



# HOCHSCHULE COBURG

Hochschule für angewandte Wissenschaften Coburg

Fakultät Elektrotechnik und Informatik

Studiengang: Informatik

Bachelorarbeit

## **Entwicklung einer hardwarebasierten Berechnung der Mandelbrotmenge auf einem FPGA**

von

Daniel Kirchner

Matrikelnummer: 02219415

Abgabe der Arbeit: 15.07.2019

Betreut durch: Prof. Oliver Engel, Hochschule Coburg

## Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde eine hardwarebasierte Visualisierung der Mandelbrotmenge auf einem FPGA realisiert.

Hierfür werden diverse mathematische und designtechnische Performanceoptimierungen vorgestellt, welche in einem parallelen FPGA-Design implementiert wurden. Das Projekt wurde für das Zybo Zynq-7000 Trainer Board entwickelt, welches über einen VGA-Output die Repräsentation des Mandelbrotfraktals in Form eines 800x600@60Hz Videosignals ausgibt.

Zur optimalen Ausnutzung der auf diesem Board gegebenen Ressourcen (DSPs, BRAM) wurde die *Vivado Design Suite* mit dem integrierten IP-Katalog verwendet.

Die entwickelte Hardware kann mit Tasern, welche über eine PMOD-Schnittstelle angeschlossen sind, gesteuert werden.

In the context of this Bachelor thesis, a hardware-based visualization of the Mandelbrot set was realized on an FPGA.

Various performance optimizations are presented, which were implemented into a parallel FPGA design. The project was developed for the Zybo Zynq-7000 Trainer Board, which outputs the representation of the fractal in form of an 800x600@60Hz video signal via VGA output.

The *Vivado Design Suite* with the integrated IP catalog was used to optimally utilize the resources given on this board (DSPs, BRAM).

The developed Hardware can be controlled through buttons, which are hooked up via a PMOD interface.

## **Inhaltsverzeichnis**

<b>Zusammenfassung</b>	<b>1</b>
<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>Abbildungsverzeichnis</b>	<b>3</b>
<b>Tabellenverzeichnis</b>	<b>4</b>
<b>Codebeispielverzeichnis</b>	<b>5</b>
<b>Abkürzungsverzeichnis</b>	<b>6</b>
<b>1 Einleitung</b>	<b>7</b>
1.1 Motivation . . . . .	7
1.2 Aufgabenstellung . . . . .	7
1.3 Mitgelieferte Skripts . . . . .	8
<b>2 Technische Grundlagen</b>	<b>9</b>
2.1 FPGAs . . . . .	9
2.2 VHDL . . . . .	10
2.3 VGA-Schnittstelle . . . . .	11
2.4 Verwendete Hardware . . . . .	12
<b>3 Theoretische Grundlagen</b>	<b>14</b>
3.1 Fraktale . . . . .	14
3.1.1 Natürlich vorkommende Fraktale . . . . .	14
3.1.2 Fraktale in der Mathematik . . . . .	15
3.2 Die Mandelbrotmenge . . . . .	17
<b>4 Umsetzung</b>	<b>20</b>
4.1 Gewähltes Zahlenformat . . . . .	20
4.2 Systemüberblick . . . . .	23
4.3 Komponentenbeschreibung . . . . .	24
4.3.1 Mandelbrot-Core . . . . .	24
4.3.2 Mandelbrot-Koordinator . . . . .	27
4.3.3 Speicher . . . . .	29

4.3.4 Lookup Tables . . . . .	30
4.3.5 VGA-Modul . . . . .	32
4.4 Peripherie und Steuerung . . . . .	33
4.5 Clock- und Resetsignal . . . . .	36
<b>5 Zusammenfassung</b>	<b>38</b>
<b>6 Ausblick</b>	<b>39</b>
<b>A Farbtabellen</b>	<b>42</b>
<b>B Ehrenwörtliche Erklärung</b>	<b>43</b>

## Abbildungsverzeichnis

1	Die Mandelbrotmenge . . . . .	8
2	Beispielhafter Aufbau einer 2-Input LUT . . . . .	9
3	Logikblock, aus [1] . . . . .	9
4	VGA-Timing/Aufbau für eine 640x480 Auflösung, aus [2] . . . . .	11
5	Zybo Zynq-7000 ARM/FPGA SoC Trainer Board, aus . . . . .	13
6	Fraktal definierter Baum, Skript: <code>/scripts/tree.py</code> , abgewandelte Version von [3]	14
7	Blumenkohl, von Rainer Renz . . . . .	15
8	200 Iterationsschritte für $c = -0.55 + 0.46i$ (rot markierter Punkt) . . . . .	17
9	Mandelbrotmenge in Graustufe . . . . .	18
10	Funktionsweise der Funktion <i>fixlen</i> . . . . .	22
11	Gesamtsystem mit 2 Kernen . . . . .	23
12	Mandelbrot-Core, schematische Darstellung . . . . .	24
13	Funktionsweise Mandelbrot-Core, Impulsdiagramm . . . . .	25
14	Mandelbrot-Koordinator, schematische Darstellung . . . . .	27
15	Mandelbrot-Koordinator, schematische Darstellung . . . . .	28
16	Block Memory Generator, IP-Bauteil . . . . .	29
17	Aufbau des VGA-Moduls . . . . .	32
18	Ausschnitt der Mandelbrotmenge mit Fadenkreuz . . . . .	33
19	Debouncer und Input-Modul, schematische Darstellung . . . . .	33
20	Darstellung einer Vergrößerung um Faktor 2 . . . . .	35
21	Prellvorgang, Signal <i>taster</i> repräsentiert den tatsächlich betätigten Taster, während <i>baustein</i> das gesendete Signal ist . . . . .	36

## Tabellenverzeichnis

1	VGA-Werte für ein 800x600@60Hz Signal . . . . .	12
2	Bereich der Mandelbrotmenge . . . . .	20
3	Farbtabelle für den Modus 2, Übergang von Rot nach Grün . . . . .	31
4	Farbtabelle für den Modus 1, Übergang von Weiß (in Menge enthalten) nach Schwarz . . . . .	42
5	Farbtabelle für den Modus 2, Übergang von Rot (in Menge enthalten) nach Grün	42

## Codebeispielverzeichnis

1	Algorithmus zur Berechnung der Iterationszahl eines $c$ . . . . .	18
2	VHDL-Funktion <i>fixlen</i> aus <a href="#"><i>/src/mandel-zybo.srcts/sources_1/imports/new/mbcore.vhd</i></a>	22

## **Abkürzungsverzeichnis**

JAX-RS Java API for RESTful Web Services

# 1 Einleitung

## 1.1 Motivation

Die stets wachsende Zahl von Komponenten, die auf einem mikroelektronischen Bauteil pro Zeiteinheit untergebracht wird, ist ein Phänomen, welches Gordon Moore schon im Jahr 1965 aufgefallen ist [4]. Die populäre, nach ihm benannte Beobachtung, dass die Anzahl der Transistoren pro integriertem Schaltkreis exponentiell mit der Zeit ansteigt, ist allgemein als das Mooresche Gesetz bekannt.

Diese Gesetzmäßigkeit machte es möglich den stets wachsenden Leistungsanforderungen an moderne Hardware gerecht zu werden, indem immer mehr (und komplexere) identische Allzweck-Prozessoren (Kerne) pro CPU verbaut wurden.

Dieses Vorgehen kann jedoch nicht unbegrenzt lange betrieben werden, da die heute verwendeten MOS-Transistoren sich rapide ihren physikalischen Grenzen annähern. Ein besserer Umgang mit dem stetig steigenden Bedarf an Rechenleistung ist die Entwicklung von spezialisierter Hardware, welche zwar nur ein kleines Aufgabenspektrum abdeckt, dies jedoch mit hoher Performanz und Energieeffizienz tut.

Ein Beispiel hierfür ist die moderne Grafikkarte (GPU), welche dem Prozessor Darstellungsberchnungen abnimmt, wodurch dieser mehr Zeit hat, andere Aufgaben zu übernehmen. Die Grafikkarte führt diese Aufgaben mit enorm hohem Durchsatz und niedrigen Berechnungszeiten durch, welche ein herkömmlicher Prozessor alleine nicht erreichen könnte.

Auch andere Hardwarekomponenten, wie die Netzwerkkarte, kryptographische Beschleuniger, oder Soundkarten sind in fast allen Computersystemen verbaut und entlasten den Hauptprozessor. Man spricht auch von heterogenen Computersystemen.

Die im Rahmen dieser Arbeit vorgestellte Hardware soll ein Beispiel für eine derartige heterogene Komponente sein. Auf einem Field Programmable Gate Array (FPGA, s. Abschnitt 2.1) soll eine performante und energieeffiziente Visualisierung der sogenannten Mandelbrotmenge realisiert werden. Diese Problemstellung ist auch durch einen ordinären Prozessor lösbar, lastet diesen jedoch enorm aus und ist somit auch nicht sehr energieeffizient.

## 1.2 Aufgabenstellung

Die Aufgabe dieses Projektes ist es, eine komplett in Hardware stattfindende Berechnung der Mandelbrotmenge durchzuführen und die Ergebnisse über eine VGA-Schnittstelle darzustellen.

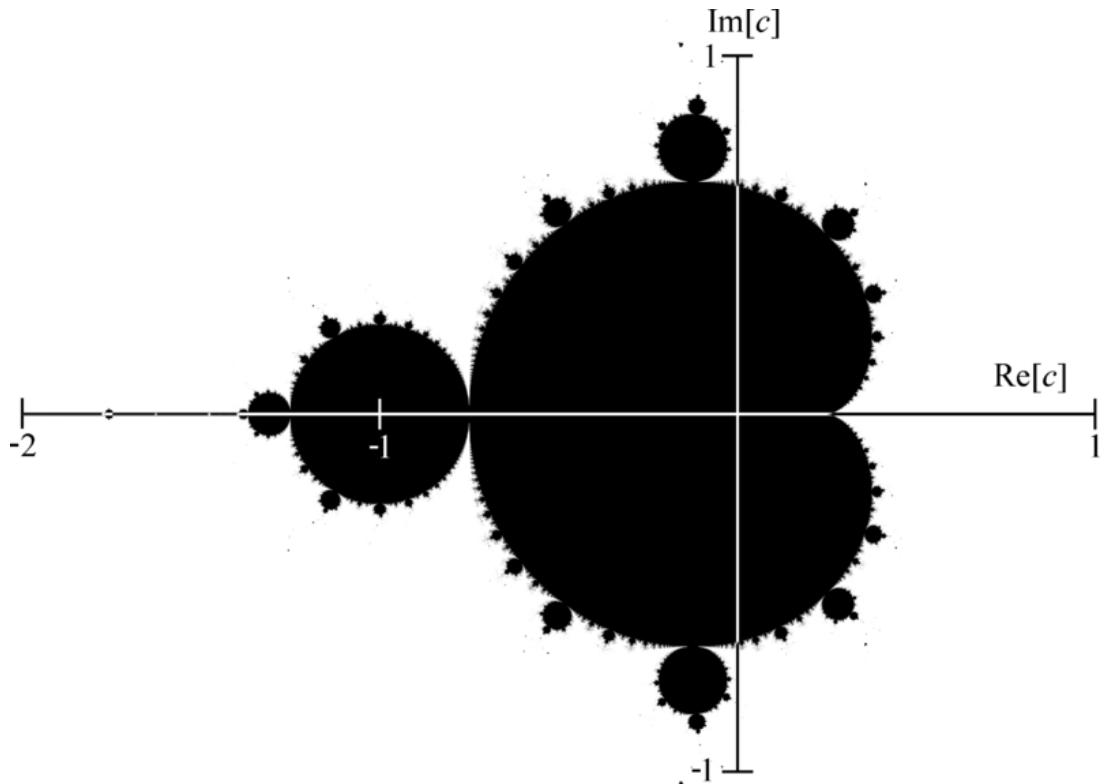


Abbildung 1: Die Mandelbrotmenge

Weiterhin soll die Hardware durch externe Peripherie konfigurierbar hinsichtlich der angestellten Berechnungen sein. So soll etwa aktuell abgebildete Bereich der Mandelbrotmenge oder auch die Farbgebung der Darstellung im laufenden Betrieb geändert werden können.

Hierfür wurde das FPGA-Trainer Board *Zybo Zynq-7000* zur Verfügung gestellt, welches in Abschnitt 2.4 genauer vorgestellt wird.

### 1.3 Mitgelieferte Skripts

Im Git-Repository *bachelorarbeit*<sup>1</sup> sind sämtliche Python-Skripts, die zur Erstellung von selbsterstellten Bildern verwendet wurden, enthalten. Des weiteren gibt die dort enhaltene Datei *README.md* Aufschluss über nützliche Skripts, die im Rahmen dieser Arbeit verwendet wurden.

Das in dieser Arbeit beschriebene System ist als VHDL-Umsetzung im Ordner */src/* zu finden, während der L<sup>A</sup>T<sub>E</sub>X-Code dieser Arbeit im Ordner */arbeit/* liegt. Wenn eine Datei aus genanntem Repository referenziert wird, wird der Pfad der Datei in **rot** dargestellt.

---

<sup>1</sup><https://github.com/DanielKirchner/bachelorarbeit>

## 2 Technische Grundlagen

Zum besseren Verständnis des Gesamtprojektes sollen in diesem Kapitel einige technische Konzepte erläutert werden.

### 2.1 FPGAs

Ein *Field Programmable Gate Array* (kurz FPGA) ist ein Schaltkreis, welcher mit Hilfe von Hardwarebeschreibungssprachen (s. Abschnitt 2.2) konfiguriert werden kann, um beliebig komplexe logische Schaltungen zu realisieren.

Das Grundelement eines solchen Bausteines bilden die sogenannten *Lookup Tables* (kurz LUTs), welche zu einem beliebigen n-bit Input ein 1-bit Output Signal produzieren. Die zugrundeliegende Logiktabelle einer LUT ist hierbei frei programmierbar.

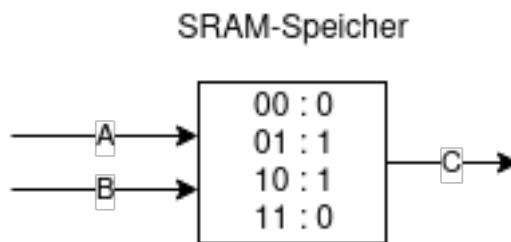


Abbildung 2: Beispielhafter Aufbau einer 2-Input LUT

Eine LUT, welche die Operation  $C = A \oplus B$  implementiert ist in Abbildung 2 zu sehen. In dieser wird in einem SRAM-Speicher für jede Inputkombination ein Outputwert hinterlegt, wodurch jede 2-Bit Funktion abgebildet werden kann. In Verbindung mit einem Flipflop<sup>1</sup> bildet eine LUT dann einen sogenannten Logikblock (s. Abbildung 3). [1]

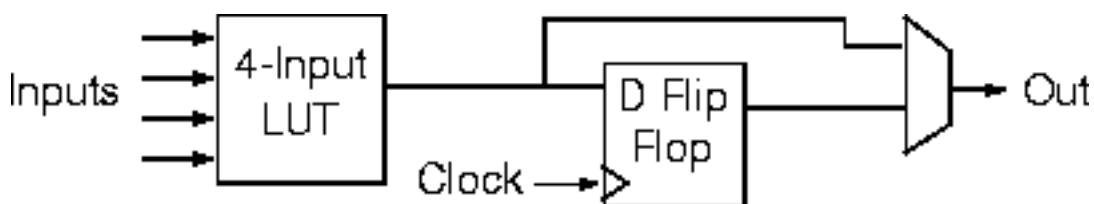


Abbildung 3: Logikblock, aus [1]

Ein FPGA verbindet nun durch ebenfalls konfigurierbare Bussysteme viele solcher Logikblöcke um komplexere Schaltkreise abzubilden.

<sup>1</sup>Ein Flipflop ist ein Speicherelement, welches einen einzigen Bit Daten halten kann.

Weiterhin verfügen FPGA-Boards oft noch über ergänzende Hardwarekomponenten, von denen die im Falle dieses Projektes vorhandenen im Folgenden gezeigt werden sollen.

**DSP** Ein *Digitaler Signalprozessor* (DSP) ist ein fest integrierter Baustein, welcher durch Multiplizierer und Akkumulatoren binäre Algorithmen beschleunigt. So übernimmt dieser etwa grundlegende mathematische Operationen, was dazu führt, dass diese flächen- und energiesparender durchgeführt werden, als bei der Verwendung von LUTs [5, S. 52]. Typische Anwendungsbereiche dieser Bausteine sind Fließkommamultiplikationen, Schnelle Fourier-Transformationen oder einfache Zähler [5, S. 14]. In diesem Projekt wurden die DSPs hauptsächlich aufgrund ihrer 25x18 Bit Multiplizierer verwendet, welche kaskadiert werden können um beliebig Breite Multiplikationen durchzuführen.

**Block RAM** FPGAs verfügen meist über *Block RAM* (BRAM), welcher zur Speicherung binärer Daten dient. Dieser Speicher ist lese-/schreibsynchrone, wodurch Inkonsistenzen beim Speicherprozess ausgeschlossen sind [6, S. 11]. Um auf diese Speicherblöcke Zugriff zu erlangen muss der von Xilinx mitgelieferte Baustein "Block RAM Generator" verwendet werden.

**IO-Komponenten** Zur Kommunikation mit der Außenwelt verfügen FPGAs über diverse Schnittstellen wie z.B. Knöpfe, Schalter, aber auch komplexere Anbindungen wie etwa VGA (s. hierzu Abschnitt 2.3) oder PMOD-Anschlüsse. Diese sind so angebunden, dass ihre Signale direkt in Logikschaltungen von LUTs integriert werden können.

## 2.2 VHDL

VHDL<sup>2</sup> ist eine Sprache zur Beschreibung und Simulation von digitalen Schaltkreisen. Spezifiziert wird diese über IEEE-Standards, der neueste zum Zeitpunkt der Abgabe dieser Arbeit ist der Standard IEEE-1076c-2007 [7].

Die Sprache verfügt über eine Standardbibliothek mit verschiedenen Datentypen, wobei im Rahmen dieser Arbeit die in Tabelle

Test	Test
------	------

---

<sup>2</sup>VHDL steht ausgeschrieben für *Very High Speed Integrated Circuit Hardware Description Language*

## 2.3 VGA-Schnittstelle

Eine *Video Graphics Array* (VGA) Schnittstelle wird durch einen Videoübertragungsstandard, welcher erstmals von IBM in ihrer *IBM Personal System/2* Modellreihe verbaut wurde, spezifiziert [8]. Das Darstellungsverfahren verwendet einen 15-poligen Anschluss, um Videosignale in variabler Auflösung und Bildwiederholungsrate zu übertragen.

Hierbei liegen die RGB-Werte eines jeden Pixels als analoge Spannungen an und werden zu bestimmten Zeitpunkten vom Bildschirm ausgelesen. Da der VGA-Standard im Jahre 1987 aufkam, war er ursprünglich noch für Kathodenstrahlröhrenbildschirme (auch Röhrenbildschirme genannt) ausgelegt. Die Elektronenstrahlen dieser Bildschirme konnten sich nicht ohne kurze Verzögerungen über die Anzeigefläche bewegen, was bedeutet, dass der VGA-Standard dies berücksichtigt und dem Bildschirm einige  $\mu s$  für größere Sprünge des Strahls einräumen muss. Die größten solchen Sprünge finden statt, wenn eine Pixelreihe übertragen wurde und der Strahl an die erste Pixelposition der nächsten Reihe bewegt werden muss oder wenn das gesamte Bild übertragen wurde und wieder zum oberen linken Pixel gesprungen werden muss. Während dieser Pausen werden keine RGB-Werte übertragen, man nennt diese zeitlich gedachten Bereiche auch "Porch"(engl. Vorbau).

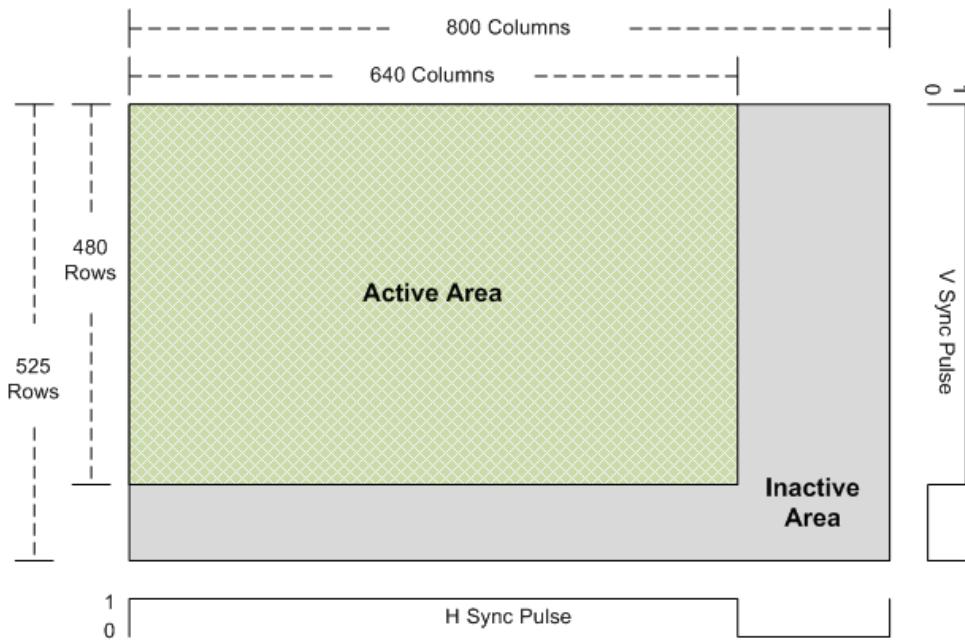


Abbildung 4: VGA-Timing/Aufbau für eine 640x480 Auflösung, aus [2]

Eine 640x480 Pixel Bild baut sich dann wie in Abbildung 4 gezeigt auf: Zuerst werden 640 Pixel RGB Daten für die erste Reihe empfangen, danach kommen 160 Pixel inaktiver Bereich ( $\Rightarrow$

Porch). Das Ende dieses Bereiches wird durch das Ansteigen des low-aktiven Signal HSYNC signalisiert. Dieser Vorgang wird nun 480 mal wiederholt, bis das gesamte Bild übertragen wurde. Draufhin wird analog zur horizontalen Synchronisation das VSYNC Signal für 45 Pixel auf auf 0 gesetzt, um den vertikalen inaktiven zu signalisieren. Die Länge des inaktiven Bereiches setzt sich aus Front-, bzw. Backporch und der Länge des Sync-Pulses zusammen. Die Bedeutungen der einzelnen Signale fügen jedoch dem nötigen Verständnis nicht mehr hinzu, weswegen die Summe dieser Bereiche als Ganzes angesehen werden kann. Wie schon erwähnt ist 640x480 jedoch nicht die einzige Auflösung, die mit einem VGA-Anschluss realisierbar ist. Andere Auflösungen (mit anderen Wiederholungsraten) müssen vom verwendeten Bildschirm unterstützt sein, und werden durch verschiedene schnelle Pixelclocks und Porches realisiert.

Die in dieser Arbeit verwendete Auflösung von 800x600 Pixeln bei 60Hz benötigt die in Tabelle 1 dargestellten Werte.

aktiver Bereich (horizontal)	Pixel	800
aktiver Bereich (vertikal)	Pixel	600
inaktiver Bereich (horizontal)	Pixel	256
inaktiver Bereich (vertikal)	Pixel	28
Pixelfrequenz	MHz	40

Tabelle 1: VGA-Werte für ein 800x600@60Hz Signal

Die in Tabelle 1 gezeigte Pixelfrequenz von 40 MHz bedeutet, dass alle 25 ns ein neuer RGB-Wert anliegen muss. Dies ist ein wichtiger Aspekt für weitere grundlegende Designentscheidungen.

## 2.4 Verwendete Hardware

Zur Umsetzung der Aufgabenstellung wird ein *Zybo Zynq-7000 ARM/FPGA SoC Trainer Board* (fortan kurz Zybo-Board genannt) verwendet (s. Abbildung 5).

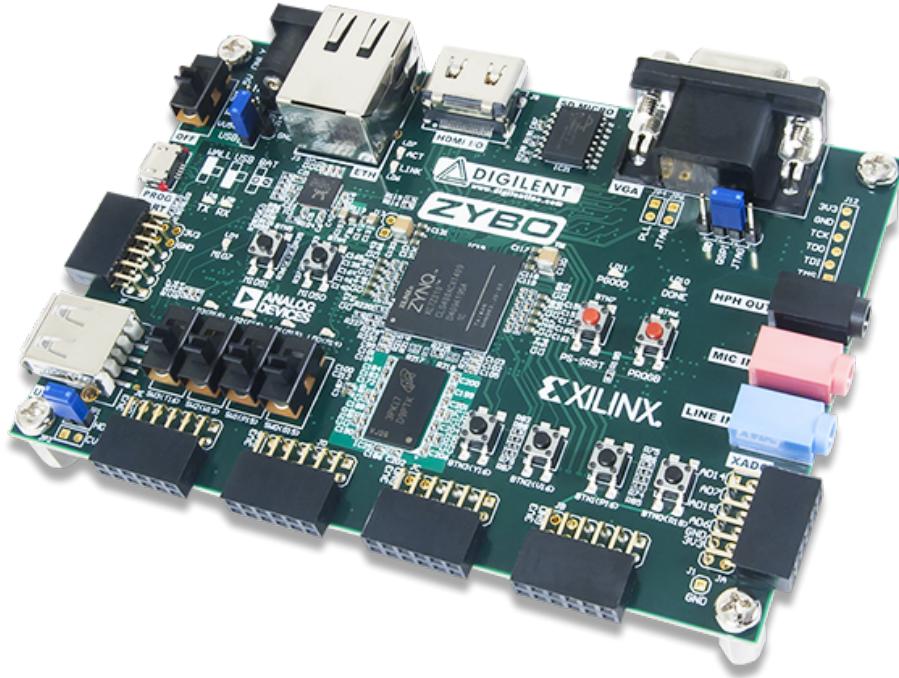


Abbildung 5: Zybo Zynq-7000 ARM/FPGA SoC Trainer Board, aus

Dieses verfügt unter anderem über folgende technischen Daten:

- Xilinx Zynq-7000 FPGA mit 28000 LUTs
- 240KB Speicher
- 80 DSPs
- ARM Cortex-A9 Zweikernprozessor
- VGA-Schnittstelle
- PMOD-Schnittstellen

Der verbaute Zweikernprozessor kommt jedoch im Rahmen dieses Projektes nicht zum Einsatz.

## 3 Theoretische Grundlagen

Die dieser Arbeit zugrundeliegenden mathematischen Grundlagen und Definitionen sollen in diesem Kapitel näher erläutert werden.

### 3.1 Fraktale

Der Begriff *Fraktal* wurde vom französisch-US-amerikanischen Mathematiker Benoît Mandelbrot geprägt und leitet sich vom lateinischen Adjektiv *fractus* ab, was „in Stücke gebrochen“ oder „irregulär“ bedeutet. Allgemein ist hiermit entweder eine natürlich vorkommende Struktur mit gewissen Eigenschaften oder eine genau definierte mathematische Menge gemeint. [9, S. 16] Für ein intuitives Verständnis des Begriffes sollen im folgenden zuerst einige natürlich vorkommende Fraktale gezeigt werden, woraufhin im nächsten Abschnitt eine formale Definition des Begriffs *Fraktal* folgen soll.

#### 3.1.1 Natürlich vorkommende Fraktale

Fraktale besitzen oft selbstähnliche Strukturen, d.h. dass sich die Gesamtstruktur eines Objektes in kleinerem Maßstab immer wieder findet. Ein Beispiel hierfür ist ein fraktal definierter Baum wie er in Abbildung 6 dargestellt ist. Der Baum wird hierbei über ein einfaches rekursives Verfahren definiert, bei dem immer wieder von jedem Teilbaum aus mit einem festen Winkel in jeweils zwei Äste abgebogen wird.

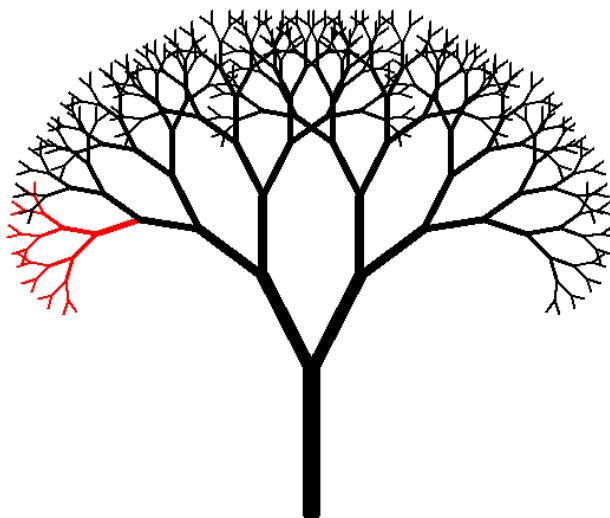


Abbildung 6: Fraktal definierter Baum, Skript: [/scripts/tree.py](#), abgewandelte Version von [3]

Der in Abbildung 6 rot markierte Teilbaum ist in seiner Struktur mit dem Gesamtbaum identisch und besitzt lediglich eine niedrigere rekursive Tiefe.

Der Blumenkohl (Abbildung 7) ist ein weiteres Beispiel für ein natürlich vorkommendes Fraktal. Auch bei diesem wiederholt sich die Gesamtstruktur in kleinerem Maßstab in den Ästen, er besitzt also ein gewisses Maß an Selbstähnlichkeit.



Abbildung 7: Blumenkohl, von Rainer Renz

Auch auf höheren Maßstäben, wie etwa bei Bergen oder ganzen Landschaften können Fraktale Strukturen immer wieder beobachtet werden. Die relativ einfachen Regeln, die diesen Fraktalen zu Grunde liegen machten sich bereits im Jahre 1980 Computergrafiker wie etwa Loren Carpenter zu Nutzen. Die relativ begrenzten Rechnerleistungen zwangen Animatoren zu diesem Zeitpunkt dazu, komplexe Landschaften Bild für Bild von Hand zu zeichnen. Durch Mandelbrot's Arbeit in fraktaler Geometrie inspiriert animierte Carpenter eine Landschaft für den Film *Star Trek II: The Wrath of Khan*, wobei er hierfür fraktale Verfahren verwendete. [10]

Die hierfür angestellten Berechnungen waren so simpel, dass pro Bildpunkt nur etwa 20 bis 40 Minuten Rechenaufwand betrieben werden mussten, was einen großen Fortschritt gegenüber der manuellen Animation darstellte. [11]

### 3.1.2 Fraktale in der Mathematik

Im Gegensatz zu den in Unterabschnitt 3.1.1 dargestellten Fraktalen, welche sich auf kleinerer Ebene nur wenige male wiederholen, sind Fraktale in der Mathematik bis zu unendlich hohem Detailgrad definiert.

Die formale Definition eines Fraktals lautet hierbei nach Mandelbrot:

*Ein Fraktal ist nach Definition eine Menge, deren Hausdorff-Besicovitch Dimension echt die topologische Dimension übersteigt.*

- Benoît Mandelbrot, aus [9, S. 27]

Die hier erwähnte Hausdorff-Besicovitch Dimension ist ein Maß, welches einem beliebigen geometrischen Raum zugeordnet werden kann, wobei die Dimension hier keine natürliche Zahl sein muss. In vereinfachter Form ermittelt sich die Hausdorff-Dimension folgendermaßen:

Man betrachte die Anzahl Kugeln (oder Kreisen)  $N$  mit Radius  $R$ , die nötig sind, um eine Punktmenge vollständig abzudecken. Geht nun  $R$  gegen 0 werden immer mehr Kugeln benötigt, um die Punktmenge abzudecken. Beobachtet wird nun in welcher Relation  $N$  zu  $R$  wächst, mit Hilfe der Formel:

$$D = - \lim_{R \rightarrow 0} \frac{\log N}{\log R}$$

wobei  $D$  die Hausdorff-Dimension ist. Betrachtet man etwa eine Linie der Länge 1, kann diese zunächst mit  $N = 1$  Kreisen des Radius  $R = 1$  abdecken. Halbiert man nun  $R$  sind doppelt so viele Kreise nötig um die Linie abzudecken. Allgemein lässt sich sagen, dass in diesem Fall  $N$  umgekehrt proportional zu  $R$  wächst. Drückt man nun  $N$  in Abhängigkeit von  $R$  aus erhält man für die Dimension einer Kurve:

$$D = - \lim_{R \rightarrow 0} \frac{\log \frac{1}{R}}{\log R} = 1$$

Analog werden etwa bei einem Rechteck  $1/R^2$  Kugeln zur Abdeckung benötigt, wenn  $R$  gegen 0 läuft. Die Dimension eines Rechtecks ist daher:

$$D = - \lim_{R \rightarrow 0} \frac{\log \frac{1}{R^2}}{\log R} = 2$$

Bei den hier gezeigten Formen ist die Hausdorff Dimension nicht höher als deren topologische Dimension<sup>1</sup> Nimmt man jedoch nun ein Fraktal, wie etwa das Sierpinski-Dreieck herbei, hat dieses oft einen gebrochenen Dimensionwert, in diesem Fall:

$$\frac{\log 3}{\log 2} \approx 1.585$$

Zusätzlich zu der bereits gezeigten Definition gilt: Jede Menge, die einen nichtganzzahligen Dimensionwert hat, ist ein Fraktal [9, S. 27].

---

<sup>1</sup>s. hierzu <https://www.math.tu-cottbus.de/~froehner/sonstiges/skripte/node9.html>

## 3.2 Die Mandelbrotmenge

Nachdem nun die formale Definition eines Fraktales bekannt ist, soll im folgenden Mandelbrotmenge erläutert werden.

Es gilt: Teil der Menge sind alle komplexen Zahlen  $c$ , für die die Iteration

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$

nicht divergiert. Im folgenden sind Iterationen für einige  $c$  gezeigt:

$$c_1 = 1; z = 2 \rightarrow 5 \rightarrow 26 \rightarrow 677$$

$$c_2 = -1; z = 0 \rightarrow -1 \rightarrow 0 \rightarrow -1$$

$$c_3 = 1 + 1i; z = 1 + 3i \rightarrow -7 + 7i \rightarrow 1 - 97i \rightarrow -9407 - 193i$$

Für  $c_1$  und  $c_3$  sieht man schnell, dass der Betrag dieser Zahlen divergiert, während es bei  $c_2$  klar ist, dass  $z$  nur zwischen 0 und -1 wechselt. Also ist  $c_2$  Teil der Menge, während es  $c_1$  und  $c_3$  nicht sind. Jedoch ist die Divergenz nicht für alle Werte so einfach festzustellen. Man betrachte folgendes  $c$ :

$$c_4 = -0.55 + 0.46i; z = 0.45 - 0.04i \rightarrow -0.34 + 0.05i \rightarrow -0.68 + 0.11i \rightarrow -0.09 + 0.29i$$

Hier ist es zunächst unklar ob eine Divergenz stattfinden wird, es müssten erst viele Iterations schritte angestellt werden, um eine Aussage treffen zu können. Nach 200 Schritten ergibt sich die in Abbildung 8 gezeigte Struktur, bei der man erkennen kann, dass  $z$  gegen einen bestimmten Punkt konvergieren scheint.

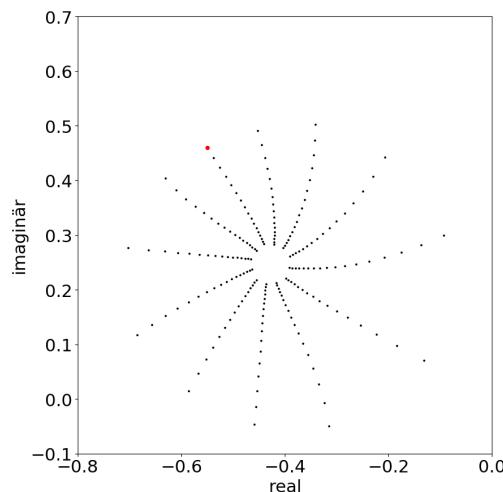


Abbildung 8: 200 Iterationsschritte für  $c = -0.55 + 0.46i$  (rot markierter Punkt)

Allgemein gilt, dass ein  $c$  nicht zur Mandelbrotmenge gehört, sobald für eine Zahl in einer Iteration gilt<sup>2</sup>:

$$|z_n| > 2$$

Zählt man nun die Iterationen, die benötigt wurden, um festzustellen, ob eine Zahl divergiert (oder eben konvergiert) und ordnet Bereichen von Iterationszahlen verschiedene Graustufen zu, erhält man ein Bild wie es in Abbildung 9 zu sehen ist.

Als Iteration wird fortan der in Zeile 4-7 dargestellte Pseudocode aus Code 1 bezeichnet.

In der Praxis ist es üblich, sich eine maximale Iterationszahl zu setzen, nach der einfach angenommen wird, dass ein gegebenes  $c$  zur Mandelbrotmenge gehört.

Eine Funktion zur Berechnung der Iterationszahl eines Punktes  $c$  der Mandelbrotmenge wird in Code 1 gezeigt.

Code 1: Algorithmus zur Berechnung der Iterationszahl eines  $c$

```

1 funktion mandelbrot(c):
2     z = 0
3     für iteration in 0 bis maxiter:
4         wenn betrag(z) > 2:
5             return iteration
6         sonst:
7             z = z*z + c
8     return iteration //maximale Iteration erreicht

```

Diese Funktion würde nun für jeden Pixel (bzw. einem dazugehörigen  $c$  Wert) aufgerufen werden, um ein Bild wie in Abbildung 9 darzustellen.

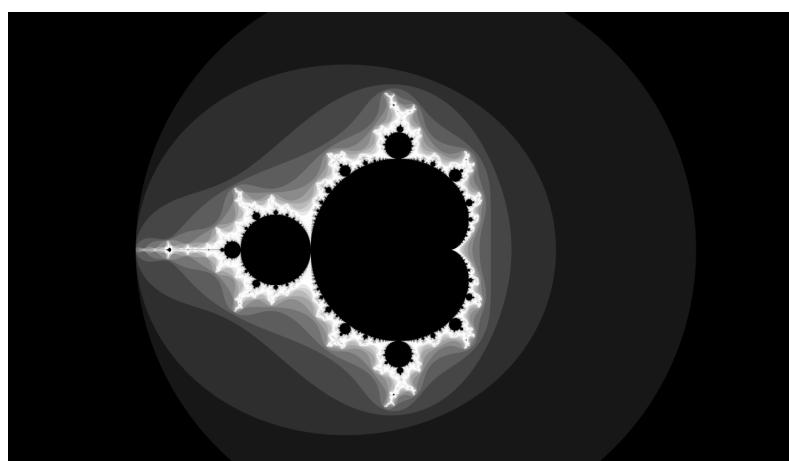


Abbildung 9: Mandelbrotmenge in Graustufe

---

<sup>2</sup>vgl. hierzu: [12]

Die eigentliche Faszination der Mandelbrotmenge liegt in ihrem Detailgrad, welcher beim Vergrößern des Randes der Menge (in Abbildung 9 weiß dargestellt), sichtbar wird. Es sind diverse (oft selbstähnliche) Strukturen zu sehen, wobei kein eindeutig visuelles Muster vorhanden zu Sein scheint. Da mit einem Vergrößern des Detailgrades präzisere Berechnungen und höhere maximale Iterationszahlen einhergehen, liegt ein enormer Rechenaufwand vor.

Einige besonders sehenswerte Teile der Mandelbrotmenge sind im dargestellt.

## 4 Umsetzung

In diesem Kapitel soll gezeigt werden, wie die in Abschnitt 1.2 formulierte Aufgabenstellung umgesetzt wurde. Hierfür wird zuerst in Abschnitt 4.1 das für die zugrundeliegenden Berechnungen verwendete Zahlenformat vorgestellt. Danach wird das in Abschnitt 4.2 gezeigte Gesamtsystem in den folgenden Abschnitten in seinen Einzelteilen erklärt.

### 4.1 Gewähltes Zahlenformat

Die Beobachtung eines komplexzahligem Punktes hinsichtlich seiner Zugehörigkeit zur Mandelbrotmenge setzt ein sehr präzises Datenformat voraus, damit sich auch bei hohen Vergrößerungsfaktoren Rechenungenauigkeiten nicht visuell bemerkbar machen.

Da die Mandelbrotmenge vollständig in dem in Tabelle 2 gezeigten Bereich liegt, muss das verwendete Datenformat nur einen kleinen Teil seiner Bits für den ganzzahligen Teil der verschiedenen Berechnungen verwenden.

Minimum Realteil	-2
Maximum Realteil	0.5
Minimum Imaginärteil	-1.25i
Maximum Imaginärteil	1.25i

Tabelle 2: Bereich der Mandelbrotmenge

Der große Teil des Bitvektors sollte also für die Darstellung der Nachkommastellen verwendet werden. Die Sprache VHDL liefert im Packet *IEEE\_numeric\_std* den Typ *signed*, welcher dazu dient vorzeichenbehaftete Ganzzahlen in einem beliebig breiten Bitvektor darzustellen.<sup>1</sup>.

Da die Rechenoperationen auf binärer Ebene für eben solche Festkommazahlen und Ganzzahlen (jeweils mit Vorzeichen) identisch sind, ist der *signed*-Typ gut zur Verwendung geeignet. Intern rechnet das FPGA also mit großen Ganzzahlen, welche jedoch eigentlich Festkommazahlen repräsentieren.

Hierbei gilt es, die Breite der *signed* Zahlen so festzulegen, dass zum einen eine maximale Präzision erreicht wird und zum Anderen die mit ihnen durchgeföhrten Multiplikationen von den eingebauten DSPs durchgeführt werden können (s. Abschnitt 2.1).

Durch Testen ergab sich, dass Multiplikationen mit mehr als 40 Bit durch LUTs umgesetzt

---

<sup>1</sup>s. hierzu die Definition in: [13]

werden (statt durch DSPs), weswegen eben diese 40 Bit sich als ideale Breite herausstellten. Da der Betrag einer Komplexen Zahl am Anfang einer Iteration nicht größer als 2 sein kann (dies wird stets geprüft), ist das größte Multiplikationsergebnis  $2 * 2 = 4$ , weswegen  $\log_2 4 + 1 = 3$  Bits für den Vorkomma teil der Zahl benötigt werden. Das höchstwertige Bit ist für das Vorzeichen reserviert, weswegen 36 Bits für den Nachkomma teil bleiben. Hiermit ist die Genauigkeit im Dezimalsystem ungefährt 11 Stellen:

$$36 * \log_{10} 2 \approx 10.84$$

Die betragsmäßig kleinste darstellbare Zahl ist damit:

$$0000,0000\dots001_2 = 2^{-36} \approx 1.455_{10} * 10^{-11}$$

Die betragsmäßig größte darstellbare Zahl ist also:

$$0111,1111\dots111_2 = 8 - 2^{-36} \approx 7.99999999998$$

Die Zahl wird im Einerkomplement dargestellt, was bedeutet, dass bei negativen Zahlen alle Bits invertiert werden. So ist die kleinste darstellbare Zahl:

$$1000,0000\dots000_2 = -8 + 2^{-36} \approx -7.99999999998$$

Selbst bei einer 134217728-fachen Vergrößerung (27 Zooms um Faktor 2) ergeben sich mit dieser Präzision noch keine visuell wahrnehmbaren Rechenfehler.

Um noch tiefere Zooms zu realisieren müsste entweder eine variable Breite der einzelnen Datenworte realisiert werden, oder der grundlegende Algorithmus zur Berechnung der Mandelbrotmenge geändert werden. Hierfür würde sich der relativ neue Algorithmus von K.I. Martin (s. [14]) eignen, welcher größtenteils unabhängig von aktueller Zoomtiefe und ohne die Verwendung von beliebig präzisen Datentypen auskommt [14].

Da die Umsetzung dieses Algorithmus jedoch um einiges komplexer als die in dieser Arbeit verwendete Methode ist und die aktuelle Präzision zufriedenstellend ist, wird von dessen Implementierung abgesehen, jedoch ist die Bitbreite der einzelnen Datenworte in der Datei *constants.vhd* konfigurierbar.

**Bitbreite bei Multiplikationen** Da bei der Multiplikation zweier 40-Bit-Zahlen das Ergebnis 80 Bit lang ist, muss dieses wieder in ein 40-Bit Format zurückgeführt werden. Dies geschieht mittels einer dafür entwickelten VHDL-Prozedur *fixlen*, welche in der Datei */src/mandelzybo.srcs/sources\_1/imports/new/mcore.vhd* zu finden ist.

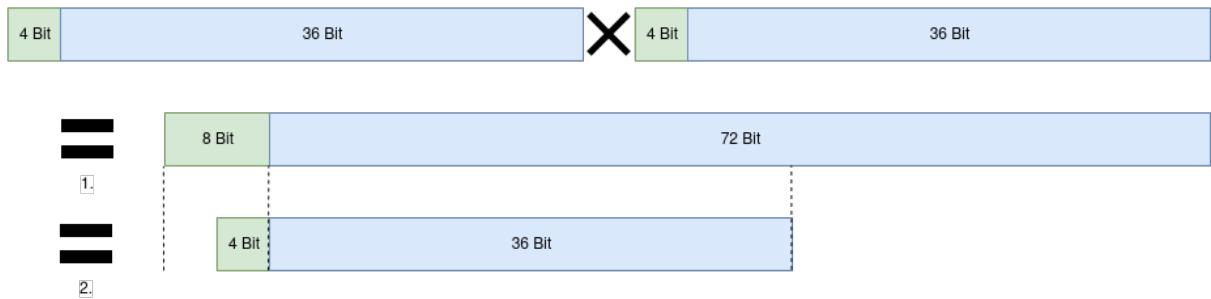


Abbildung 10: Funktionsweise der Funktion *fixlen*

Die Funktionsweise dieser Prozedur ist in Abbildung 10 dargestellt. Die aus der Multiplikation in Schritt 1. entstehende 80 Bit Zahl besteht aus 8 Bits Vorkommateil und 72 Bit Nachkommateil. Um diese Zahl nun wieder in das normale Format zurückzuführen werden die ersten 4 Bit des Vorkommateils und die letzten 36 Bits des Nachkommateils abgeschnitten (Schritt 2.). Hierbei verliert das Ergebnis der Multiplikation an Präzision, was jedoch unvermeidbar ist, da sich die Länge der Datenworte sonst bei jeder Multiplikation verdoppeln würde. Umgesetzt wird dieses Verfahren in einer VHDL-Funktion *fixlen* (Code 2), welche durch die Konstante *N\_BITS* parametrisierbar ist.

Code 2: VHDL-Funktion *fixlen* aus [/src/mandel-zybo.srcc/sources\\_1/imports/new/mbcore.vhd](#)

```

1 --N_BITS sei hier 16
2 function fixlen (input : signed(N_BITS*2-1 downto 0)) return signed is
3     variable ret : signed(N_BITS-1 downto 0);
4 begin
5     ret := input(N_BITS*2-1-4 downto N_BITS*2-4-N_BITS);
6     return ret;
7 end fixlen;
8
9 a <= "1111000011110000";
10 b <= "0000111100001111";
11
12 a <= a*b; --Fehler!
13 a <= fixlen(a*b);

```

Da das Ergebnis der 16-Bit Multiplikation in Zeile 10 32 Bit breit ist, kann dieses nicht dem 16-Bit Wert *a* zugewiesen werden. Stattdessen muss dieses Ergebnis wie in Zeile 11 gezeigt wieder auf eine Breite von 16 Bit gekürzt werden.

**Umwandlung zwischen Zahlensystemen** Funktionen zur Umwandlung zwischen dem Dezimalsystem und dem verwendeten Festkommasystem sind im Skript `/scripts/utils.py` enthalten.

## 4.2 Systemüberblick

In diesem Abschnitt soll ein Überblick über das entwickelte Gesamtsystem gegeben werden, während in den folgenden Abschnitten die einzelnen Komponenten erläutert werden.

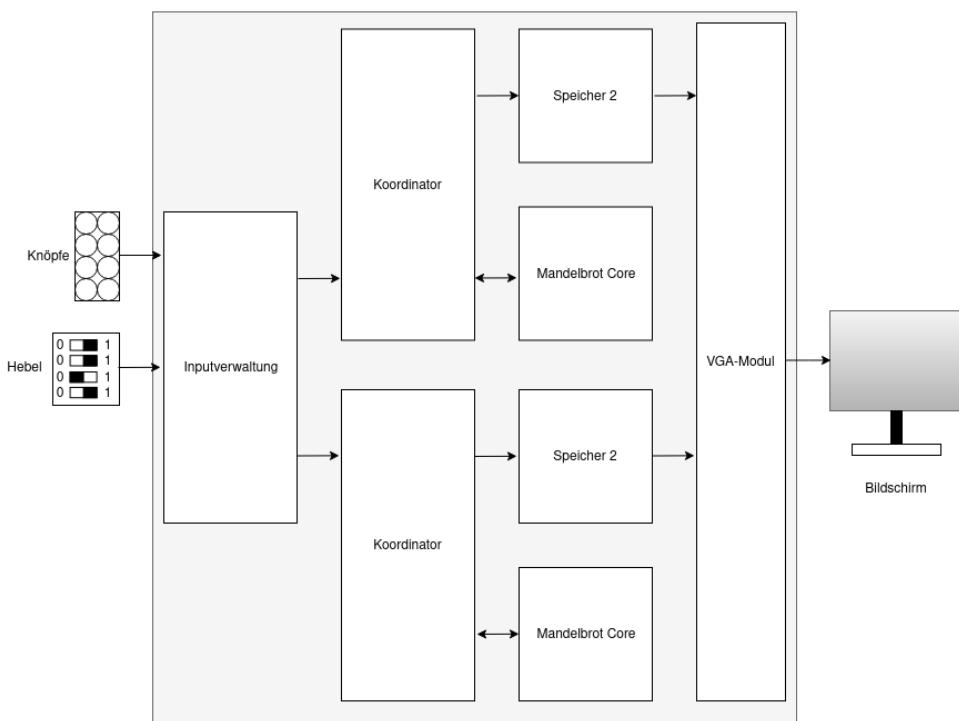


Abbildung 11: Gesamtsystem mit 2 Kernen

In Abbildung 11 ist der konzeptuelle 2-kernige Aufbau des Gesamtsystems dargestellt.

Die Darstellung erfolgt über eine VGA-Schnittstelle, welche durch das VGA-Modul bedient wird. Dieses liest stetig Pixelwerte aus den vorhandenen Speicherblöcken aus und generiert aus diesen ein gültiges VGA-Signal.

Der Speicher wird wiederum von den sogenannten *Koordinatoren* gefüllt. Jeder Koordinator verwaltet die Berechnungen eines *Mandelbrot-Cores* und schreibt dessen Rechenergebnisse in seinen eigenen Speicher.

Die Koordinatoren erhalten Informationen wie z.B. den aktuell darzustellenden Bereich von der Inputverwaltung, welche die externen Peripheriegeräte wie Hebel und Taster verwaltet.

## 4.3 Komponentenbeschreibung

Die genaue Funktion und Umsetzung aller genannten Module wird in den folgenden Abschnitten gezeigt. Alle Signale der vorgestellten Bauteile sind high-aktiv, wenn dies nicht der Fall ist, wird dies explizit deklariert. Zudem sind alle Inputs/Outputs **fettgedruckt**, um sie von Variablen in Gleichungen unterscheiden zu können.

### 4.3.1 Mandelbrot-Core

Der zentralste Baustein des Chips ist der Mandelbrot-Core (s. Abbildung 12), welcher für ein gegebenes  $c$  einen Iterationswert errechnet.

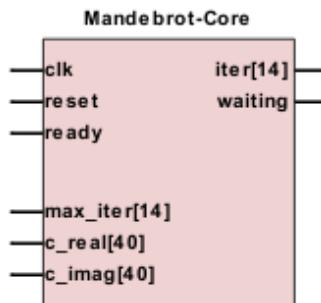


Abbildung 12: Mandelbrot-Core, schematische Darstellung

Als Input erhält ein Mandelbrot-Core zunächst die üblichen Clock- und Resetsignale (**clk,reset**). Des weiteren liegt der zu untersuchende Punkt  $c$  an den Eingängen **c\_real** (Realteil) und **c\_imag** (Imaginärteil) an. Die maximale Zahl von Iterationen, nach der angenommen wird, dass ein Punkt nicht zur Mandelbrotmenge gehört wird über das Eingangssignal **max\_iter** übermittelt. Sobald über das Signal **ready** angezeigt wird, dass **c\_real**, **c\_imag** und **max\_iter** stabil anliegen beginnt der Kern den Iterationsvorgang.

Hierbei wird jeden Zyklus von **clk** eine Iteration abgeschlossen.

Über den Ausgang **waiting** wird angezeigt, dass ein endgültiges Ergebnis am Ausgang **iter** anliegt und der Kern bereit für neue Inputs ist. Das Ergebnis kann Werte zwischen (jeweils inklusive) 0 (falls  $|c| > 2$ ) und **maxiter** annehmen.

Abbildung 13 zeigt ein Impulsdiagramm der In- und Outputs eines Mandelbrot-Cores. Zu bestimmten ist in diesem Fall die Iterationszahl für den Punkt  $-0,7487 - 0,7487i$ , wobei die maximale Iterationszahl 100 sein soll. Diese Werte werden an die Inputs **c\_real,c\_imag** und

**max\_iter** angelegt. Sobald diese stabil vorliegen, wird dies durch einen Puls des Signales **ready** signalisiert, wodurch der Core seine Berechnungen startet. Nach einigen Zyklen des Clocksignals liegt dann die Iterationszahl (in diesem Fall 3) am Ausgang **iter** an.

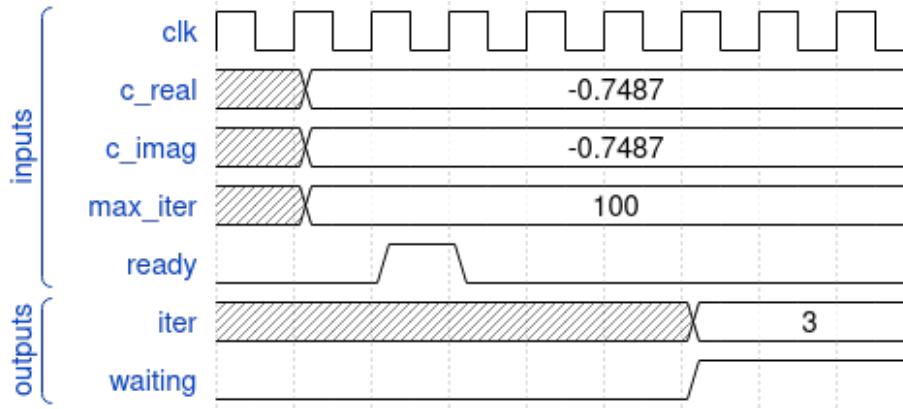


Abbildung 13: Funktionsweise Mandelbrot-Core, Impulsdiagramm

Vor diesem Zeitpunkt ist der Zustand von **iter** undefiniert, erst nach der steigenden Taktflanke von **waiting** liegt das richtige Ergebnis an. Ab diesem Zeitpunkt ist der Core auch wieder bereit, neue Werte von  $c$  zu verarbeiten.

Da der Mandelbrot-Core über einen den Eingang **max\_iter** verfügt und diesen auch bei jeder Berechnung neu ausliest, kann die maximale Iterationszahl im laufenden Betrieb jederzeit geändert werden,<sup>2</sup> was nützlich ist um bei steigender Vergrößerungsstufe den Detailgrad der Berechnung erhöhen zu können.

Die Mandelbrot-Cores werden mit einer eigenen Clock, die unabhängig von z.B. der VGA-Clock läuft, getaktet. Die maximal mögliche Taktrate ist hierbei abhängig von der Anzahl der Rechenschritte, die in jedem Iterationsschritt durchgeführt werden müssen. Eine Iteration besteht aus folgenden Bestandteilen:

- Überprüfen, ob maximale Iterationszahl bereits erreicht ist, wenn dem so ist, ist die Berechnung abgeschlossen
- Wenn nicht, berechne neuen Wert für  $z$ , falls dieser größer als 2 ist, ist die Berechnung abgeschlossen
- Zeige den aktuellen Zustand der Berechnung über die Signale **waiting** und **iter** an

Die Abfrage (A) der aktuellen Iterationszahl und der Vergleich mit der maximal zulässigen ist hierbei sehr trivial und nimmt deshalb auch keine nennenswerte Zeit in Anspruch, muss also

<sup>2</sup>Neue Werte für die Signale **c\_real**, **c\_imag** und **max\_iter** werden erst beim nächsten **ready** Puls übernommen

auch nicht weiter optimiert werden.

Der komplexzahlige Iterationsteil (B) besteht jedoch aus mehreren Multiplikationen, Additionen und Vergleichen, weswegen hier ein großer Teil der pro Iteration entstehenden Verzögerung entsteht. Es gilt, diesen Bestandteil (B) weitestgehend zu optimieren, was durch einfache algebraische Umformungen geschehen kann.

In jedem Schritt muss überprüft werden, ob  $z$  kleiner 2 ist:

$$abs(z) \leq 2$$

Der Betrag (abs) einer komplexen Zahl mit Realteil  $z_r$  und Imaginärteil  $z_i$  ist als ihr Abstand vom Ursprung definiert:

$$abs(z) = \sqrt{z_r^2 + z_i^2}$$

Also muss geprüft werden, ob gilt:

$$\sqrt{z_r^2 + z_i^2} \leq 2$$

Das Ziehen der Wurzel kann in diesem Fall durch das Quadrieren beider Seiten umgangen werden:

$$z_r^2 + z_i^2 \leq 4 \quad (4.1)$$

Falls dies wahr ist, muss der Wert von  $z$  aktualisiert werden und die Iterationszahl kann um eins erhöht werden:

$$z_{n+1} = z_n^2 + c$$

$$iter = iter + 1$$

Das Quadrat einer komplexen Zahl mit Realteil  $a$  und Imaginärteil  $b$  wird folgendermaßen berechnet:

$$(a + bi)^2 = a^2 + 2abi + b^2 * i^2 = a^2 + 2abi - b^2$$

Im Fall von  $z$  gilt also:

$$z^2 + c = (z_r + z_i i)^2 = z_r^2 - z_i^2 + 2z_r z_i i + c$$

Da die Bestandteile der komplexen Zahlen getrennt gespeichert sind, müssen diese isoliert betrachtet werden (das = Zeichen ist in diesem Kontext als Zuweisungsoperator zu verstehen):

$$z_r = z_r^2 - z_i^2 + c_r \quad (4.2)$$

$$z_i = 2z_r z_i i + c_i = z_r z_i i + z_r z_i i + c_i \quad (4.3)$$

Die Quadrate von  $z_r$  und  $z_i$  kommen sowohl in Gleichung 4.1 als auch in Gleichung 4.2 vor, müssen aber nur ein mal berechnet werden, was zwei Multiplikationen spart.

Auch das Ersetzen der Multiplikation mit zwei in Gleichung 4.3 durch eine Addition erspart eine Multiplikation. Der nicht optimierte Iterationsschritt (B) enthält eine Wurzel, sechs Multiplikationen und vier Additionen. Durch Wiederverwenden bereits berechneter Werte und algebraische Optimierung sind nun lediglich 3 Multiplikationen und fünf Additionen nötig.

Das optimierte Iterationsverfahren läuft bei einer maximalen Clockfrequenz von 50MHz stabil, was bedeutet dass eine einzelne Iteration maximal 20ns in Anspruch nimmt.

Prinzipiell können beliebig viele Mandelbrot-Cores in das Gesamtsystem integriert werden, jedoch wurde bisher nur der einkernige Betrieb realisiert.

### 4.3.2 Mandelbrot-Koordinator

Der in Abbildung 14 gezeigte Mandelbrot-Koordinator verwaltet einen Mandelbrot-Core und schreibt dessen Ergebnisse in den in Unterabschnitt 4.3.3 beschriebenen RAM.

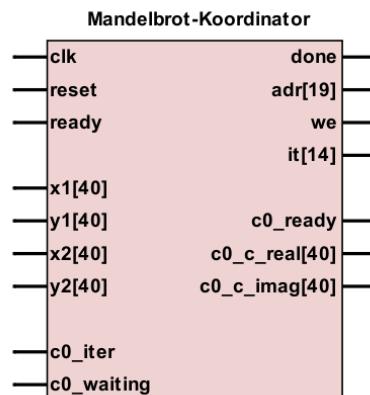


Abbildung 14: Mandelbrot-Koordinator, schematische Darstellung

Berechnet wird die Mandelbrotmenge in dem durch die Punkte ( $x_1/y_1$ ) und ( $x_2/y_2$ ) aufgespannten Rechteck (s. Abbildung 15). Liegen diese Eingangssignale stabil an, kann durch das Aktivieren des Eingangs **ready** der Rechenvorgang gestartet werden.

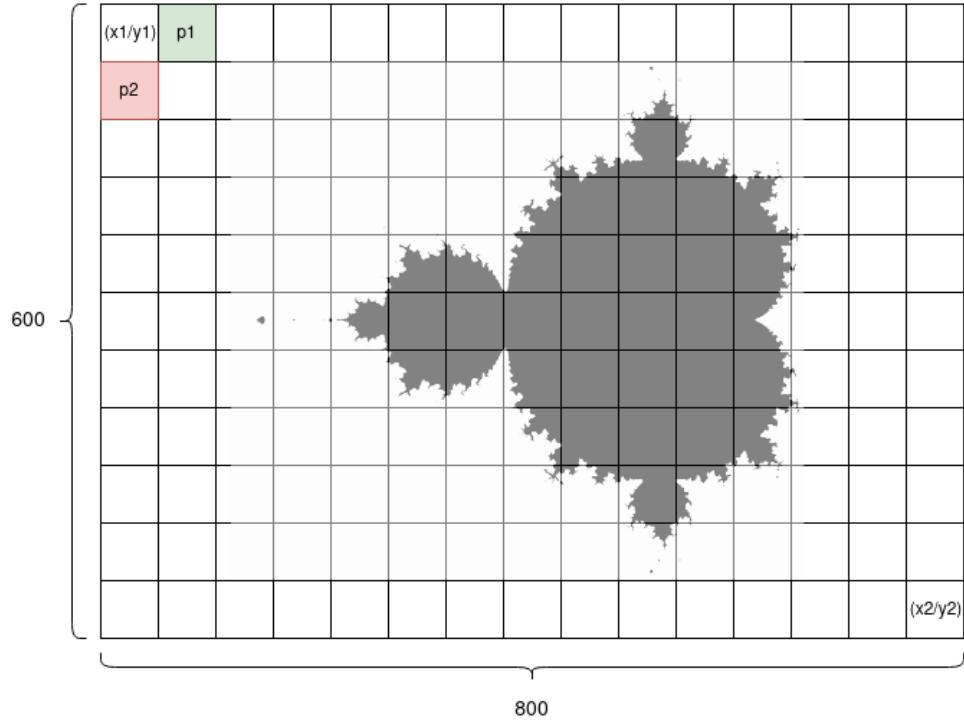


Abbildung 15: Mandelbrot-Koordinator, schematische Darstellung

Der obere linke Pixel repräsentiert den Punkt  $(x1/y1)$ . Um zu einer gegebenen Pixelkoordinate  $(p_x/p_y)$  den zugehörigen komplexzahligen Punkt  $(c_r/c_i)$  zu finden wird folgende Formel verwendet:

$$c_r = \mathbf{x1} + step_x * p_x$$

$$c_i = \mathbf{y1} + step_y * p_y$$

mit:

$$step_x = \frac{x2 - x1}{800}$$

$$step_y = \frac{y2 - y1}{600}$$

Damit das dargestellte Bild nicht gestaucht ist, sollte das Seitenverhältnis von 800x600 Pixeln eingehalten werden:

$$\frac{x2 - x1}{y2 - y1} = \frac{4}{3}$$

Wenn die Mandelbrotmenge beispielsweise für das Rechteck mit den Eckpunkten  $(-2/-2)$  und  $(2/1)$  dargestellt werden soll gilt:

$$step_x = \frac{2 - (-2)}{800} = 0,005$$

$$step_y = \frac{1 - (-2)}{600} = 0,005$$

Dem Pixel (250/100) würde folgender Punkt  $c : -0,75 - 1,5i$  zugeordnet werden:

$$c_r = -2 + 0,005 * 250 = -0,75$$

$$c_i = -2 + 0,005 * 100 = -1,5$$

Diese Werte werden an die zugehörigen Outputsignale **c0\_c\_real** und **c0\_c\_imag** angelegt und anschließen durch Aktivieren des Ausgangs **c0\_ready** bestätigt.

Im Anschluss wartet der Mandelbrot-Koordinator bis das Eingangssignal **c0\_waiting** aktiv wird und schreibt den an **c0\_iter** anliegenden Wert in den RAM.

Hierfür werden die Signale **adr,we** und **it** verwendet, wobei deren genaue Funktionsweise im Kapitel Unterabschnitt 4.3.3 näher erläutert wird. Wenn jeder Pixel abgearbeitet wurde, wird dies durch das Signal **done** signalisiert und der Koordinator ist bereit, neue Eingangssignale aufzunehmen.

### 4.3.3 Speicher

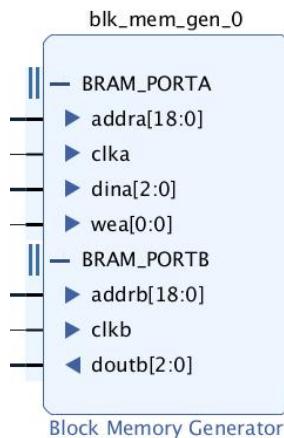


Abbildung 16: Block Memory Generator, IP-Bauteil

Der Block Memory Generator ist ein mit der Vivado Design Suite mitgeliefertes IP-Bauteil, welches ermöglicht den auf FPGA-Boards verfügbaren Block-RAM zu verwenden.

Verwendet wird der Block-RAM im Rahmen dieses Projekts, um die Iterationswerte, die jedem Pixel zugeordnet werden (s. Unterabschnitt 4.3.1) zu speichern.

Der Block-RAM des Zybo-Board setzt sich aus 60 32768-Bit RAMB18E1-Blöcken zusammen, verfügt also über 1966080 Bit (240 KB) Speicher [6, S. 14] [15]. Da in diesem Speicher ein 800x600 Pixel Bild gespeichert (und ausgelesen) werden soll, ergeben sich 4 Bit Speicher pro

Pixel:

$$\lfloor \frac{1966080 \text{ Bit}}{800 * 600 \text{ Pixel}} \rfloor = 4 \frac{\text{Bit}}{\text{Pixel}}$$

Ein Datenwort in diesem Speicher kann also 16 ( $2^4$ ) Zustände annehmen, wodurch 16 verschiedene Farbwerte gespeichert werden können.

Um alle Datenworte adressieren zu können wird ein 19-Bit Adresswort benötigt:

$$\lceil \log_2(800 * 600) \rceil = 19$$

Da die Mandelbrot-Koordinatoren einen 14-Bit Iterationswert liefern, muss dieser zur Einspeicherung auf eine Breite von 4 Bit gemappt werden. Diese Problemstellung wird in Unterabschnitt 4.3.4 behandelt.

Im Betrieb wird der Speicher permanent vom VGA-Modul ausgelesen, während die Mandelbrot-Koordinatoren neue Werte im Speicher ablegen. Da die Synchronisation dieser beiden Vorgänge enorm aufwändig wäre und die Anzahl der Lese-/Schreibzugriffe pro Sekunde eingeschränken würde, muss der Speicher im sog. *Dual-Port*-Modus betrieben werden.

Ein Speicher im Dual-Port-Modus verfügt über zwei Ports, welche völlig unabhängig voneinander agieren (**BRAM\_PORTA**, **BRAM\_PORTB**; fortan **Port A** bzw. **Port B** genannt). Jeder dieser Ports verfügt über einen eigenen Takt (**clka**, **clkb**) und einen eigenen Adressbus. **Port A** wird rein zum Schreibzugriff verwendet, während **Port B** rein zum Lesezugriff konfiguriert ist. Dies äußert sich darin, dass **Port A** einen *Write Enable*-Eingang (Schreibzugriffsaktivierung) **wea** hat, jedoch keinen Datenausgang. **Port B** verfügt wiederum über ein Adresssignal **addrb** und einen zugehörigen Datenausgang **doutb**, jedoch über keinen Write-Enable.

### 4.3.4 Lookup Tables

Lookup Tables (kurz LUTs) sind Bausteine, welche für einen Index  $i$  einen bestimmten Tabelleneintrag als Output liefern können. Die Tabelleneinträge sind hierbei vordefiniert, was einige Berechnungen, die sonst in Hardware durchgeführt werden müssten, erspart.

Im Folgenden werden die im Rahmen dieses Projekts verwendeten LUTs vorgestellt.

Da in dem in Unterabschnitt 4.3.3 beschrieben Speicher nur 4-Bit Datenworte gespeichert werden können, ist es nicht möglich, für jeden einzelnen Pixel RGB-Werte zu hinterlegen, da diese auf dem Zybo-Board 16 Bit breit sind [15]. Es muss also eine Farbtabelle existieren, welche von 4-Bit Iterations-Werten auf 16-Bit RGB-Werte übersetzt (**LUT1**).

Da der Mandelbrot-Koordinator 14-Bit Iterationswerte liefert, der Speicher jedoch nur 4-Bit Datenworte speichern kann, muss auch hier eine Übersetzung stattfinden (**LUT2**).

Für die Umsetzung von **LUT2** wird folgende Formel verwendet:

$$I_F = \lceil \frac{i}{i_{max}} * 15 \rceil$$

Hierbei sind  $i$  und  $i_{max}$  14-Bit Zahlen, wobei  $I_F$  eine 4 Bit Zahl ist, wodurch  $I_F$  in den Speicher geschrieben werden kann.

Beim Auslesen wird wiederum **LUT1** verwendet, um die per VGA dargestellten RGB-Werte zu erhalten. Die verwendeten RGB-Werte sind in einer frei konfigurierbaren Farbtabelle (Datei [/src/mandel-zybo.srcc/sources\\_1/imports/new/lut\\_colors.vhd](#)) hinterlegt. Sei die maximale Iterationszahl  $i_{max} = 100$ . Zur Ermittlung der Farbe eines Pixels muss zunächst der Farbindex  $I_F$  errechnet werden. Wenn etwa 33 Iterationen genötigt wurden sieht ergibt sich:

$$I_F = \lceil \frac{33}{100} * 15 \rceil = \lceil 4,95 \rceil = 5$$

Nun kann der zugehörige RGB-Wert aus einer Farbtabelle (z.B. Tabelle 3) entnommen und an das VGA-Modul weitergeleitet werden<sup>3</sup>:

Index $I_F$	Farbe(RGB)	Index $I_F$	Farbe(RGB)
0	#FF0000	8	#778800
1	#EE1100	9	#669900
2	#DD2200	10	#55AA00
3	#CC3300	11	#44BB00
4	#BB4400	12	#33CC00
5	#AA5500	13	#22DD00
6	#996600	14	#11EE00
7	#887700	15	#00FF00

Tabelle 3: Farbtabelle für den Modus 2, Übergang von Rot nach Grün

Zwischen den verschiedenen Farbtabellen kann im laufenden Betrieb jederzeit umgeschalten werden, hierfür erhält **LUT1** noch den Input zweier, auf dem Board integrierten, Schalter. Insgesamt sind also vier verschiedene Farbtabellen auswählbar, welche in Anhang A dargestellt sind.

<sup>3</sup>Die hier dargestellten RGB-Werte sind 24 Bit breit, da dies die übliche RGB-Notation ist. Die RGB-Werte auf dem Zybo-Board sind nur 16 Bit breit.

### 4.3.5 VGA-Modul

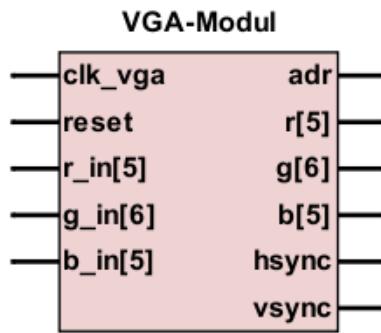


Abbildung 17: Aufbau des VGA-Moduls

Das VGA-Modul (s. Abbildung 17) setzt das in Abschnitt 2.3 beschriebene Videosignal (800x600 Pixel bei 60 Bildern pro Sekunde) um.

Hierfür wurde das VGA-Modul aus [16] an das Zybo-Board angepasst. Intern führt das VGA-Modul stets zwei Zähler  $x$  und  $y$ , welche die Position des gerade zu beschreibenden Pixels halten. Mit Hilfe der auf diesen Positionsdaten beruhenden Formel wird die aktuell auszulesende Adresse im Speicher berechnet:

$$\text{adr} = y * 800 + x$$

Sobald diese Adresse an den Output **adr** angelegt wird, liegen die korrekten RGB aus dem Speicher<sup>4</sup> an den Inputs **r\_in**, **g\_in** und **b\_in** an. Diese Signale werden dann direkt an die Ausgänge **r**, **g** und **b** weitergeleitet.

Um mit dem Bildschirmtakt synchron zu arbeiten werden die Signale **hsync** bzw. **vsync** wie in Abschnitt 2.3 beschrieben betrieben.

Da die in Abschnitt 4.4 vorgestellte Steuerung stets den Mittelpunkt des Bildschirms als Referenzpunkt verwendet, ist es nützlich diesen zu kennzeichnen. Dies geschieht durch ein Farbenkreuz, welches aus zwei roten Linien besteht (s. Abbildung 18).

<sup>4</sup>Der Speicher hält 4-Bit Iterationswerte, welche von den in Unterabschnitt 4.3.4 vorgestellten LUTs in RGB-Werte übersetzt werden.

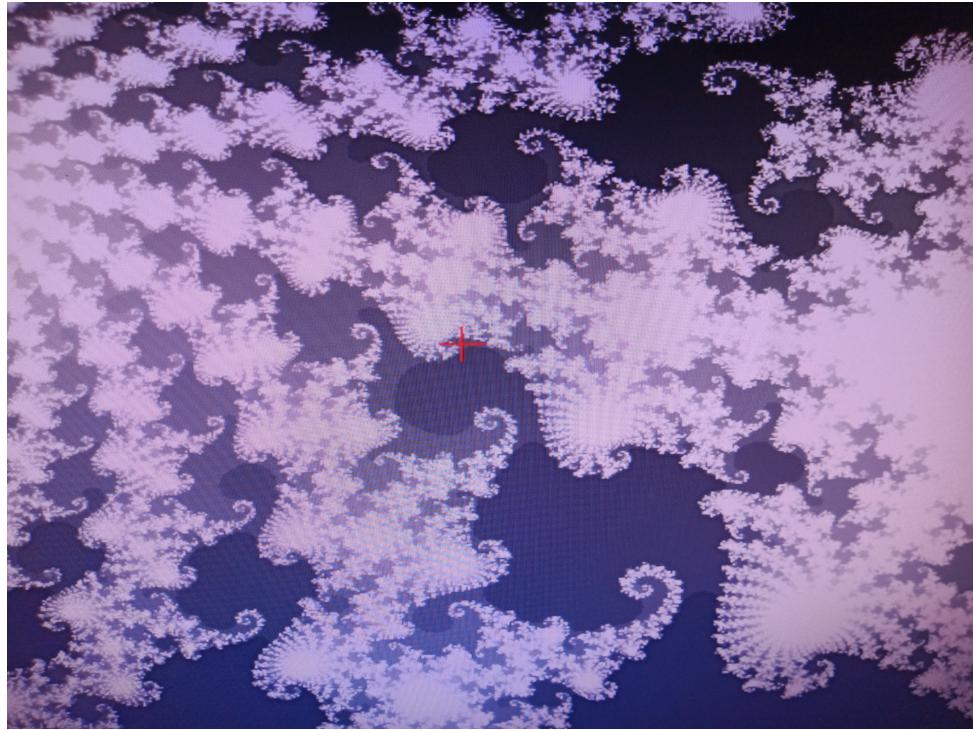


Abbildung 18: Ausschnitt der Mandelbrotmenge mit Fadenkreuz

Dieses Fadenkreuz wird vom VGA-Modul erzeugt, indem immer, wenn eine der folgenden Bedingungen erfüllt ist, ein rein roter RGB-Wert an den Bildschirm gesendet wird:

- Horizontale Position gleich 309 UND vertikale Position zwischen 289 und 309
- Vertikale Position gleich 409 UND horizontale Position zwischen 389 und 409

#### 4.4 Peripherie und Steuerung

Die Darstellung lässt sich über die an einer PMOD-Schnittstelle angeschlossenen Taster manipulieren. Alle Inputs werden vom Input-Modul (Abbildung 19) verwaltet:

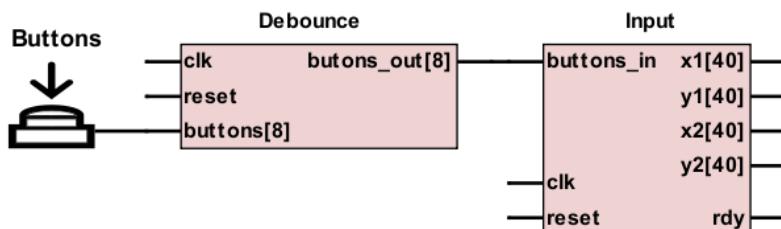


Abbildung 19: Debouncer und Input-Modul, schematische Darstellung

Das Input-Modul fragt zu jedem Takt von **clk** den Zustand der Taster vom Debouncer ab. Wenn einer (oder mehrere) Taster gedrückt sind wird eine der folgenden Aktionen ausgeführt:

- Bewegen des Bildbereichs
- Vergrößern/Verkleinern des aktuellen Ausschnitts
- Ein-/Ausschalten Hochauflösungsmodus

Für das Bewegen des Bildbereichs werden vier Taster verwendet. Ein Drücken dieser Taster bewirkt, dass der gesamte dargestellte Bereich der Mandelbrotmenge um einen bestimmten Wert verschoben wird. Dies wird dadurch erreicht, dass die dem Mandelbrot-Koordinator übergeben Koordinaten (**x1,y1,x2,y2**) um einen bestimmten Wert erhöht/verringert werden. Wird z.B. der Taster „RECHTS“ gedrückt, werden die **x1** und **x2** erhöht, was zur Folge hat, dass sich der dargestellte Bereich auf dem Bildschirm nach rechts verschiebt.

Das Vergrößern/Verkleinern des aktuellen Ausschnitts der Mandelbrotmenge geschieht stets relativ zum Mittelpunkt der dargestellten Fläche (s. Unterabschnitt 4.3.5). Da immer um Faktor 2 vergrößert werden soll, werden die Koordinaten folgendermaßen angepasst:

$$x_{1neu} = x_1 + (x_2 - x_1)/4$$

$$y_{1neu} = y_1 + (y_2 - y_1)/4$$

$$x_{2neu} = x_2 - (x_2 - x_1)/4$$

$$y_{2neu} = y_2 - (y_2 - y_1)/4$$

Zum intuitiven Verständnis ist der Vergrößerungsvorgang in Abbildung 20 (nächste Seite) dargestellt. Zum Verkleinern werden die Koordinaten analog angepasst:

$$x_{1neu} = x_1 - (x_2 - x_1)/2$$

$$y_{1neu} = y_1 - (y_2 - y_1)/2$$

$$x_{2neu} = x_2 + (x_2 - x_1)/2$$

$$y_{2neu} = y_2 + (y_2 - y_1)/2$$

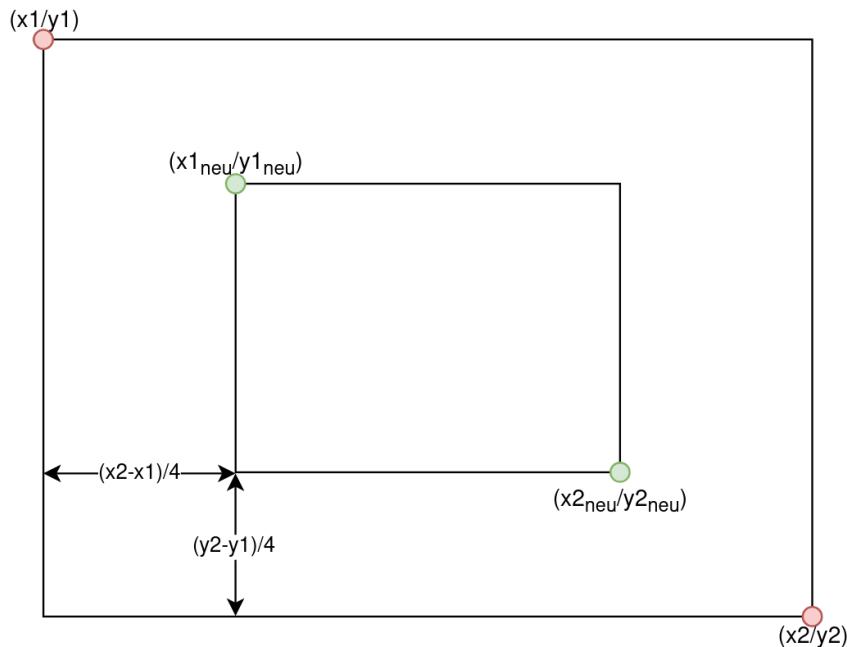


Abbildung 20: Darstellung einer Vergrößerung um Faktor 2

Der aktuelle Vergrößerungsfaktor wird intern gespeichert und wird verwendet, um zu bestimmten, wie groß die Schritte beim Bewegen des Bildbereichs sein sollen. Dies ist sinnvoll, da eine konstante Schrittgröße ab einem bestimmten Vergrößerungsfaktor keine Feinjustierung der aktuellen Position mehr zulässt.

Die Schrittgröße ist nach dem Reset auf 0.25 festgelegt und halbiert sich mit jeder Vergrößerung. Da beim Annähern an den „interessanten“ Rand der Mandelbrotmenge immer mehr Iterationen benötigt werden, um festzustellen, ob ein Punkt zur Menge gehört, muss auch die maximale Iterationszahl **max\_iter** dementsprechend erhöht werden. Diese berechnet sich abhängig von der aktuellen Vergrößerungsstufe  $z$ :

$$\text{max\_iter} \leq 90 + 25 * z$$

Wird der sog. Hochauflösungsmodus eingeschalten, wird stattdessen folgende Gleichung verwendet:

$$\text{max\_iter} \leq 90 + 100 * z$$

Dies hat zur Folge, dass das Bild sich relativ langsam aufbaut, jedoch einen sehr hohen Detailgrad besitzt, da nun potentiell fast vier mal so viele Iterationen pro Pixel durchlaufen werden müssen.

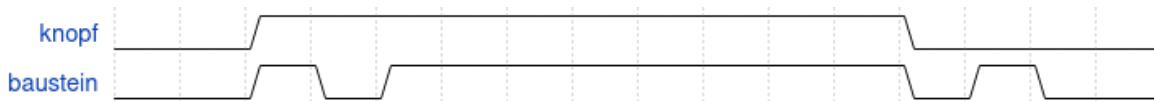


Abbildung 21: Prellvorgang, Signal *taster* repräsentiert den tatsächlich betätigten Taster, während *baustein* das gesendete Signal ist

Die Signale der Taster sind nicht durch Hardwarevorkehrungen entprellt (eng. *debounce*), was bedeutet, dass kurze Schwankungen vorliegen können, bevor das Signal beim Drücken und Loslassen der Taster stabil anliegt (s. Abbildung 21).

Um dies zu umgehen wird dem Input-Modul ein Entpreller vorgeschaltet, welcher einen Tasterdruck erst weiterleitet, wenn dieser für eine bestimmte Zeit lang stabil anliegt. Umgesetzt wird dies durch ein Schieberegister, in welches jedem Takt der aktuelle (potentiell prellende) Zustand des Tasters geschoben wird.

Liegt dann ein bestimmtes Muster an, wird der entsprechende Bit von **buttons\_out** für einen Takt von **clk** auf logisch 1 gesetzt. Das hierfür abgefragte Muster besteht aus 7 Takten logisch 1, gefolgt von 2 Takten logisch 0. Dies stellt sicher, dass pro Tasterdruck nur ein Signal geschickt wird (beim Loslassen des Tasters), was die weitere logische Verarbeitung einfacher macht.

## 4.5 Clock- und Resetsignal

Alle Bauteile (ausser die LUTs) erhalten sowohl ein Clocksignal **clk**, als auch ein Resetsignal **reset**. Das Clocksignal ist hierbei ein periodisches Signal (s. z.B. Abbildung 13), welches zur zeitlichen Synchronisation zwischen allen Bauteilen dient. Zu jeder steigenden Flanke von **clk** durchlaufen alle Module ihre Schaltlogik und speichern ihre Ergebnisse in verschiedene Formen von Speichereinheiten. Eine Ausnahme hiervon sind die LUTs, welche ihren Output direkt abhängig vom Input ändern, und somit völlig taktunabhängig arbeiten.

Das Resetsignal dient dazu, einen definierten Anfangszustand des Gesamtsystems herzustellen. So wird bei jedem Takt von **clk** abgefragt, ob das Resetsignal aktiv ist. Wenn dem so ist, werden in jedem Modul einige Aktionen ausgeführt:

1. Alle Outputs werden auf einen bestimmten Wert gesetzt (meist 0)
2. Intern verwendete Signale werden auf Anfangszustände gesetzt
3. Alle Zustandsautomaten kehren in ihren Initialzustand zurück

Auf dem Board werden zwei verschiedene Takte verwendet, welche mit Hilfe der *Clock Generator-IP*<sup>5</sup> von Xilinx generiert werden. Der erste Takt ist das vom VGA-Modul verwendete Signal **clk\_vga**, welches mit 40 MHz betrieben werden muss (vgl. hierfür Unterabschnitt 4.3.5 und Tabelle 1).

Der zweite Takt **clk** wird von allen anderen Komponenten verwendet und muss langsam genug sein, um die aufwändigen Berechnungen der Mandelbrot-Cores in nur einem Zyklus zu ermöglichen. Die hierbei empirisch ermittelte maximale Taktrate, bei der das System stabil läuft, beträgt 50 MHz. Bei höheren Taktraten können die Mandelbrot-Cores ihre Berechnungen der neuen  $z$ -Werte nicht abschließen, was zu undefinierten Zuständen der Ergebnisse führt. Dies äußert sich dann in stark verrauschten Darstellungen mit massiven Rechenfehlern.

---

<sup>5</sup>s. hierzu: [https://www.xilinx.com/products/intellectual-property/clock\\_generator.html](https://www.xilinx.com/products/intellectual-property/clock_generator.html)

## 5 Zusammenfassung

Die in Abschnitt 1.2 definierte Aufgabenstellung konnte komplett umgesetzt werden. Das entwickelte FPGA-Design kann über eine VGA-Schnittstelle beliebige Teile der Mandelbrotmenge darstellen und ist hierbei durch Hardwaretaster steuerbar. Das Design hat auf dem verwendeten Zybo 7000 Board folgende Ressourcenauslastung:

Ressource	Auslastung %
LUT	27,7
Flipflop	2,15
BRAM	98,33
DSP	22,5

Eine Erweiterung um weitere Rechenkerne wäre ressourcetechnisch also ohne Probleme möglich.

Das reine Logikdesign nimmt hierfür nur 0.258 Watt Leistung auf und ist somit enorm energieeffizient.<sup>1</sup>

Die Wahl des Zybo Boards in Verbindung mit der Vivado Design Suite erwies sich als praktikabel, jedoch verfügt die Entwicklungssoftware über einige Fehler, welche das Projekt teilweise in einen korrupten Zustand versetzten. Insgesamt stellte sich der Ansatz, alle Berechnungen in Hardware stattfinden zu lassen als herausfordernd, aber auch enorm lehrreich heraus.

---

<sup>1</sup>Wert aus dem Power Report der von der Vivado Design Suite bei jedem Projekt generiert wird.

## 6 Ausblick

In diesem Kapitel sollen einige Erweiterungen/Verbesserungen vorgestellt werden, die aufgrund des begrenzten Zeitraums der Projektbearbeitung noch nicht umgesetzt werden konnten.

**Mehrkerniger Betrieb** Prinzipiell könnten beliebig viele Mandelbrot-Cores parallel arbeiten, da diese keine gemeinsamen Ressourcen verwenden. Auch greifen die verschiedenen Mandelbrot-Koordinatoren nie auf den selben Speicherbereich zu, was Synchronisationsprobleme ausschließt. Da der mehrkernige Betrieb jedoch aufwändig zu implementieren ist, wurde er zum Zeitpunkt der Abgabe dieser Arbeit nicht umgesetzt.

**Hinzufügen neuer Farbmodi** Das Zybo Board verfügt über vier Hebel, von denen bisher zwei für die Auswahl der vier Farbmodi verwendet werden. Eine Erweiterung auf 16 auswählbare Farbmodi durch Verwendung aller vier Hebel wäre relativ einfach möglich.

**Effizienterer Mandelbrot-Algorithmus** Die Verwendung des Algorithmus aus [14] würde noch extremere Vergrößerungen der Mandelbrotmenge erlauben, ist jedoch komplexer umzusetzen. Siehe hierzu Abschnitt 4.1.

## Literaturverzeichnis

- [1] FPGA Architecture for the Challenge. [http://www.eecg.toronto.edu/~vaughn/challenge/fpga\\_arch.html](http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html). Zugriff am: 12.06.2019.
- [2] Go Board - VGA Introduction. <https://www.nandland.com/goboard/vga-introduction-test-patterns.html>. Zugriff am: 12.06.2019.
- [3] Stackoverflow Antwort: Drawing a Fractal Tree in Python, User sheldonzy. <https://stackoverflow.com/a/46754459>. Zugriff am: 12.06.2019.
- [4] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.
- [5] Xilinx, Inc. *7 Series DSP48E1 Slice*, März 2018. v1.10.
- [6] Xilinx, Inc. *7 Series FPGAs Memory Resources*, Februar 2019. v1.13.
- [7] IEEE Standard VHDL Language Reference Manual - Procedural Language Application Interface. *IEEE Std. 1076-2007*, 2007.
- [8] Chronology of IBM Personal Computers. <https://web.archive.org/web/20150221071923/http://pctimeline.info/ibmpc/ibm1987.htm>. Zugriff am: 12.06.2019.
- [9] Benoît Mandelbrot. *Die fraktale Geometrie der Natur*. Springer-Verlag, 2013.
- [10] The First Completely Computer-Generated (CGI) Cinematic Image Sequence in a Feature Film. <http://www.historyofinformation.com/detail.php?entryid=3584>. Zugriff am: 12.06.2019.
- [11] Vol Libre: The First Fractal CGI Movie. <http://www.historyofinformation.com/detail.php?entryid=3690>. Zugriff am: 12.06.2019.
- [12] user147263. Mandelbrot sets and radius of convergence. Mathematics Stack Exchange. URL:<https://math.stackexchange.com/q/890448> (version: 2014-12-19).
- [13] NUMERIC\_STD arithmetic package for synthesis. [https://www.csee.umbc.edu/portal/help/VHDL/packages/numeric\\_std.vhd](https://www.csee.umbc.edu/portal/help/VHDL/packages/numeric_std.vhd). Zugriff am: 12.06.2019.
- [14] K.I. Martin. Superfractalthing maths, 2013.
- [15] Zybo Reference Manual. <https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual>. Zugriff am: 12.06.2019.

- [16] Noureddine Ait Said. VGA Driver FPGA Altera DE1. [https://github.com/noureddine-as/VGA\\_Driver\\_FPGA\\_Altera\\_DE1](https://github.com/noureddine-as/VGA_Driver_FPGA_Altera_DE1). Zugriff am: 12.06.2019, MIT-Lizenz.

## A Farbtabellen

Index $I_F$	Farbe(RGB)	Index $I_F$	Farbe(RGB)
0	#FFFFFF	8	#777777
1	#EEEEEE	9	#666666
2	#DDDDDD	10	#555555
3	#CCCCCC	11	#444444
4	#BBBBBB	12	#333333
5	#AAAAAA	13	#222222
6	#999999	14	#111111
7	#888888	15	#000000

Tabelle 4: Farbtabelle für den Modus 1, Übergang von Weiß (in Menge enthalten) nach Schwarz

Index $I_F$	Farbe(RGB)	Index $I_F$	Farbe(RGB)
0	#FF0000	8	#778800
1	#EE1100	9	#669900
2	#DD2200	10	#55AA00
3	#CC3300	11	#44BB00
4	#BB4400	12	#33CC00
5	#AA5500	13	#22DD00
6	#996600	14	#11EE00
7	#887700	15	#00FF00

Tabelle 5: Farbtabelle für den Modus 2, Übergang von Rot (in Menge enthalten) nach Grün

Die Farbtabellen der Modi 3 und 4 sind die beiden hier gezeigten Tabellen in umgekehrter Farbreihenfolge (also z.B. rot → grün wird zu grün → rot).

## B Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie nicht an anderer Stelle als Prüfungsaarbeit vorgelegt habe.

---

Ort, Datum

---

Unterschrift