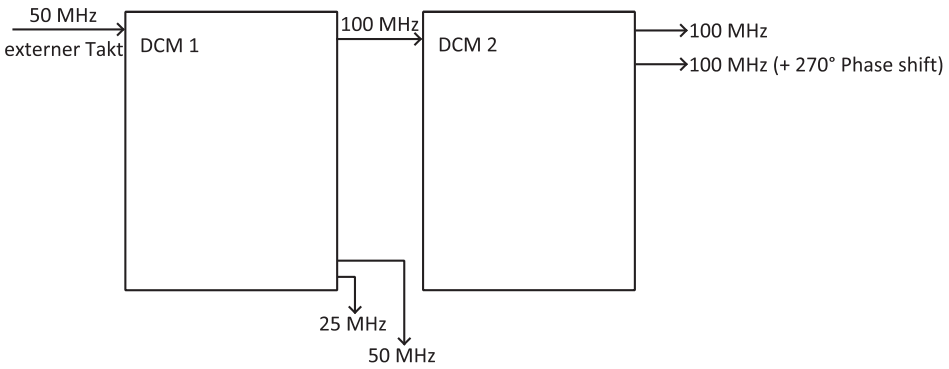


Abbildung 3: Zusammenschaltung der DCMs zur Taktgenerierung

Tabelle 1: Aufbau des Bytes, das ein Pixel speichert

7	6	5	4	3	2	1	0
ist Maus	rot	grün	blau	ist temporär	rot	grün	blau
Maus	temporäre Ebene				permanente Ebene		

2.3 RAM

Als RAM ist das 64-MByte-DDR-RAM-Modul MT46V32M16 der Firma Micron im Einsatz. Da dieses in der Ansteuerung und vom Timing etwas komplexer ist, ist ein eigener DDR-RAM-Controller notwendig. Da das Projekt keine eigene Entwicklung eines solchen Controllers werden soll, nutze ich hier den fertigen Controller „DDR SDRAM Controller Core“ von OpenCores. Diesen habe ich insoweit modifiziert, dass die Taktgenerierungs- und Resetlogik entfernt ist, da diese Funktionen durch das eigene Takt- und Resetmodul gestellt werden.

Dieser RAM-Controller ist im Weiteren mit dem RAM-Arbiter verbunden, der das Herzstück der gesamten Schaltung ist. Dieser verwaltet, wann welches der anderen Module, VGA, Maus oder Programmlogik auf den RAM zugreifen darf. Dabei wird der VGA-Lese-Port mit oberster Priorität behandelt, da die VGA-Ausgabe und Darstellung zu jedem Zeitpunkt gewährleistet sein muss, um keine Löcher oder Verzerrungen im Bild zu haben. Die nächst niedrigere Priorität hat das Modul, das für die Anzeige des Mauszeigers zuständig ist, um die Latenz zwischen Mausbewegung und Darstellung gering zu halten. Das dritte Modul, das auf den RAM zugreift, ist die eigentliche Programmlogik. Diese benötigt zwei Schreibports, da sie in zwei Zeichenebenen, einer temporären und einer permanenten, arbeitet. Dabei hat der temporäre Schreibport zwar die höhere Priorität, was aber keinen weiteren Einfluss hat, da die Programmlogik nie beide Ports gleichzeitig ansteuert.

Der RAM-Controller ist so gestaltet, dass er immer im Burst-2-Modus arbeitet, also immer zwei Wörter schreibt oder liest, wobei ein Wort immer aus 16 Bit besteht. Die Anwendung braucht pro Pixel ein Byte, also acht Bit, die in

Tabelle 2: Aufbau der Adresse, um ein Pixel zu speichern oder zu lesen

Bit 19 .. 10	Bit 9 .. 0
y-Koordinate des Pixels	x-Koordinate des Pixels

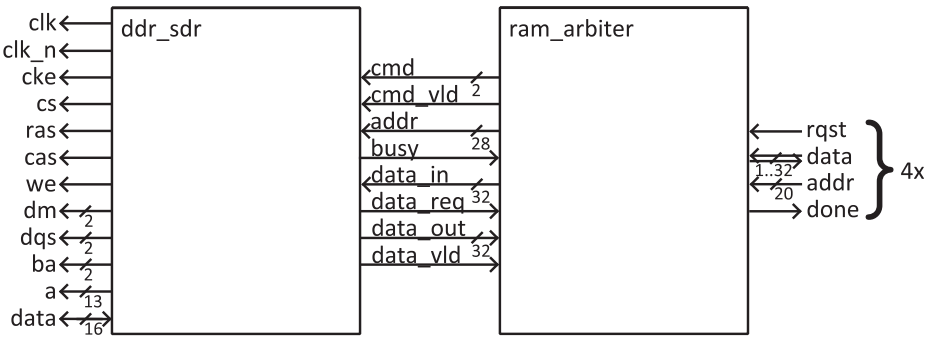


Abbildung 4: Schnittstellen der RAM Module

Tabelle 1 detailliert aufgelistet sind. Es werden also immer vier Pixel gleichzeitig angesteuert. Das ist beim Lesen für die VGA-Ausgabe durchaus praktisch, bereitet beim Schreiben einzelner Pixel aber einen Mehraufwand. Es wird also immer ein Viererblock Pixel gelesen, das aktuelle Pixel geändert, und im Anschluss der geänderte Viererblock wieder zurück geschrieben. Dabei wird von dem jeweiligen Pixel nur der Teil geändert, der je nach Schreibport gerade relevant ist, also das Mausbit, die temporäre Ebene oder der permanente Teil.

Zur Adressierung der Pixel werden die y-Koordinate und die x-Koordinate, wie in Tabelle 2 dargestellt, zusammengefügt. Die x-Koordinate stellt dabei die niederwertigen Bits, damit beim Lesen über den VGA-Port immer vier benachbarte, in direkter Folge darzustellende, Pixel gelesen werden.

Wie in Abbildung 4 dargestellt, werden die Schnittstellen der RAM-Module zu den jeweiligen Lese-/Schreibmodulen hin auch immer spezialisierter aber dafür weniger komplex.

2.4 Maus

Die Maussteuerung besteht aus zwei Modulen. Zum einen ein PS2-Maus-Controller, zum anderen ein Modul, das in Abhängigkeit der aktuellen Mauszei-

Tabelle 3: Aufbau der Kommunikation über PS2

0	1	2	3	4	5	6	7	8	9	10
Startbit	Datenbits 0 ... 8								Paritätsbit	Stopbit

Tabelle 4: Aufbau des von der Maus gesendeten Datenpaketes

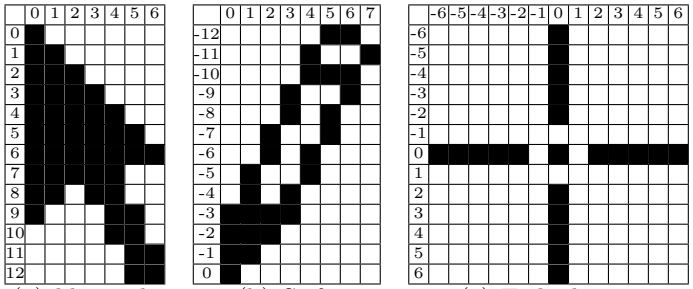
	0	1	2	3	4	5	6	7
Byte 1	y-Überl.	x-Überl.	y-Vorz.	x-Vorz.	immer 1	mittlere Taste	rechte Taste	linke Taste
Byte 2	x-Bewegung							
Byte 3	y-Bewegung							

gerposition und einem von der Programmlogik übermittelten Toolstatus einen Mauszeiger in den Speicher schreibt. Das PS2-Maus-Modul enthält ein PS2-Untermodul, das wiederum aus einem Lese- und einem Schreibmodul besteht.

Das elektrische Interface besteht aus zwei open-collector-Signalen. Das eine Signal dient der Taktübertragung, wobei der Takt nur von der Maus generiert wird. Die andere Leitung dient der seriellen Übertragung der Daten, wobei immer acht-Bit-Wörter, also ein Byte, am Stück übertragen werden. Neben den acht Datenbits werden auch ein Start-, ein Stopp- und ein Paritätsbit übertragen. In Tabelle 3 ist das Datenwort im Detail dargestellt. Das Paritätsbit ist immer dann gesetzt, wenn die Anzahl der auf '1' gesetzten Datenbits eine ungerade Zahl ist. Die Lese- und Sendeeinheiten überwachen und steuern die Datenübertragung und geben jeweils ein Erfolgs- oder Fehler-Statusbit aus.

Die Lese- und Sendeeinheiten sind beide direkt vom Mauscontroller umgeben, der die Kommunikation mit der Maus steuert, in welcher Reihenfolge was gesendet oder gelesen wird. Der Controller sorgt dafür, dass die Maus ordnungsgemäß initialisiert wird und gibt seinerseits immer die aktuellen Mauskoordinaten und den Maustastenstatus aus. Die Maus sendet dazu immer ein Paket aus drei Byte, dass den Status der Tasten und die Bewegung der Maus

Abbildung 5: Mauszeiger



(a) klassisch (b) Stift (c) Fadenkreuz

seit dem letzten Packet beschreibt. Der genaue Inhalt dieses Datenpaketes ist in Tabelle 4 beschrieben.

Das zweite Modul schreibt in Abhängigkeit der Koordinatenausgabe des Mauscontrollers und dem Toolstatus aus der Programmlogik den Mauszeiger in den Speicher. Dazu sind drei verschiedene Zeigergraphiken innerhalb des Moduls definiert (siehe Abbildung 5). Es gibt den klassischen Mauszeiger für die Menüführung, wenn die Maus außerhalb der Zeichenfläche (siehe Abbildung 6) ist. Wenn die Maus innerhalb der Zeichenfläche ist, wird entweder ein kleiner Stift als Zeiger verwendet, wenn das Freihand-Zeichentool aktiv ist, ansonsten ein kleines Fadenkreuz zur besseren pixelgenauen Ansteuerung. Ändern sich nach einer Bewegung der Maus die Mauskoordinaten, wird der alte Zeiger von der alten Position gelöscht und der neue Zeiger an die neue Position gezeichnet.

2.5 VGA

Auch die VGA-Ausgabe besteht aus zwei Teilen. Dass ist zum einen ein Modul, das die Pixel aus dem Speicher holt und einen zentralen FIFO enthält. Zum anderen ein Modul, das dafür zuständig ist, dass für die Monitoransteuerung die benötigten Signale generiert werden. Ein FIFO ist ein Zwischenspeicher, der auf der einen Seite Daten einliest solange er nicht voll ist und auf der anderen Seite diese Daten nach Bedarf in selber Reihenfolge wieder ausgibt. Es werden für den analogen VGA-Anschluss fünf Signale benötigt, horizontale und vertikale Synchronisation und jeweils ein Signal für die Farben Rot, Grün und Blau. Die beiden Synchronisationssignale sind so konfiguriert, dass

Tabelle 5: Übersicht der möglichen Farben

	rot	grün	blau
	0	0	0
	1	0	0
	0	1	0
	0	0	1
	0	1	1
	1	0	1
	1	1	0
	1	1	1

das erzeugte Bild eine Auflösung von 640 mal 480 Pixeln hat und mit einer Bildwiederholfrequenz von 60 Hz angezeigt wird.

Das FIFO-Modul sorgt dafür, dass zur richtigen Zeit immer das aktuelle Pixel-Byte (siehe Tabelle 1) an das Ausgabemodul gesendet wird. Dazu wird der FIFO, solange er nicht voll ist, immer mit den jeweils nächsten vier Pixeln befüllt, die dafür aus dem RAM gelesen werden. Der FIFO ist 32 Bit breit, da immer 32 Bit, also vier Pixel, gleichzeitig aus dem RAM gelesen werden. Die Schreibtiefe beträgt 16 Wörter. Da der FIFO im First-Word-Fall-Through-Modus arbeitet, also immer das nächste zu lesende Wort ungefragt im Ausgangsregister liegt, können sogar 17 Wörter gespeichert werden, es liegen also maximal bis zu 68 Pixel gleichzeitig im FIFO. Der FIFO wird immer dann durch einen internen Reset mit der Ausgabe synchronisiert, wenn diese das aktuelle Bild vollständig an den Monitor gesendet hat. In diesem Fall fängt der FIFO schon an, die neuen Bilddaten einzulesen, während die horizontalen und vertikalen Synchronisationssignale noch außerhalb des Bildbereiches arbeiten. Die neuen Pixeldaten werden erst dann an die VGA-Ausgabe weitergereicht werden, wenn diese wieder anfängt, dass nächste Bild auszugeben.

Der FIFO-Puffer-Speicher ist nicht zuletzt dadurch notwendig, dass nicht vorhergesagt werden kann, wie lange ein Lesezyklus im RAM dauert, da immer mal wieder Refresh-Zyklen dazwischen kommen, um den Speicherinhalt zu erhalten, oder Precharge-Befehle zwischengeschaltet werden, wenn sich bei der Adressierung die interne Speicherbank oder -zeile ändert. Damit in diesem Fall keine Löcher im Bild entstehen, werden ausreichend viele Pixel im Zwischenspeicher vorgehalten. Der FIFO nutzt den so genannten distributed RAM, der die Lookup-Tabellen als Speicher nutzt, da der gesamte Block-RAM des FPGAs später noch für die Programmlogik gebraucht wird.

Ist das Bit für die temporäre Ebene gesetzt (siehe Tabelle 1), werden die RGB-Werte dieser Ebene genutzt, ansonsten die Werte der permanenten Ebene. Ist das Mausbit gesetzt, werden genau die jeweils negierten Werte dessen angezeigt, die sonst zu sehen wären. Der Mauszeiger ist also immer in invertierten Farben dargestellt, und damit auf jedem Untergrund zu sehen.

3 Programmlogik

3.1 Allgemein

Im Programmmodul ist das eigentliche Programm enthalten, also die graphische Oberfläche, die in Abbildung 6 dargestellt ist, die Menüs und das ganze Verhalten in Bezug auf die Mausinteraktion. Das Programm hat drei unabhängige Menüs zur Auswahl des Zeichenwerkzeuges, der Zeichenfarbe und des Zeichenstiles. Zeichenwerkzeuge sind ein Freihandwerkzeug, lineare Linien, leere und gefüllte Rechtecke sowie leere und gefüllte Ellipsen. Gezeichnet werden kann in den acht verschiedenen Farben, die in Tabelle 5 dargestellt sind. Zusätzlich wird in einem bestimmten Zeichenstil gezeichnet, was im Grunde der Zeichenspitze des verwendeten virtuellen Stiftes entspricht, also in verschiedenen Strichbreiten oder auch schräge kurze Linien. In Abbildung 7 sind die

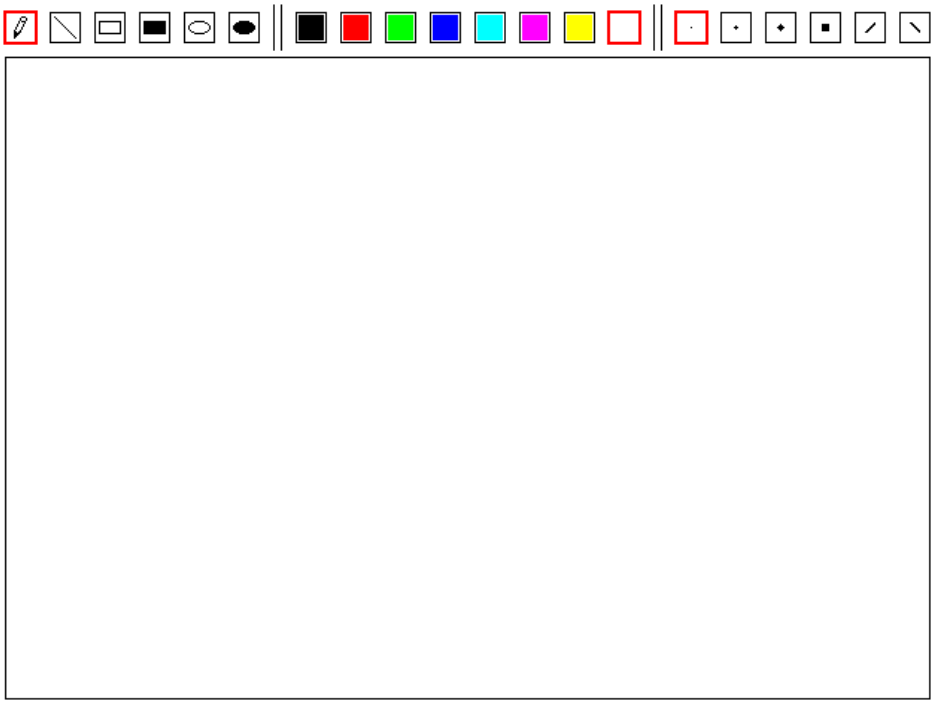
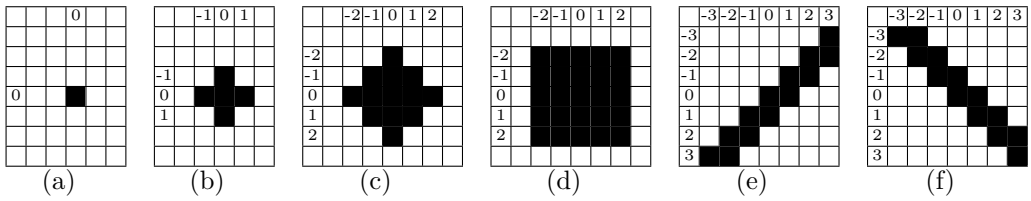


Abbildung 6: Die Benutzeroberfläche; oben das dreiteilige Menü mit den rot umrandeten Voreinstellung, die große Zeichenfläche liegt direkt untendrunter

Abbildung 7: Zeichenstile



Zeichenstile detailliert abgebildet.

Das Programmmodul enthält für die jeweiligen Objekte, also Linien, Rechtecke und Ellipsen, jeweils ein Untermodul, das nacheinander die Pixelkoordinaten des Objektes berechnet und ausgibt. Zu diesen Koordinaten werden jeweils alle Pixelkoordinaten des jeweils aktuellen Zeichenstils hinzuaddiert. Das Gesamtergebnis wird dann in den RAM gespeichert.

3.2 Initialisierung & Datenstruktur

Die Bildschirmelemente der graphischen Oberfläche, wie Rechtecke, Linien, Ellipsen und kleine Symbole sind in Arrays und Records gespeichert. Diese können dann in Schleifen nacheinander ausgelesen und in den RAM geschrieben werden. So ist die Struktur und Menüführung recht übersichtlich gehalten und leicht modifizierbar. Ein Beispiel dazu ist im Quellcodeausschnitt 1 zu sehen. Während der Initialisierung werden diese Datenstrukturen schrittweise ausgelesen und die entsprechenden Objekte auf die Oberfläche, also in den RAM, geschrieben.

Damit die Objekte, die im laufenden Betrieb gezeichnet werden, nicht das bestehende Bild wegwischen, solange sie noch nicht fest sind, werden sie während des Zeichnens und der Bewegung der Maus erst in eine temporäre Ebene im RAM gespeichert, und erst beim Loslassen der Maustaste fest in den permanenten Bildspeicher übertragen. Da das permanente Ändern des RAM-Inhalts sehr zeitaufwendig ist und durch das ständige Löschen und Neuzeichnen beim Bewegen der Maus während des Zeichnens ein Flimmern auf dem Monitor entsteht, wird der Inhalt in einem Puffer zwischengespeichert, dessen Inhalt erst in den RAM übertragen wird, wenn nach einer Mausbewegung das neue Bild fertig berechnet ist.

Die Zeichenfläche ist 619 mal 429 Pixel groß. Für die Adressberechnung des

Quellcode 1: Beispiel für Datenstrukturen von Objekten, Menüs, etc.

```
-- Beispiel für die Definition einer Zeichenspitze
type tPixel is record
                 x : integer;
                 y : integer;
               end record;
type tPixelCloud is array(positive range 1 to 37) of tPixel;
type tPixelObject is record
                       pixel : tPixelCloud;
                       count : positive;
                     end record;
-- [...]
constant cSmallBall : tPixelObject := (count => 5,
                                       pixel => (( 0, -1),
                                                 (-1,  0),
                                                 ( 0,  0),
                                                 ( 1,  0),
                                                 ( 0,  1),
                                                 others => (0, 0)));


-- Beispiel für die Definition eines Menüs, hier das Zeichenstil-Menü
type tStyleBox is record
                    x1    : natural;
                    y1    : natural;
                    x2    : natural;
                    y2    : natural;
                    style : tPixelObject;
                  end record;
type tMenuStyleBoxes is array(positive range <>) of tStyleBox;
-- [...]
constant cMenuStyleBoxes : tMenuStyleBoxes(1 to 6) := ((460,  10, 480,  30, cDot),
                                                       (490,  10, 510,  30, cSmallBall),
                                                       (520,  10, 540,  30, cBigBall),
                                                       (550,  10, 570,  30, cSquare),
                                                       (580,  10, 600,  30, cSlash),
                                                       (610,  10, 630,  30, cBackslash));
```

Pufferspeichers werden die beiden Koordinatenwerte eines Pixels konkateniert, wie es auch für die Adressierung im DDR-RAM gemacht wird (siehe Tabelle 2). Binär umgerechnet ist der Maximalwert innerhalb der Zeichenfläche für x = 618 = 1001101010 und für y = 428 = 110101100. Bilden für die Adresse die y-Koordinaten die höherwertigen Bits, wäre die maximal mögliche Adresse 438890. Wenn die höherwertigen Bits durch den x-Anteil gestellt werden, wäre der Maximalwert 316844. Demzufolge ist der zweiten Variante der Vorzug zu geben, was genau andersherum ist, als bei der Adressierung des DDR-RAM. Da pro Pixel nur ein Bit benötigt wird, um festzuhalten, ob er dann mit der aktuellen Farbe in den RAM geschrieben wird oder nicht, werden insgesamt 309,4 kBit als Puffer Speicher benötigt. Der FPGA hat für die Variante mit einem Bit pro Adresse 20 Blöcke zu je 16 kBit als Block-RAM, also insgesamt 320 kBit. Der temporäre Pufferspeicher benötigt also den kompletten Block-RAM.

3.3 Menü

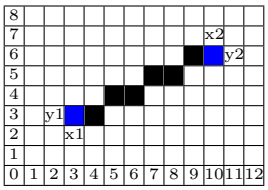
Wird die linke Maustaste gedrückt, wird überprüft, ob sich der Mauszeiger über der Zeichenfläche oder über einem der drei Menüs befindet, die in der Abbildung 6 im oberen Bereich zu sehen sind. Ist der Mauszeiger über der Zeichenfläche, wird entsprechend der Menüeinstellungen das betreffende Objekt gezeichnet. Wird die Maustaste über den Menüs gedrückt, wird zuerst erfasst, welches Menü betroffen ist. Im Anschluss wird in der Liste der Menü-Buttons des jeweiligen Menüs nachgeschaut, ob sich der Mauszeiger während des Drückens der Maustaste über einem der Buttons befand. Wenn dem so ist, wird gewartet, bis die Maustaste wieder losgelassen wird. Befindet sich dabei der Mauszeiger immer noch über demselben Button, wird der alte deaktiviert und der neue aktiviert. Ein Beispiel für die Definition eines Menüs ist im Quellcodeausschnitt 1 im unteren Teil zu sehen.

3.4 Linie

Zum Zeichnen von Linien gibt es ein Untermodul, dass aus den Eingangssignalen für zwei Koordinatenpunkte alle Zwischenpunkte ausgibt, die nötig sind, um eine durchgehende gerade Linie zu bilden. Über ein Startsignal wird aus den anliegenden Koordinaten eine neue Linie berechnet. Dabei wird über ein Next-Signal jeweils die Koordinaten des jeweils nächsten Pixels geladen. Über ein Busy-Flag wird angezeigt, ob noch weitere Pixelkoordinaten zu erwarten sind oder nicht.

Zur Berechnung der Koordinaten wird der Bresenham-Algorithmus genutzt. In der Grundvariante im ersten Quadranten, x2 > x1 und y2 > y1, Steigung kleiner als 1, geht man davon aus, dass man in jedem Schritt die x-Koordinate um 1 erhöhen kann. In mehr oder weniger regelmäßigen Abständen wird auch die y-Koordinate um 1 erhöht. Um festzustellen, wann in y-Richtung erhöht

Abbildung 8: Bresenham-Algorithmus für Linienrasterung



werden muss, wird eine Fehlervariable mitgeführt. Der Fehler beginnt bei der Hälfte von dx = x2 - x1. In jedem Schritt wird dy = y2 - y1 abgezogen. Fällt der Fehlerwert unter 0 wird y um ein Schritt erhöht und der Fehlerwert mit dem Aufaddieren von dx wieder über 0 gebracht. Durch das Verrechnen von dx mit dy bleibt der Anstieg der Linie erhalten und kommt beim Berechnen der Koordinaten auch im Punkt (x2, y2) an.

Um auch Linien in anderen Quadranten zu zeichnen, muss das Ganze nur mit entsprechend anderen Vorzeichen betrachtet werden. Dazu wird zu Beginn der Berechnung eine Fallunterscheidung vorgenommen, um zu bestimmen, in welchen Quadranten die Linie führt, und welche Vorzeichen bei der Berechnung zu nutzen sind.

Der Bresenham-Algorithmus ist von der Komplexität her sehr einfach und benötigt nur Addierer als arithmetische Funktionen. Damit ist er auch wie geschaffen für den Einsatz auf Hardwareebene.

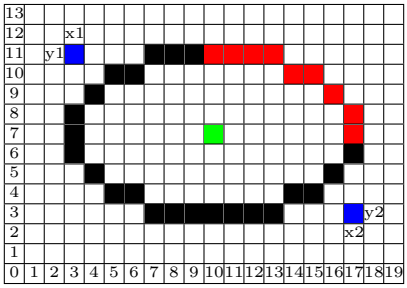
Auch das Freihandtool nutzt das Linienmodul. Es werden einfach, solange die linke Maustaste gedrückt ist, Linien aneinander gereiht. Dabei ist der Endpunkt einer Linie gleichzeitig Startpunkt einer neuen Linie, deren Endpunkt wiederum die neue Mausposition ist.

3.5 Rechteck

Auch die Berechnung von Rechtecken liegt in einem eigenen Modul. Hier ist die Berechnung aber sehr simpel. Begonnen wird in der linken oberen Ecke, dann wird die x-Komponente einfach solange erhöht, bis sie am rechten Rand des Rechteckes angekommen ist. Sofern nicht der untere Rand erreicht ist, wird die y-Komponente um 1 erhöht und am linken Rand wieder angefangen. Soll das Rechteck gefüllt sein, wird in jeder Zeile von links nach rechts in Einerschritten vorwärts gegangen, ansonsten wird direkt vom linken zum rechten Rand gesprungen, sofern es nicht der obere oder untere Rand ist.

Da jeder Pixel, der von dem Modul hier berechnet wird, noch einmal mit jedem Pixel des aktuellen Zeichenstiles verrechnet wird, ergeben sich im Maximalfall 25 Pixel die in den RAM geschrieben werden pro hier berechneter Koordinate. Für gefüllte Rechtecke bedeutet das aber, dass jeder Pixel im Inneren des Rechteckes bis zu 25 mal übereinander gezeichnet wird. Um dass zu verhindern, werden Füllpixel mit einem speziellen Fill-Flag gekennzeichnet. Ist dieses Flag vom Rechteck-Modul für das aktuelle Pixel gesetzt, wird es direkt in den RAM geschrieben, ohne vorher mit den Pixeln des Zeichenstiles

Abbildung 9: Bresenham-Algorithmus für Ellipsenrasterung



verrechnen zu werden.

3.6 Ellipse

Für die Berechnung der Ellipsen wird eine weitere Variante des Bresenham-Algorithmus genutzt. Als Ausgangswert dienen die Koordinaten eines Rechtecks (in Abbildung 9 blau dargestellt), in das die Ellipse eingepasst wird. Aus diesen Koordinaten werden der Mittelpunkt und die Halbachsen berechnet. Die Berechnungen des Fehlergliedes zur Steuerung, wann in x- oder in y-Richtung ein Schritt weiter gegangen wird, ist hier aber wesentlich komplizierter, als das bei der Berechnung der Linie der Fall ist. Es werden mehrere Multiplikatoren benötigt, die dazu aber auf dem FPGA schon in hoch optimierter Form vorhanden sind.

Berechnet wird aber lediglich ein Viertel der Ellipse (rot in Abbildung 9), die restlichen drei Viertel werden durch Spiegelung hinzugefügt. Den genauen Algorithmus habe ich dem deutschsprachigen Wikipedia-Artikel zum Bresenham-Algorithmus entnommen und entsprechend für VHDL umgearbeitet. Auch hier gibt es eine gefüllte Variante, die ein Fill-Flag nutzt, um Füllpixel zu kennzeichnen und unnötige Mehrarbeit am RAM zu vermindern.

4 Ergebnis

4.1 Auswertung

Die Anwendung lässt sich leicht direkt im Live-Betrieb testen. Dazu muss nur an das Board ein passender Monitor und eine passende Maus angeschlossen werden. Durch Ausprobieren aller Funktionen lässt sich die prinzipiell richtige Arbeitsweise aller Module nachweisen. Ein Beispiel ist auf dem Foto in Abbildung 10 zu sehen. Alle Objekte werden in korrekter Weise gezeichnet. Die Bildausgabe ist stabil und ohne Flimmern oder Verzerrungen.

Die Entwicklung des Projektes hat einige Eigenheiten des Xilinx Synthesetools gezeigt. Mit unter wurde so stark optimiert, dass Funktionen nicht mehr dem entsprachen, wie sie mit VHDL beschrieben wurden. Das Ellipsen-Modul hat zum Beispiel erst dann richtig funktioniert, nachdem in der Liste der zu initialisierenden Ellipsen „Blind-Ellipsen“ stehen, die nicht auf der Oberfläche zu sehen sind. Andere Probleme ließen sich alleine dadurch beheben, dass man manchen Signalen andere Bezeichnungen gegeben hat.

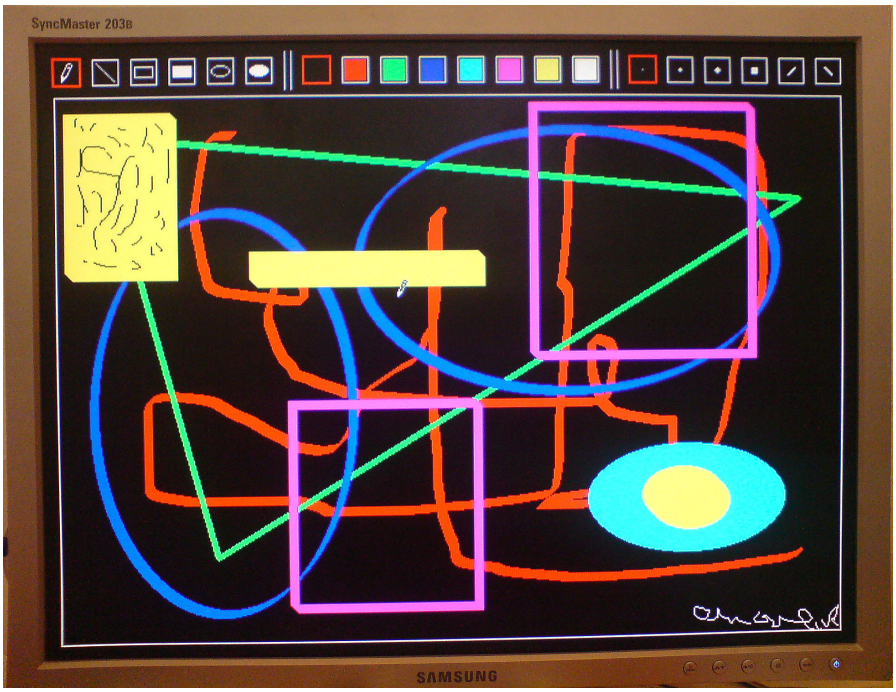


Abbildung 10: Die Anwendung in Funktion

Alles in allem ließ sich aber wunderbar zeigen, dass man mit relativ einfachen Mitteln auf einem FPGA graphische, menügeführte und mausgesteuerte Anwendungen entwickeln kann.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literatur

[1] Spartan-3E FPGA Starter Kit Board User Guide - UG230(v1.0)
Xilinx, 2006
http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[2] The PS/2 Mouse/Keyboard Protocol
Adam Chapweske, 2003
http://www.computer-engineering.org/ps2protocol/

[3] The PS/2 Mouse Interface
Adam Chapweske, 2003
http://www.computer-engineering.org/ps2mouse/

[4] 512Mb: x4, x8, x16 DDR SDRAM - Rev. A
Micron, 2007
http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf

[5] Spartan-3 Generation FPGA User Guide - UG331(1.0)
Xilinx, 2006
http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[6] Spartan-3E FPGA Family: Complete Data Sheet - DS312(3.4)
Xilinx, 2006
http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[7] Constraints Guide - ISE 7.1i
Xilinx, 2005
http://www.xilinx.com/itp/xilinx7/books/data/docs/cgd/cgd0001_1.html

[8] The VHDL Cookbook - First Edition
Peter J. Ashenden, 1990
http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf

[9] Bresenham-Algorithmus
Wikipedia, 2011
http://de.wikipedia.org/wiki/Bresenham-Algorithmus