OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

VGA-Grafikkarte auf einem FPGA

An der Fakultät für Informatik und Mathematik der
Ostbayerischen Technischen Hochschule Regensburg
im Studiengang
Technische Informatik

eingereichte

Projektarbeit in DAFP(Angewandte FPGA Programmierung)

Name:	Michael Braun
Matrikelnummer:	3113161
Name:	Korbinian Federholzner
Matrikelnummer:	3114621
Name:	Michael Schmidt
Matrikelnummer:	2907322
Erstgutachter:	Prof. Dr. Daniel Münch
Abgabedatum:	09.1.2020

Inhaltsverzeichnis

1. Einleitung

Die Grundaufgabenstellung des eigenen Projektes war die Ausgabe einer Digitaluhr auf einem Bildschirm. Dazu sollte auf einem FPGA eine kleine Grafikkarte erstellt werden. Dazu wurde stand ein Arty-7 zur Verfügung. Die Daten sollten dabei nach den Berechnungen über eine VGA-Schnittstelle an eine Monitor geschickt werden, welcher sie in der Auflösung 640x480 bei 60 Hz ausgeben sollte. Dadurch wurden diverse Zeiten vorgegeben die im Kapitel Timing genauer betrachtet werden. Diese sind für die richtige Synchonisation zwischen Bildschirm und FPGA verantwortlich, denn beim Schreibvorgang wird bei der ersten Pixelreihe links oben begonnen und nach einer gewissen Zeit mit der nächsten fortgesetzt. Vor Beginn und nach dem Ende des Schreibens einer Zeile muss dabei eine gewisse Zeit gewartet werden. Wird dabei die letzte Reihe erreicht, springt der „Schreibkopf" wieder zur ersten Reihe, wobei der Schreibvorgang nach einer festgelegten Zeitspanne mit einem neuen Bild begonnen wird.

2. Lösungsansätze

Unser Lösungsansatz für die Aufgabenstellung beeinhaltet eine Dreiteilung des Projekts, in Ausgabe, Speicher und Bilderstellung. Die Schnittstellen zwischen den drei Bereichen sind in zu sehen. Dabei beginnt der Ablauf dieser Grafikkarte in der Bilderstellungskomponente „IMG_CREATE". In dieser Komponente läuft die Uhr, deren Ausgabewerte durch „charmaps_ROM" in Bitmaps verändert werden. Diese werden danach in einzelne Bits aufgespalten und mit einer Farbkombination verknüpft und in den Speicher geschrieben.
Im Speicher werden die ankommenden Daten an der richtigen Adresse abgelegt und dann der Ausgabe zur Verfügung gestellt. Dabei wird aus den beiden Seitenzahlen der Pixel des Bildschirms auch noch die aktuelle Leseadresse bestimmt.
Die Komponente „Syncïst dabei für die Synchronisierung zwischen Bildschirm und Grafikkarte zuständig. Diese stellt dabei dem Speicher auch die Zähler zur Pixeladresse zur Verfügung. Zusatzlich benötigt die Schnittstelle zum Monitor dabei noch die Komponente „Blank_Check". Diese verhindert dass außerhalb der fest vorgegebenen Schreibgrenzen Daten an den Bildschirm gesendet werden. Dazu überprüft die Komponente die Zeitintervalle und setzt wenn nötig den Ausgang auf „0".

3. Bilderstellung

von Michael Schmidt

3.1. Überblick

Der grundsätzliche Code für eine Uhr wurde zur Verfügung gestellt. Lediglich die Umformung in passende Zeichen, die in den Speicher geschrieben werden, wurde implementiert. Dabei werden die Zeichen in ASCII-Code von der Komponente „CLOCK_MACHINE" zur Verfügung gestellt. Diese werden dann basierend auf Zeichen und Zeile in eine Adresse umgewandelt, die an die Komponente „charmaps_ROM" übergeben werden. Dort wird dann das richtige Byte dazu ausgewählt. Dieses Byte zeigt durch „1" an das dieser Pixel gesetzt wird und bei „0" nicht. Dieses Byte wird dann in einzelne Bits/Pixel zerlegt und mit der Farbauswahl und einer Adresse, ebenfalls basierend auf Zeichen und Zeile an den Speicher übergeben.

3.2. Lösungsansätze

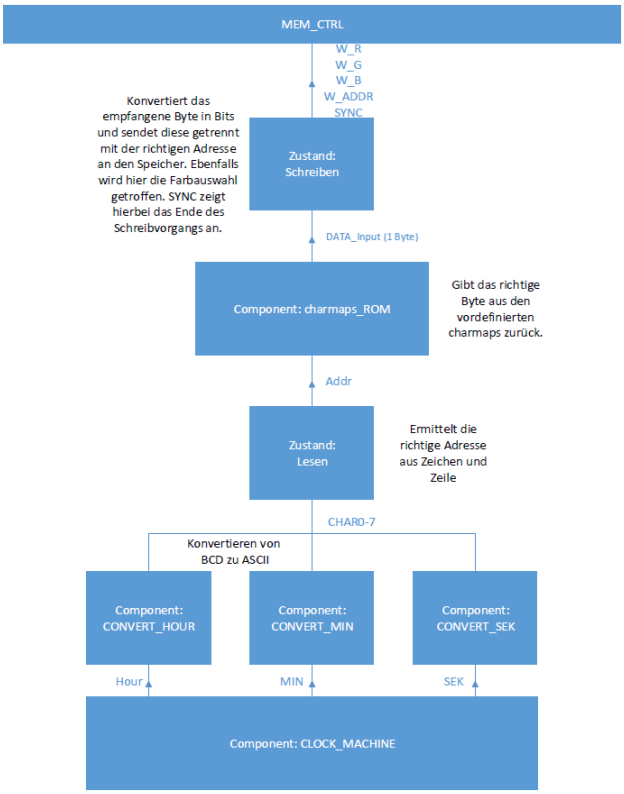

Abbildung 1: Schematische Darstellungder Komponente IMG_CREATE

Durch die Vorgabe der anderen Komponenten war bereits ein gewisser Weg vorgegeben. Einige Unterschiede könnten lediglich in der Programmierung entstehen. Eine Trennung in zwei Prozesse ist hierbei möglich, da zwei grundsätzliche Arbeitsschritte zu erledigen sind. Dabei würde der erste Prozess das Ansprechen der Komponenten „charmaps_ROM" übernehmen und der zweite das Weiterleiten der Farben mit der richtigen Adresse an den Speichercontroller. Allerdings ergibt sich dadurch die Problematik der Synchronisierung zwischen beiden Prozessen. Dies könnte eventuell noch über einen dritten Prozess, der beide synchonisiert gelöst werden. Eine weitere Möglichkeit ist eine State-Machine, bei der zwischen zwei Zuständen hin und her gewechselt wird. Diese Zustände würden dabei die jeweiligen Aufgaben der Prozesse übernehmen und nach Abschluss dieser in den anderen wechseln.

3.3. Umgesetzte Lösung

Der grundsätzliche Aufbau wird in der folgenden Grafik 1 anhand eines Blockdiagramms dargestellt.

Dabei wird die gesamte Einheit unterhalb von „MEM_CTRL" als „IMG_Creation" beschrieben. Diese besteht aus den Komponenten „CLOCK_MACHINE" , „CONVERT" und „charmaps_ROM", sowie einem Prozessmit zwei Zuständen in denen . Der Grundaufbau der Uhr beginnt dabei in der „CLOCK_MACHINE", welche die Zahlen der Uhr festlegt. Diese werden dann von der Komponente „CONVERT" in zwei getrennte Ziffern umgerechnet und anschließend in ASCII umgewandelt. Mit diesen Ausgangswerten beginnt der Prozess zu rechnen. Die State-Machine beginnt dabei im Zustand „Lesen" . Dort werden die letzten sechs Zeichen der Bitfolge mit 16 multipliziert. Anschließend wird noch ein Zähler für die aktuelle Zeile hinzuaddiert. Dadurch entsteht die richtige Adresse für „charmaps_ROM" .

Dies ist dadurch möglich, da die letzten sechs Bit des ASCII-Codes genau der Adresse der Charmaps entsprechen. Die genaue Charmapaufteilung ist in der Grafik 2 zu sehen. Jedes Kästchen/Zeichen besteht dabei aus 16 Zeilen mit einer Länge von einem Byte. Dabei wird immer zuerst die erste Zeile des ersten Zeichens, dann die zweite Zeile des ersten Zeichens bearbeitet. Somit wird erst ein vollständiges Zeichen geschrieben bevor ein zweites angefangen wird. Nach einem kompletten Durchlauf des ersten Zeichen werden alle Zähler, bis auf den der anzeigt bei welcher Ziffer bzw. Doppelpunkt man sich befindet, zurückgesetzt.

Nachdem eine Zeile eines Zeichens gelesen wurde, wechselt der Zustand zu „Schreiben" , wo, nachdem das Byte von „charmaps_ROM" ausgewählt wurde, dieses in

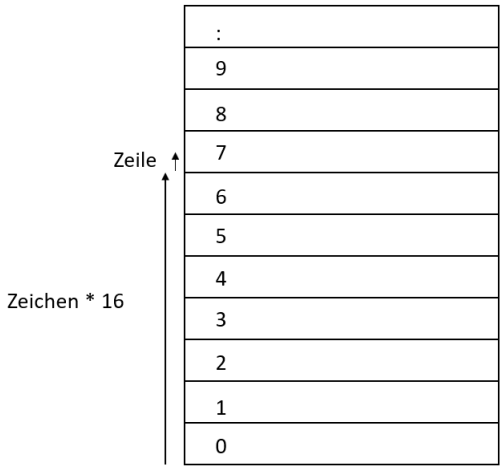

Abbildung 2: Charmap-Aufteilung und Adresse

seine einzelnen Bits zerlegt werden soll. Dort wird dann das ankommende Byte von der Stelle 7 bis zur Stelle 0 durchgegangen und jeweils eine Farbkombination in den Speicher geschrieben. Die Adresse wird dabei mit Konstanten verrechnet, die die Abstände zum Bildschirmrand festlegen, siehe 3. Die Adresse wird dabei nach folgender Gleichung berechnet:
Pixeladresse =
(Vertikaler Abstand $*$ horizontaler Maximalwert) +
(Zähler Zeilen $*$ horizontaler Maximalwert) +
Horizontaler Abstand + Bit innerhalb des Zeichens + (Zeichen $*$ 8)
Dadurch wird die richtige Speicherzelle mit den 12 Bit des Farbcodes gefüllt, vier Bit für jede Farbe. Dieser reicht daher pro Farbe von 0 bis 255. Beim Schreibvorgang wird somit nur der Bereich beschrieben in dem die Uhr auch zu sehen ist. Daher werden auch nur 16 Zeilen $*$ 8 Zeichen $*$ 8 Byte $= 1024$ Speicherzellen direkt angesprochen, wie in 3 zu sehen ist. Des weiteren wurde ein Ansatz für eine Funktion programmiert, bei der die Farbgebung kontinuirlich wechselt. Dabei ist darauf zu achten, dass immer mindestens ein gewisser Wert übergeben wird, da ansonsten der Fall auftreten kann, dass diverse Pixel nicht angezeigt werden. Diese Funktion konnte allerdings nicht mehr ausreichend getestet werden, da bei dem momentane Stand die Farbgebung nur aufgrund eines einzelnen Bits erfolgt.

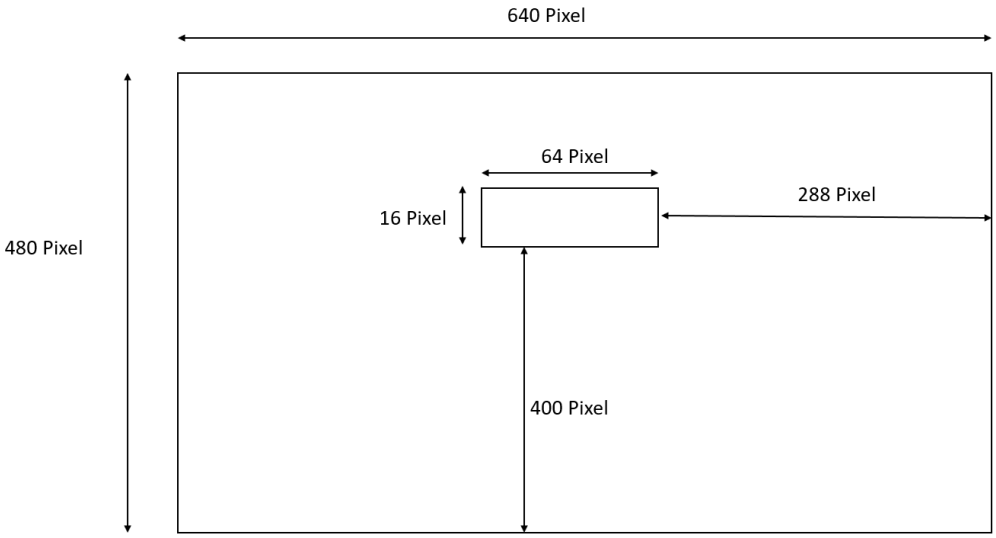

Abbildung 3: Aufteilung Bildschirm bzw. Adressraum

3.4. Probleme während der Umsetzung

Zuerst entschied ich mich für die Umsetzung mit zwei Prozessen die sich selbst synchronisieren. Nachdem allerdings keine robuste Anwendung der Implementierung möglich war, änderte ich mein Vorgehen in das einer State-Machine. Die meisten Probleme verursachte dabei die fehlerhafte Synchronierung zwischen beiden Prozessen. Nachdem das über Standardwerte, die zwar übergeben, aber anschließend herausgefiltert werden einigermaßen gelöst war, trat der Fall auf, dass die Zeichen kurz nach dem ersten Schreibvorgang bereits wieder mit „0" überschrieben wurden. Dadurch kam es auch zu keiner Ausgabe auf dem Bildschirm, was eine aufwändige Fehlersuche mittels Modelsim notwendig machte. Auch die korrekte Schreibadresse bereitete Probleme, da bereits ein verschobener Zähler dazu führte, dass das Bild wanderte bzw. sich selbst überschrieb.

3.5. Mögliche Erweiterungen

Die Anzeige könnte um eine Anzeige in Millisekunden erweitert werden. Dies ist ebenfalls in die andere Richtung möglich (Tage/Wochen/etc.). Dazu müsste allerdings die Komponente „CLOCK_MACHINE" um die richtigen Zähler erweitert werden. Ebenfalls verändert werden müsste der Prozess „Addr_finding" und zwar um zwei weitere Zustände des Zählers „Count_Char". Die Adresse würde sich dabei nicht unterscheiden, da die gleichen Zeichen aus „charmaps_ROM" verwendet werden.
Eine weitere Erweiterung wäre die Ausgabe der Uhr in einem bewegten Modus, wie

ältere Bildschirmschoner. Dazu müsste ein weitere Zähler eingefügt werden, der nach einem vollständigen Schreibvorgang weiter zählt und dabei die Adresse verändert. Dies bewirkt allerdings nur eine Verschiebung in horizontaler Richtung. Zusätzlich könnte auch noch ein zweiter Zähler eingeführt werden, der in vertikaler Richtung weiter zählt, nach dem Modus:
Vertikaler Zähler $*$ horizontaler Maximalwert = Vertikale Verschiebung
Dadurch würde dann eine diagonale Verschiebung erfolgen. Außerdem sollten noch Grenzen festgelegt werden, die verhindern, dass die Uhr nicht mehr zu sehen ist, da sie in einem Bereich angezeigt wird, der duch die Komponente „Blank_Check" nicht an den Bildschirm gesendet wird. Da die Grenzen der Uhr fest vorgegeben sind muss dazu lediglich eine Abfrage erfolgen. Bei einer sich bewegenden Uhr müssen allerdings alle nicht beschriebenen Pixel wieder auf „0" gesetzt werden, da ansonsten die Überbleibsel des vorherigen Schreibvorgangs noch zu sehen sind.
Auch eine Erweiterung auf eine andere Auflösung wäre möglich. Dabei müssten der Spericher vergrößert, die Werte für Sync und die Adressberechnung für den Speicher angepasst werden. Die Uhr würde dabei jedoch verkleinert dargestellt werden, da momentan nur ein Pixel geschrieben wird. Außerdem könnten noch andere Zeichen ausgegeben werden. Im Falle von Zeichen müsste jedoch die Adressefunktion angepasst werden um auch mit den ASCII-Werten der Buchstaben zurechtzukommen.

3.6. Zusammenfassung

Durch den späten Umschwung auf die State-Machine wurde viel Zeit in eine von Grund auf andere Art der Implementierung investiert. Diese Zeit fehlt dann später für die möglichen Erweiterungen des bestehenden und funktionsfähigen Programms. Allerdings wurden bei der Fehlersuche immer neue Kenntnisse und Durchhaltevermögen vermittelt. Außerdem bewegte sich die Verfahrensweise auf einem sehr niedrigen Niveau, wodurch die Fehlersuche entsprechend kompliziert war.

4. Speicherverwaltung

Von Michael Braun

4.1. Überblick

5. Timing und Blank_Check

Von Korbinian Federholzner

5.1. Überblick

5.2.

6. Fazit

Zusammengefasst lässt sich sagen, dass die Programmierung auf einem FPGA deutlich aufwändiger ist als auf herkömmlichen Mikrocontrollern. Außerdem ist eine komplett andere Einstellung bezüglich der Komponenten und deren Zusammenspiel notwendig. Das Abstraktionsniveau ist bei VHDL auch deutlich höher als bei Hochsprachen. Zusätzlich ist immer auch noch eine zeitliche Komponente entscheidend, da alles nur im Zyklus der Clock abläuft und dann bereits die richtigen Werte anliegen müssen, um eine korrekte Verarbeitung zu ermöglichen. Außerdem führten bereits kleiner Änderungen im Code zu einem komplett anderen Verhalten innerhalb der Simulation und der Implementierung. Auch die Fehlersuche mittels einer Simulation und ohne einen Debugger war wesentlich komplizierter als bei anderen Projekten. Allerdings wurden dabei auch wichtige Kenntnisse vermittelt und einem ein umfangreicher Überblick über die Arbeitsweise eines FPGA verschafft.

Anhang

A. Abkürzungsverzeichnis

B. Abbildungsverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

Inhalt des Datenträgers

- *Bericht*
  - *Bericht*
  - *Vortragsfolien*
  - *Stundennachweis*
- *Quellcode*
  - *SourceFiles*
  - *Extras*