hochschule mannheim

Generierung und Implementierung von zellulären Automaten für FPGAs unter Verwendung von Location Constraints

Ngọc Minh Nguyễn

Master-Thesis

zur Erlangung des akademischen Grades Master of Science (M.Sc.)

Studiengang Informationstechnik

Fakultät für Informationstechnik

Hochschule Mannheim

30.04.2021

Betreuer

Prof. Dr. Rüdiger Willenberg

Prof. Dr.-Ing. Kurt Ackermann

Nguyễn, Ngọc Minh:
Generierung und Implementierung von zellulären Automaten für FPGAs unter Verwendung von Location Constraints / Ngọc Minh Nguyễn. –
Master-Thesis, Mannheim: Hochschule Mannheim, 2021. 20 Seiten.

Nguyễn, Ngọc Minh:
Generation and implementation of cellular automata for FPGAs using location constraints / Ngọc Minh Nguyễn. –
Master Thesis, Mannheim: University of Applied Sciences Mannheim, 2021. 20 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 30.04.2021

Ngọc Minh Nguyễn

Abstract

Generierung und Implementierung von zellulären Automaten für FPGAs unter Verwendung von Location Constraints

Hier kommt Abstract.

Generation and implementation of cellular automata for FPGAs using location constraints

The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators. To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is more simple and regular than that of the individual languages. The new common language will be more simple and regular than the existing European languages. It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a skeptical Cambridge friend of mine told me what Occidental is.

Danksagung

Zunächst möchte ich mich bei meinem Betreuer Herrn Prof. Dr.-Ing. Rüdiger Willenberg und Herrn Prof. Dr.-Ing. Kurt Ackermann, die mir stets mit wertvollen Tipps und Ratschlägen zur Seite standen und mir diese herausfordernde Thesis zugetraut haben.
Ein großer Dank geht ebenfalls an meine Freundin Sarah Nordt, die mir immer mit Rat und Tat zur Seite stand, wenn ich wieder einmal in den Untiefen der deutschen Sprache verloren war.
Und zu guter Letzt möchte ich meinem Bruder Minh Khuê Nguyễn und meiner Mutter Thị Nguyệt Bùi danken, die mir das Studium ermöglichen, mich immer wieder unterstützen und motivieren, sowohl finanziell als auch moralisch.

Inhaltsverzeichnis

Danksagung iii

1. Einleitung 1

2. Einführung und Grundbegriffe 3
2.1. Zelluläre Automaten . . . 3
2.1.1. Gitterstruktur . . . 3
2.1.2. Zustände . . . 4
2.1.3. Nachbarschaft . . . 5
2.1.4. Übergangsregel . . . 5
2.1.5. Randbedingungen . . . 6
2.1.6. Startkonfiguration . . . 7
2.2. Game of Life . . . 7

3. FPGA 9
3.1. Xilinx FPGA Architektur . . . 9
3.1.1. Logikblöcke . . . 11
3.2. FPGA Entwurfsablauf . . . 12
3.2.1. Relatively Place Macros . . . 13
3.3. Xilinx Primitives . . . 13
3.3.1. FDRE . . . 14
3.3.2. LUT6_5 . . . 14
3.3.3. CARRY4 . . . 14

4. Verwandte Arbeiten 15

5. Zelluläre Automaten in FPGA 16
5.1. Hardwarespezifikation . . . 16
5.2. Implementation einer GoL-Zelle . . . 16
5.2.1. Erstes Design . . . 16
5.2.2. Logik Minimierung . . . 16
5.2.3. Ergebnis . . . 16
5.3. Spielfeld Implementation . . . 17
5.3.1. Festlegung benachbarten Zellen . . . 17

5.3.2. Ergebnis 17

6. Prozessorsystem zur Datentransfer 18
6.1. Das CA Core Modul 18
6.2. GoL IP-Core 18
6.3. Blockdiagramm 18
6.4. Tcl-Skript zur Platzierung von GoL-Zellen 18
6.5. GoL-Modul Treiber Implementation 18
6.6. Ergebnis 18

7. Auswertung und Ergebnisse 19
7.1. Zedboard Auslastung 19
7.1.1. Timing 19
7.1.2. Resourcen 19
7.2. Microblaze vs Zynq-7000 19

8. Fazit 20

Abkürzungsverzeichnis vi

Tabellenverzeichnis vii

Abbildungsverzeichnis viii

Quellcodeverzeichnis ix

Index xi

A. Erster Anhang xi

B. Zweiter Anhang xii

Kapitel 1

Einleitung

Das Konzept zelluläre Automaten (ZA) wurde von John von Neumann in den späten vierziger Jahren eingeführt und dienen zur Modellierung und Simulation von dynamischen und komplexer Systeme. Ein zellulärer Automat besteht aus gleichmäßiger Anordnung identischer Zellen, wobei jede Zelle kann durch Wechselwirkung mit seinen Nachbarzellen von einem Zustand zum nächsten Zustand annehmen. Durch das Zusammenwirken mehrere Zellen können somit komplexe Systeme geschaffen werden.
Mit ZA lassen sich unzähligen Anwendungsgebiete simulieren unter anderem die Entstehung des Leben, Modelle im Verkehrswesen oder die Ausbreitung von Epidemien.
In der heutigen Zeit ist der Bedarf an Simulationsumgebungen mit hoher Komplexität gestiegen. Algorithmen die zur Verarbeitung von Daten und die Durchführung von tausenden von Berechnungen pro Sekunde, benötigen viel Rechenleistung, die typischerweise zu hohen Kosten und lange Simulationen verursachen.
Durch die Verwendung von Mehrkern-Prozessoren mit nebenläufiger Programmierung, um Aufgaben auf verschiedene Threads zu verteilen, ermöglicht eine beschleunigte Ausführung. Die Schwierigkeit in der nebenläufige Programmierung besteht darin, die Verteilung der Aufgaben an die einzelnen Threads. Hinzukommt, durch den Austausch von Daten zwischen mehreren Threads, kann die Performance darunter leiden.
Eine mögliche Hardwarelösung für die algorithmische Verarbeitung bietet Field Programmable Gate Arrays (FPGA). Ein FPGA besteht aus programmierbare Logikblöcke, die in einer Gitterstruktur angeordnet sind. Man programmiert sie mithilfe einer speziellen Hardwarebeschreibungsprache, beispielsweise VHSIC Hardware

Description Language (VHDL), logische Schaltungen, die anschließend in die Logikblöcke implementiert werden. Logische Schaltungen arbeiten voll parallel und verfügen über begrenzte Ressourcen verglichen mit einem gängigen Computer.
Durch ihre massiv parallelen Arbeitsweise und deren innere Architektur ähneln FPGAs einem ZA. Mit den vorhandenen Gitterstruktur lässt sich flexible logische Schaltungen realisieren, welche über Software konfigurieren können.
In der folgende Thesis wird ein FPGA der Firma Xilinx als Rechenbeschleuniger verwendet, um ein klassisches ZA Systems, das Game of Life (GOL), zu berechnen. Für das konfigurieren der Rechenbeschleuniger kommt ein Mikrocontroller zum Einsatz.

Kapitel 2

Einführung und Grundbegriffe

In diesem Kapitel wird zunächst eine allgemeine Beschreibung von zelluläre Automaten erläutert. Es werden der Grundaufbau und Ablauf angesprochen, um ein grundlegendes Verständnis über solche Systeme zu schaffen.

2.1. Zelluläre Automaten

Wie in der Einleitung beschrieben bestehen zelluläre Automaten aus gleichmäßiger Anordnung identischer Zellen. Zellen lassen sich als Zustände formulieren und die Zustandsänderung jeder Zelle wird in diskrete zeitliche Schritte betrachtet.

$$t_0, t_1, t_2, ..., t_k$$

wobei t_0 den Startzeitpunkt festlegt. Die Entwicklung einer Zelle hängt von ihrem aktuellen Zustand und den Zustand ihrer benachbarten Zellen ab. Grundsätzlich wird die zeitliche Entwicklung des ZA durch die Anfangskonfiguration der Zellen festgelegt.

2.1.1. Gitterstruktur

Die einzelnen Zellen sind in ein diskreter Raum, in dem die Entwicklung stattfindet, miteinander verbunden. Unterschiede gibt es in der Dimension (ein, zwei oder dreidimensional) und der Geometrie (reckteckig, hexagonal).
Die lineare Anordnung ist der einfachste Fall, hier sind die Zellen in einer eindi-

mensionaler Raum verbunden.

x_0	x_1	x_2	x_3	x_4	x_5	...	x_n

x_i steht für Zelle an der Position i. Wobei $i = 0, 1, ..., n$.
Ein Anwendungsbeispiel für ein eindimensionaler zellulärer Automat ist das Nagel-Schreckenberg-Modell. Hierbei handelt es sich um die Simulation einspurige Autobahn.

Bei zweidimensionaler Anordnung besteht der Raum aus einem quadratischen Gitter mit m x n Zellen.

Abbildung 2.1.: Beispiel eines zweidimensionalen Gitter: a) rechteckig und b) hexagonales

xij für $i = 0, 1, ..., n$ und $j = 0, 1, ..., m$.

In dieser Thesis wird auf zweidimensionaler rechteckiger Gitterstruktur beschränkt, da sie sich einfacher in die FPGA matrixförmiger Anordnung implementieren lässt.

2.1.2. Zustände

Jede Zelle wird durch ihr Zustand zu jedem Zeitpunkt definiert. Ihre Zustandsmenge Q ist abzählbar und können sowohl symbolische Bedeutung als auch Zahlen besitzen. Wichtig hierbei ist, dass die Zustände unterscheidbar sein und sich aufzählen lassen müssen.

$$Q = \{q_1, q_2, ..., q_s\}$$

Im einfachsten Fall sind es binäre ZA beispielsweise Ja/Nein oder Tot/Leben, s.Abb.2.3, die eine erstaunliche Komplexität aufweisen.

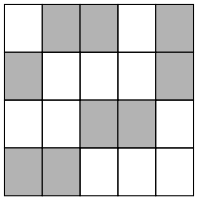


Abbildung 2.2.: Beispiel eines zweidimensionalen 4x5 Gitter mit zwei Zuständen. Wobei Q aus $s = 2$ Zuständen besteht. Weiß q_1 für “tot” und Grau q_2 für "lebendig".

2.1.3. Nachbarschaft

Als Nachbarschaft ist der Interaktionsradius einer Zelle definiert. Die benachbarten Zellen in dieser Umgebung stehen in Wechselwirkung mit der Referenzzelle, wobei jede Zelle die Nachbarschaft nach den gleichen Regeln bestimmt wird.
Es gibt für das zweidimensionale Gitter sehr viele Möglichkeiten zur Definition von Nachbarschaften, jedoch sind die **Von-Neumann-Nachbarschaft** und die **Moore-Nachbarschaft** von praktischer Bedeutung.

Abbildung 2.3.: Nachbarschaften für zweidimensionales Gitter: a) Von-Neumann-Nachbarschaft und b) Moore-Nachbarschaft.

In Abb. 2.3 ist die Von-Neumann-Nachbarschaft zusehen, diese berücksichtigt vier direkte angrenzenden Nachbarzellen, mit denen sie eine gemeinsame Kante hat. Die Moore-Nachbarschaft hingegen berücksichtigt zusätzlich die vier Eckzellen, sodass sie insgesamt acht Nachbarzellen besitzt.

2.1.4. Übergangsregel

Die wichtigste Eigenschaft eines ZA ist die Übergangsregel. Sie definiert den Zustand einer Zelle zum Zeitpunkt t_{k+1} in Abhängigkeit von den Zuständen der Zellen

in seiner Nachbarschaft zum Zeitpunkt t_k. Dabei bestimmen Regeln den Zustand einer Zelle im jeweils folgenden Zeitschritt.
Es werden grundsätzlich zwei Arten von Regeln unterschieden.

- Die deterministische Regeln bestimmt die Entwicklung eines ZA. Bei Wiederholung eines Automaten mit dem gleichen Startkonfiguration wird dieser die selben Ergebnisse liefern.
- Die stochastische Regeln geben eine Wahrscheinlichkeit für eine Zustandswechsel für die Referenzzelle an unter den jeweiligen Gegebenheiten an.

2.1.5. Randbedingungen

Ein diskreter Raum muss vorhanden sein um ein ZA simulieren zu können. Wie in Abschnitt 2.1.2 erwähnt ergibt sich der neue Zustand einer Zelle aus seiner Nachbarschaft. Liegt eine Zelle jedoch am Rand des Gitters müssen klare Regeln definiert werden, da die Nachbarzellen nicht im Gitter enthalten sind. Für diese Sonderfällen müssen daher besondere Regeln festgelegt werden und dies wäre die Randbedingungen.
Üblicherweise werden **reflektierende** und **periodische** Randbedingungen verwendet.
Bei reflektierende Randbedingungen ändert sich der Zustand der Referrenzzelle nicht, wenn ihre Nachbarschaft nicht komplett im Gitter vorhanden sind.

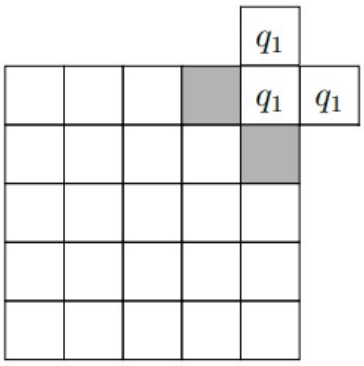


Abbildung 2.4.: Reflektierende Randbedingung am Beispiel Neumann-Nachbarschaft

Bei periodische Randbedingung werden die fehlenden Zellen durch Zellen am anderen Rand des Gitters ergänzt s. Abb. 2.5.

(a) (b)

Abbildung 2.5.: Periodische Randbedingungen am Beispiel a) Neumann- und b) Moore-Nachbarschaft.

2.1.6. Startkonfiguration

Der Grundaufbau und Ablauf eines ZA ist somit beschrieben. Es fehlt lediglich die Startkonfiguration oder auch die initiale Zuweisung eines Zustand für alle Zellen zum Zeitpunkt t_0, bevor die Simulation gestartet werden kann.
Es kann künstlich oder auch zufällig generiert werden. Die Bedeutung der Startkonfiguration wird im folgenden Abschnitt am Beispiel vom Game of Life veranschaulicht.

2.2. Game of Life

Das Game of Life oder Spiel des Lebens wurde von John Horton Conway entwickelt um eine Simulation zur Entwicklung des Lebens[Zitat]. Es existiert unzähligen Variationen und Verallgemeinerungen zum GOL. In dieser Thesis wird auf das einfachste Grundmodell beschränkt.
Bei GOL besteht die Zustandsmenge Q aus zwei Zuständen.

$$Q = \{q_1, q_2\}$$

q_1 für tot und q_2 für lebendig.
Die Moore-Nachbarschaft wird hier gewählt aus Abb.2.3 und die Entwicklung einer Zelle wird von definierten Regeln determiniert. Als Randbedingung erfolgt die periodische Randbedingung.

Die Regeln zur Definition eines Zeitschrittes werden wie Folgt beschrieben:

1. Lebt die Referenzzelle (x_{ij}) und es leben weniger als drei der neun möglichen Zellen, dann stirbt x_{ij} in der folgenden Generation.

2. Lebt x_{ij} und es leben in der Nachbarschaft drei oder vier der neun möglichen Zellen, dann überlebt x_{ij} in der folgenden Generation.

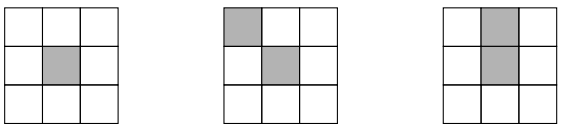
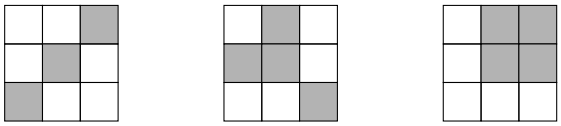
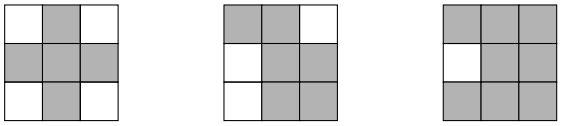
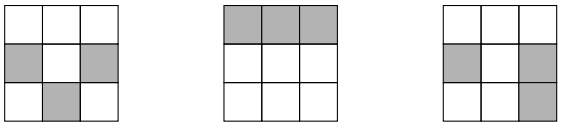
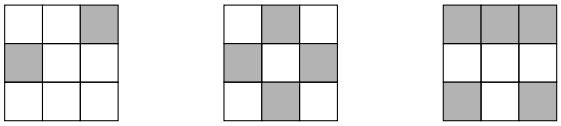
3. Lebt x_{ij} und es leben in der Nachbarschaft mehr als vier der neun möglichen Zellen, dann stirbt x_{ij} an Überbevölkerung in der folgenden Generation.

4. Wenn in der Nachbarschaft von x_{ij} genau drei von neun möglichen Zellen leben, x_{ij} jedoch nicht lebt, dann wird x_{ij} neu geboren.

5. Wenn in der Nachbarschaft mehr oder weniger als drei der neun möglichen Zellen leben, dann behält x_{ij} seinen aktuellen Zustand in der folgenden Generation.

Für alle Zellen treten genau einer der Fünf oben genannten Regeln auf und somit sind sie deterministisch. Die gewählten Startkonfiguration entscheidet über die Entwicklung der Zellen und sie können über viele Generationen hinweg zu komplexen Strukturen entwickeln.

Tabelle 2.1.: Skizze zur Visualisierung der Regeln im GOL. Es sind jeweils drei mögliche Nachbarschaften von x_{ij} zum Zeitpunkt t_k, welche im folgenden Zeitschritt t_{k+1} zu einem Zustand führen. Weiß steht für tot und grau für lebendig. Dabei ist die Referrenzelle x_{ij} immer das mittlere Feld.

x_{ij} und die Nachbarschaft zum Zeitpunkt t_k	Zustand x_{ij} zum Zeitpunkt t_{k+1}
	
	
	
	
	

Kapitel 3

FPGA

Dieses Kapitel behandelt die Grundelementen zu FPGA. Es werden die Architektur und den Entwurfsablauf beschrieben, wobei der Schwerpunkt auf Xilinx FPGA Zynq-7000 AP SoC XC7Z020-CLG484 (ZedBoard) gelegt wird.

3.1. Xilinx FPGA Architektur

FPGA besteht aus kleinen Hardwareelementen, die konfiguriert werden können, dass sie eine logische Funktion ausführen. Ein großer Vorteil von FPGAs gegenüber Prozessorsystemen ist, verschiedene Systemkomponenten auf einem Chip zu integrieren. Dies bedeutet nicht nur einen geringeren Platzbedarf, sondern auch eine erhöhte Performance.
In dieser Arbeit wird ein ZedBoard der Firma Xilinx eingesetzt s. Abb. 3.1, jedoch gilt die Architektur, die in den folgenden Abschnitten beschrieben für die meisten modernen FPGA-Chips von Xilinx.

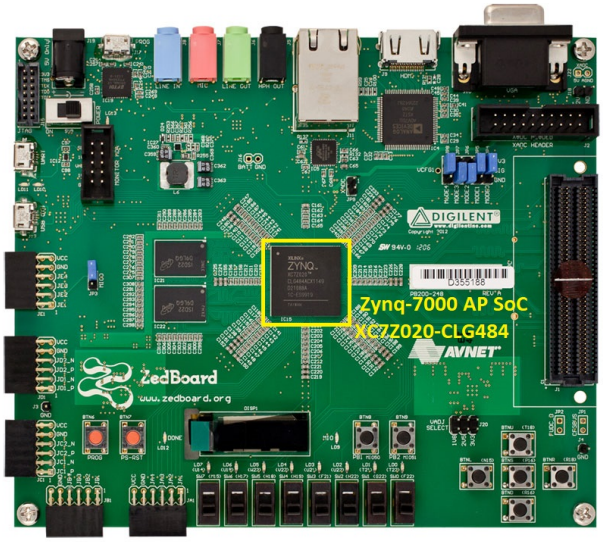


Abbildung 3.1.: Xilinx ZedBoard

Generell besteht das ZedBoard aus zwei Hauptteilen, einem Verarbeitungssystem engl. Processing System (PS), das aus einem Dual-Core ARM Cortex-A9 Prozessor besteht und einem programmierbaren Logik (PL), welche das eigentliche FPGA entspricht. Die Verbindung zwischen PL und PS werden über das Advanced eXtensible Interface (AXI) verbunden s Abb. 3.2.

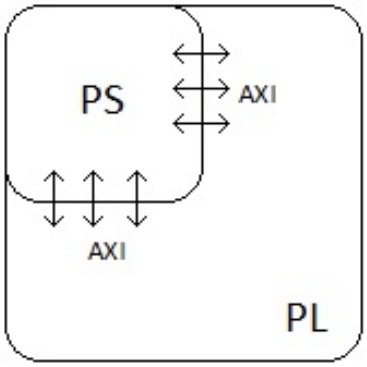


Abbildung 3.2.: Vereinfaches Modell der Zynq Architektur

Der Hauptbaustein eines PL ist ein sogenannter konfigurierbare Logikblock engl. configurable logic block (CLB) und neben jedem CLB befinden sich eine Schaltmatrix, die eine flexible Routing Möglichkeiten bietet, um Verbindungen von einem CLB zum anderen herzustellen s Abb. 3.3.

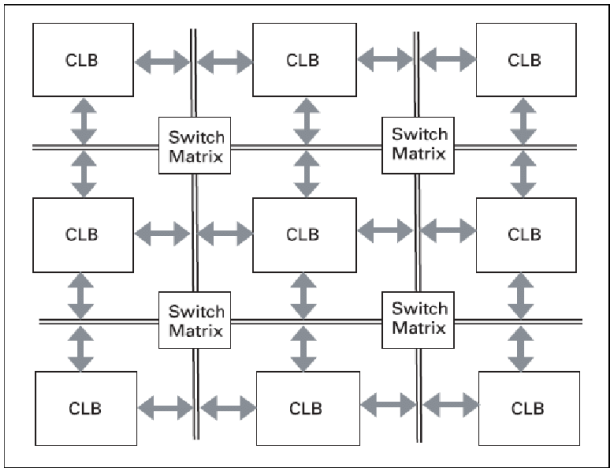


Abbildung 3.3.: Grundlegende Architektur eines PL.

3.1.1. Logikblöcke

Logikblöcke oder auch CLBs sind kleine regelmäßige Gruppierungen von Logikelementen, die in einem zweidimensionales Gitter auf dem PL angeordnet sind. Ein CLB kann zwei SLICEL oder einen SLICEL und einem SLICEM enthalten. Der Unterschied zwischen diesen beiden Slices -Typen sind, dass SLICEM mehr Konfigurationsmöglichkeiten bei der Look Up Table (LUT)s besitzt.
Jeder Slice verwendet die folgenden Grundelementen engl. Basic Element of Logic (BEL) um Logik-, Arithmetik und ROM-Funktionen bereitzustellen:

- Vier LUTs mit jeweils sechs unabhängige Eingängen (A1 bis A6) und zwei unabhängigen Ausgängen (O5 und O6), für jeden der vier LUTs in einem Slice (A,B,C und D)
- Acht Speicherelemente Flip Flop (FF), davon können vier flakengesteuerte D-FlipFlop oder pegelabhängige Latches konfiguriert werden.
- Multiplexer ermöglichen LUT-Kombinationen von bis zu vier LUTs in einem Slice.
- Einen Carry-Logik um eine dedizierte schnelle arithmetische Addition/Subtraktion in einem Slice durchführen.

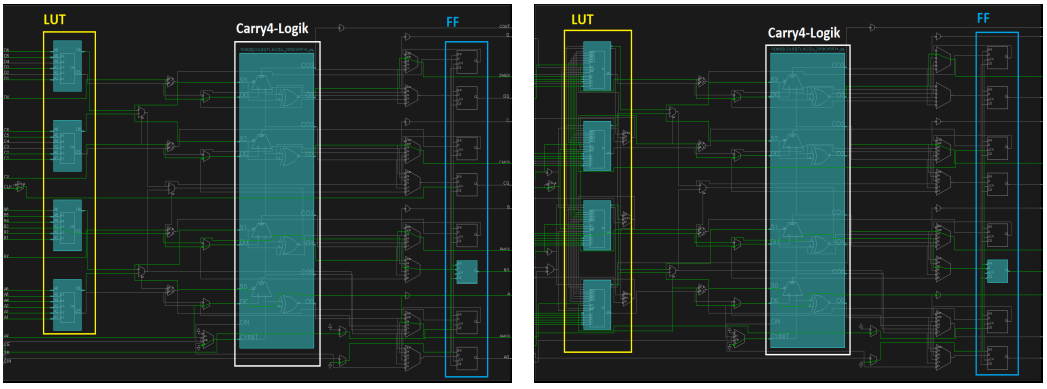


Abbildung 3.4.: BELs auf einer SLICEL (a) und SLICEM (b)

3.2. FPGA Entwurfsablauf

In jedes technisches Systems steht zu Beginn der Systementwurf. Aus gegebener Anforderungen werden Spezifikationen des Systems definiert. Spezifikationen bilden die Grundlagen für abstrakte Systembeschreibung, die alle benötigten Systemkomponenten sowie Schaltplänen enthält, dafür müssen Systembeschreibungen in lesbarer Form vorliegen (Entwurfseingabe).

Typischen Entwurfseingabe Formate sind Hardwarebeschreibungssprachen wie VHDL oder Verilog. Aus einer beispielsweise VHDL-Beschreibung wird zunächst überprüft, durch Simulationen, ob der Entwurf die Spezifikation des Systems erfüllt (Validierung).

Ist die Validierungsphase erfolgreich, folgt dann die Sythese. Hier wird der Entwurf vom Synthesewerkzeug analysiert und in eine Netzliste überführt. Die Netzliste ist unabhängig von der Zieltechnologie und besteht aus Komponenten der Digitaltechnik wie Logikgattern, FFs etc.. .

In der nächsten Phase müssen die sogenannten “Constraints” angegeben werden womit das System betrieben werden soll. Hierzu gehören Pin-Belegung, Taktrate und andere physikalischen Implementierungsdetails. Anhand von Constraints sowie die Netzliste werden folgende Schritte (*Map und PAR*) abgearbeitet und zu einem Schritt zusammengefasst.

Beim *Mappen* (engl. abbilden) werden Logik auf die verfügbaren Ressourcen auf das Ziel-FPGA abgebildet. Als Beispiel werden Boole’schen Gleichungen auf LUTs

realisiert, Speicherelemente werden auf geeigneter FF ausgewählt.
Beim Platzieren und Verdrahten Place and Route (PAR) werden z.B LUT oder FF Hardware-Ressourcen auf dem FPGA zugewiesen (platzieren) und nachfolgend Verbindungen mit dem Verdrahtungsressourcen definiert (verdrahten).
Demnach steht nun eine vollständige Funktionsbeschreibung, die auch Design bezeichtet wird. Zur Überprüfung des Designs sollte eine Timingsimulation durch geführt werden um zu schauen, ob der Entwurf allen zeitlichen Anforderungen erfüllt.
Abschließend kann im letzten Schritt ein Bitstrom generiert und auf das Ziel-FPGA geladen werden.

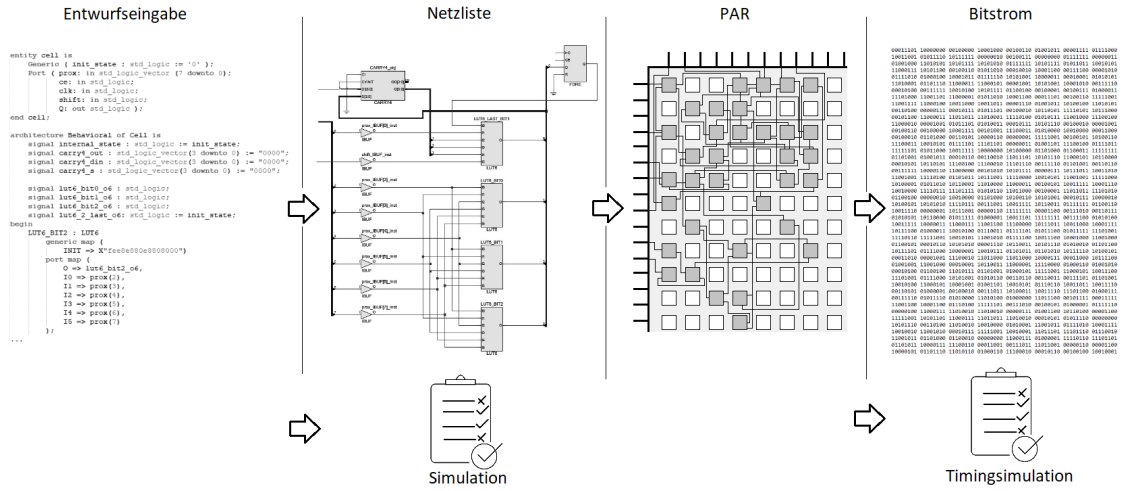


Abbildung 3.5.: Grundlegende Entwurfsablauf. Begin von links nach rechts

3.2.1. Relatively Place Macros

Definieren von relativ platzierten Makros

3.3. Xilinx Primitives

Primitive und Makros sind die "Bausteine"von Komponenten Bibliotheken. Xilinx-Bibliotheken bieten sowohl Primitive als auch allgemeine Makrofunktionen auf hoher Ebene. Primitive sind grundlegende Schaltungselemente, wie AND- und OR-Gatter. Jedes Primitiv hat einen eindeutigen Bibliotheksnamen, Symbol und eine Beschreibung. Makros enthalten mehrere Bibliothekselemente, die Primitive und andere Makros enthalten können.

Makroprimitiven

3.3.1. FDRE

3.3.2. LUT6_5

3.3.3. CARRY4

Kapitel 4

Verwandte Arbeiten

Automatic Generation of Cellular Automata on FPGA von Andre Costa Lima und Jo~ao Canas Ferreira.

Thesis Ziele.

Kapitel 5

Zelluläre Automaten in FPGA

Warum ist das eine gute Lösungsansatz ?
Ziel... (Resourcen einsparen, kürzere Verdrahtung, da alle Zellen in Hardware Zellen untergebracht werden, ebenfalls liegen alle benachbarten Zellen physikalisch nebeneinander.)

5.1. Hardwarespezifikation

Zedboard... Entwicklungssystem...

5.2. Implementation einer GoL-Zelle

5.2.1. Erstes Design

Das erste Design erläutern. Simple und Intuitiv wie in Software. Problem: Resourcen verschwendung.

5.2.2. Logik Minimierung

Erläuterung warum und wie die Methodik zur Minimierung funktioniert. Kompaktheit durch das Einsetzen der Primitives.

5.2.3. Ergebnis

Synthese-Ergebnis zeigen. Ergebnis der Zelle-Testbench zeigen (Simulationsdatei).

5.3. Spielfeld Implementation

Die Idee das Spielfeld als Schieberegister.

5.3.1. Festlegung benachbarten Zellen

Durch das Schieben der Startkonfigurationsdaten müssen die Reihenfolgen der benachbarten Zellen neu festgelegt werden.

Vorher -> Nachher

5.3.2. Ergebnis

Synthese-Ergebnis Ergebnis der Spielfeld-Testbench zeigen.

Kapitel 6

Prozessorsystem zur Datentransfer

Das MicroBalze-System wird hier gewählt, (Gründe nennen)

6.1. Das CA Core Modul

FSM

6.2. GoL IP-Core

6.3. Blockdiagramm

6.4. Tcl-Skript zur Platzierung von GoL-Zellen

6.5. GoL-Modul Treiber Implementation

6.6. Ergebnis

Kapitel 7

Auswertung und Ergebnisse

7.1. Zedboard Auslastung

Vergleiche für ohne local constraints und mit local constraints

7.1.1. Timing

WNS... Für unterschiedliche Systemfrequenzen

7.1.2. Resourcen

Für unterschiedliche WxH

7.2. Microblaze vs Zynq-7000

Vergleich uC und fpga

Kapitel 8

Fazit

Abkürzungsverzeichnis

AXI Advanced eXtensible Interface

BEL Basic Element of Logic

CLB configurable logic block

FF Flip Flop

FPGA Field Programmable Gate Arrays

GOL Game of Life

LUT Look Up Table

PAR Place and Route

PS Processing System

PL programmierbaren Logik

VHDL VHSIC Hardware Description Language

ZA zelluläre Automaten

Tabellenverzeichnis

2.1. Skizze zur Visualisierung der Regeln im GOL. Es sind jeweils drei mögliche Nachbarschaften von x_{ij} zum Zeitpunkt t_k, welche im folgenden Zeitschritt t_{k+1} zu einem Zustand führen. Weiß steht für tot und grau für lebendig. Dabei ist die Referrenzelle x_{ij} immer das mittlere Feld. 8

Abbildungsverzeichnis

2.1. Beispiel eines zweidimensionalen Gitter: a) rechteckig und b) hexagonales 4
2.2. Beispiel eines zweidimensionalen 4x5 Gitter mit zwei Zuständen. Wobei Q aus s = 2 Zuständen besteht. Weiß q_1 für “tot” und Grau q_2 für "lebendig". 5
2.3. Nachbarschaften für zweidimensionales Gitter: a) Von-Neumann-Nachbarschaft und b) Moore-Nachbarschaft. 5
2.4. Reflektierende Randbedingung am Beispiel Neumann-Nachbarschaft 6
2.5. Periodische Randbedingungen am Beispiel a) Neumann- und b) Moore-Nachbarschaft. 7

3.1. Xilinx ZedBoard 10
3.2. Vereinfaches Modell der Zynq Architektur 10
3.3. Grundlegende Architektur eines PL. 11
3.4. BELs auf einer SLICEL (a) und SLICEM (b) 12
3.5. Grundlegende Entwurfsablauf. Begin von links nach rechts 13

Listings

Anhang A

Erster Anhang

Hier ein Beispiel für einen Anhang. Der Anhang kann genauso in Kapitel und Unterkapitel unterteilt werden, wie die anderen Teile der Arbeit auch.

Anhang B

Zweiter Anhang

Hier noch ein Beispiel für einen Anhang.