FPGA-Systementwurf

Rosbeh Etemadi
Universität Paderborn

29. Mai 2007

Die Arbeit behandelt die drei grundverschiedenen Entwicklungssprachen VHDL, Matlab/Simulink, sowie Handel-C. Es werden Grundlagen auf den die Sprachen basieren, und ihre Vor- sowie Nachteile erläutert. Man wird mit dieser Arbeit alle drei Sprachen soweit kennen lernen, dass man sich im Nachhinein selbst in die interessanteste Sprache weiter einarbeiten kann.

Inhaltsverzeichnis

1 Motivation

In dieser Arbeit wird der grundsätzlichen Frage nachgegangen, welche Entwicklungsansätze es gibt, um Systeme für FPGAs zu entwickeln. Einige dieser Entwicklungsansätze werden schon seit längerem mit Erfolg in der Praxis angewandt. Diese sind Hardwarebeschreibungssprachen, wie z.B. VHDL oder VERILOG. Nur sind mit Hardwarebeschreibungssprachen auch Vor- und Nachteile gegeben, die größere Einsatzgebiete, sowie Entwicklung erschweren. Um diesen Nachteilen aus dem Weg zu gehen, kann man andere Ansätze in Betracht ziehen, was im Rahmen dieser Arbeit geschehen soll.

2 FPGAs

In diesem Kapitel wird eine kürze Einführung in FPGAs gegeben, um die Anforderungen an die Entwicklungssprachen näher erleutern zu können. Es wird eine grundsächliche Betrachtung, sowie Realisierungsgrenzen der FPGAs geben. Weiter soll der strukturelle Aufbau eines speziellen FPGAs in Form eines Bildes veranschaulicht werden. Und als letzten Punkt wird der Entwicklungsablauf der FPGAs näher gebracht, der die Hauptangabe an Anforderung für Entwicklungssprachen darstellen wird.

2.1 Grundlagen von FPGAs

FPGA steht für Field Programmable Gate Array und ist im allgemeinen Sinne ein programmierbarer Halbleiterbaustein. FPGAs bestehen aus drei Grundelementen, diese sind:

- I/O-Blöcke
- Logische Blöcke (CLB: Configurable Logic Block)
- Verbindungen zwischen Logischen Blöcken

Unter I/O-Blöcken versteht man die Eingänge, Ausgänge, sowie die Kombination der beiden(Bidirektional).Logische Blöcke (CLBs)sind die Konfigurationseinheiten, die wiederum aus look-up tables (LUTs) bestehen. Look-up tables sind im Prinzip mit Wahrheitstafeln zu vergleichen. Sie beschreiben das Verhalten von Komponenten, die auf einem CLB realisiert werden. Der dritte Bestandteil sind die Verbindungen der I/O-Blöcke mit den CLBs. Programmiert werden die FPGAs, indem man Schalter auf den FPGAs setzt, die z.B. Verbindungen zu gewissen CLBs mit den dazugehörigen I/O-Blöcken darstellen. Es gibt unterschiedliche Schaltertechnologien auf den FPGAs. Die häufigsten sind ANTI-FUSE und SRAM. ANTI-FUSE sind Schalter, die nach einmaligem programmieren festgebrannt werden. Das hat den Vorteil, dass es ohne Strom trotzdem mit der programmierten Konfiguration erhalten bleibt. SRAM dagegen ist re-programmierbar, denn die Schalter werden nicht wie bei der ANTI-FUSE Technologie festgebrannt. Damit man aber die Konfiguration nicht bei einem Stromentzug verliert, besitzen diese

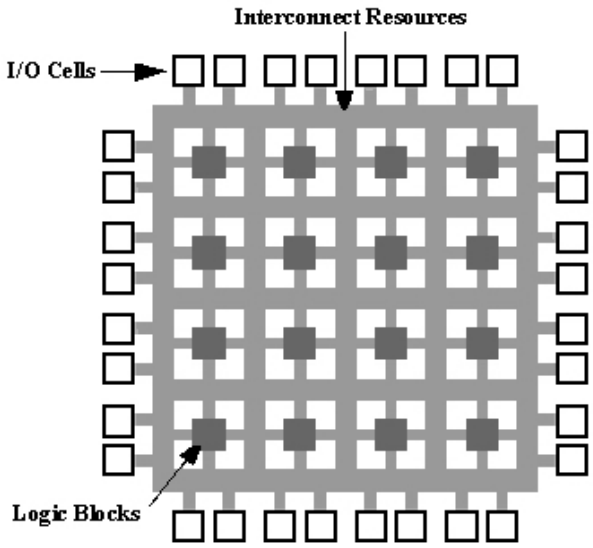

Abbildung 1: Struktureller Aufbau eines FPGAs

FPGAs einen zusätzlichen Speicher, indem die Konfiguration gesichert wird. Es besteht die Möglichkeit, mehrere Konfigurationen zu speichern, und diese nach Belieben auf das FPGA zu spielen. Die am meisten genutzten sind daher auch FPGAs mit der SRAM Technologie, weil sie diese hohe Flexibilität besitzen. Zur besseren Veranschaulichung wird auf Abbild 1 der strukturelle Aufbau eines FPGA abgebildet.

2.2 Was wird mit FPGAs realisiert

Man kann auf FPGAs logische Bausteine wie AND, OR, NOT, NOR und NAND realisieren, sowie die komplexe Verbindung von logischen Bauteilen zu Komponenten, wie Decoder oder Encoder herstellen. Weiterhin sind auch mathematische Funktionen kein Problem für FPGAs. Im Grunde müsste die Frage lauten, was man mit FPGAs nicht realisieren kann. Denn jedes denkbare digitale System ist auf FPGAs umsetzbar. Umso verständlicher ist es, dass FPGAs in den verschiedensten Einsatzfeldern genutzt werden. Daher produziert die Forschung immer mehr FPGAS, die gleichzeitig leistungsfähiger werden.

2.3 Entwicklungsablauf von FPGAs

Die erste Instanz bei der Entwicklung von Systemen für FPGAs ist das Beschreiben des Systems in einer Entwicklungssprache. Wir gehen hier davon aus, dass man sich die Hardwarebeschreibungssprache VHDL entschieden hat. Wenn man nun einen korrekt programmierten Code erzeugt hat, der das System beschreibt, wird dieser Code synthetisiert. Wenn der VHDL Code synthetisierbar ist, erzeugt man eine FPGA Netzliste. Eine Netzliste beschreibt den VHDL Code durch eine schematische Abbildung des

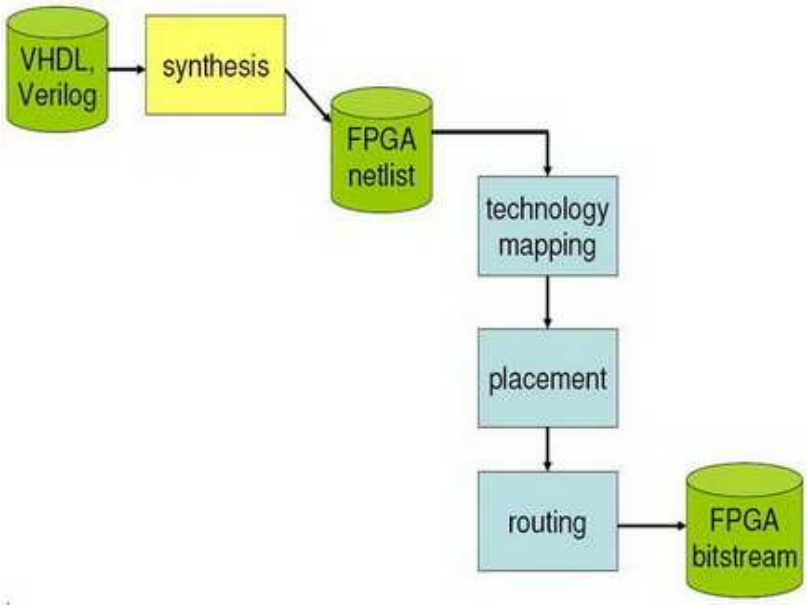

Abbildung 2: Entwicklungsablauf von einem FPGA

Systems. Die nächsten Schritte, die noch gemacht werden müssen, sind dafür da, die Netzliste auf das konkrete FPGA zu mappen. Dafür werden drei Schritte benötigt. Der erste Schritt ist das Technology Mapping. In diesem Schritt wird überprüft, ob man die Netzliste auf das konkrete FPGA transformieren kann. In dem zweiten Schritt wird versucht, eine optimale Platzierung der einzelnen Bauteile des Systems auf den konkreten FPGA zu finden. Dieser Schritt wird als Placement bezeichnet. In dem letzten Schritt, dem sogenannten Routing wird das Verbinden der Bauteile auf den konkreten FPGA behandelt. Die beiden Punkte Placement und Routing sind keine trivialen Probleme. Daher wird das Erzeugen des letzten Bit-Files bei komplexen Systemen meist einige Zeit in Anspruch nehmen. Es ist nicht ungewöhnlich, dass dieses Verfahren bis zu zwei Stunden dauern kann. Wenn man am Ende das gewünschte Bit-File bekommt, muss man dieses noch auf das FPGA spielen. Damit erhält man dann eine FPGA mit der Konfiguration des Systems. In Abbild 2 wird der Entwicklungsablauf nochmals übersichtlich gezeigt.

3 Entwicklungssprache VHDL

In diesem Kapitel wird die erste von insgesamt drei verschiedenen Entwicklungssprachen, und somit auch ihr Entwicklungsansatz besprochen. Es wird zunächst eine Grundlage von VHDL besprochen, sowie eine schematische VHDL Spezifikation. Weiter wird anhand eines Beispielcodes versucht, die Sprache in ihren Grundzügen besser zu verstehen. Sowie ein Ablaufdiagramm mit der Entwicklungsumgebung von Xilinx ISE gezeigt.

3.1 Grundlagen von VHDL

VHDL, welches die Abkürzung von Very High Speed Integrated Circuit Hardware Description Language ist, kann als eine Hardwarebeschreibungssprache bezeichnet werden. Sie ist in Europa die am meisten genutzte Hardwarebeschreibungssprache neben VERILOG. VERILOG weist eine starke Ähnlichkeit mit VHDL auf, und wird in den USA hauptsächlich angewandt. Weiterhin wird VHDL benutzt um komplizierte digitale Systeme zu beschreiben. Die Grundkonzepte von VHDL basieren darauf, dass eine spezifische Hardware in abstrakten Code umgewandelt wird. Des Weiteren ist die Sprache hierarchisch aufgebaut. Das heißt, dass Grundbausteine beschrieben werden, die zu einer neuen Komponente zusammengeschaltet werden. Diese Komponente könnte man nun wieder mit anderen Komponenten oder Bausteinen Zusammenschalten, und eine nächsthöhere Komponente definieren. Dies führt dazu, dass man in der Bibliothek eine Vielzahl von vordefinierten Bausteinen vorfindet. Somit wird vermieden, dass man immer wieder dieselben Bausteine wie z.B. einen Decoder beschreiben muss.

3.2 VHDL Spezifikation

In VHDL gibt es drei Konstrukte, die man nutzt, um eine Hardware zu beschreiben. Das erste ist die sogenannte ENTITY. Sie beschreibt ein Bauteil, indem es als Black Box betrachtet wird. Wenn z.B. ein ODER Bauteil mit zwei Eingängen und ein Ausgang in VHDL beschrieben werden soll, dann beschreibt die ENTITY nur die beiden Eingänge und den einen Ausgang. Nun muss noch das Verhalten dieser ENTITY beschrieben werden. Dies wird in VHDL in der ARCHITECTUR gemacht. Die ARCHITEKTUR beschreibt das Verhalten eines Bauteiles entweder funktional oder strukturell. Man kann mehrere ARCHITEKTUREN für eine ENTITY beschreiben. Um das letztendliche Bauteil zu definieren muss man jetzt noch die beiden Elemente ENTITY und ARCHITEKTUR mit einander verbinden. Dieses wird mit der CONFIGURATION gemacht, sie macht nichts anderes als zu sagen, dass ENTITY-ODER und ARCHITEKUT-ODER zu einer CONFIGURATION zusammen gehören. Daher ist es auch möglich, dass man auch hier mehrere CONFIGURATION definieren kann. In Abbildung 3 wird die VHLD Spezifikation, sowie in Abbildung 4 ein Beispielcode von einem D Flip-Flop, nochmal übersichtlich beschrieben.

3.3 Ablaufdiagramm von FPGA-Synthese mit ISE

In der Abbildung 5 sieht man ein schematisches Ablaufdiagramm mit der Entwicklungsumgebung Xilinx ISE. Man geht so vor, dass man ein XOR Baustein entwickeln will. Wie in Kapitel 3.3 schon erwähnt wird aus dem VHADL Code nach der Plattform unabhängigen Synthese die schematische Darstellung des XOR Bausteins gezeigt. Mit dem IMPLEMENT DESIGN wird in Xilinx ISE das Technologie Mapping, placement, sowie Routing vorgenommen, welches am Ende dann den XOR Bit-File erzeugt. Mit ISE kann man das Bit-File auf das FPGA aufspielen, wenn es an einer Schnittstelle angeschlossen ist.

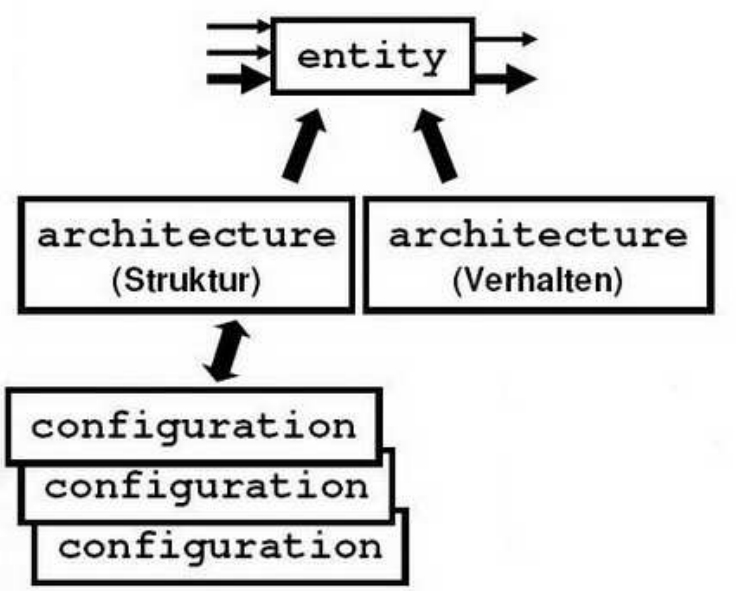

Abbildung 3: allgemeine VHDL Spezifikation

```
entity DFF is
   port ( D, Clk, Reset : in  std_logic;
          Q             : out std_logic );
end entity DFF;

architecture Behavioral of DFF is
begin
 process (Clk, Reset)
  begin
    if (Reset = '0') then
       Q <= '0';
    elsif (Clk'event and Clk ='1') then
       Q <= D;
    end if;
 end process;
end architecture Behavioral;
```

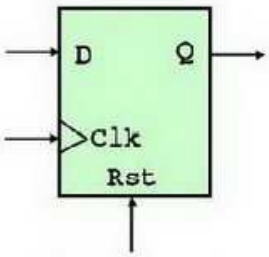

Abbildung 4: VHDL Beispielcode (D Flip-Flop)

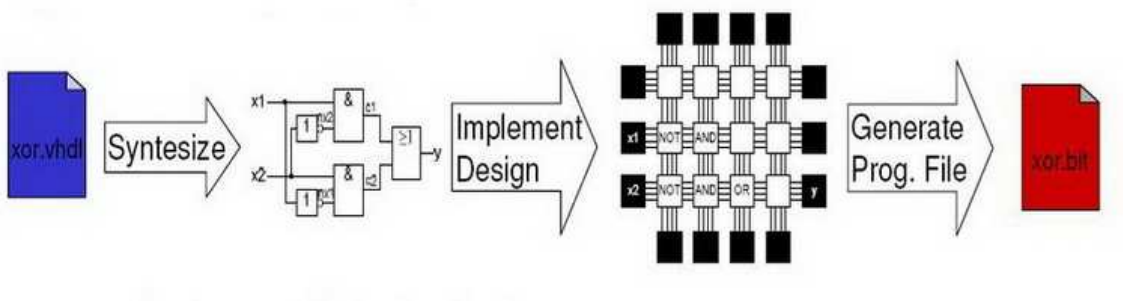

Abbildung 5: FPGA-Synthese mit Xilinx ISE

4 Matlab/Simulink

In diesem Kapitel wird Mathlab, Simulink, sowie ein Screenshot der Entwicklungsumgebung vorgestellt. Zum Abschluss wir dann noch ein Beispielmodell von dem Fibonatschi Generator kurz vorgestellt.

4.1 Was ist Matlab

Matlab ist eine kommerzielle plattformunabhängige Software der Firma The MathWorks. Sie wird benutzt um mathematische Probleme zu lösen und deren Ergebnisse grafisch darzustellen. Man kann Matlab mit Mupad, das an der Uni Paderborn mit entwickelt wurde, vergleichen. Genauso wie Mupad verfügt auch Matlab über einen Satz von speziellen Befehlen, die der Benutzer ausführen kann, indem er die dazugehörigen Parameter übergibt. Beispiele dazu wären Funktionen wie POT oder SOLVE. Bei der PLOT Funktion handelt es sich um eine Funktion, die die gewünschte Funktion visuell darstellt. Die Funktion SOLVE löst eine mathematische Gleichung. Speziell wurde Matlab entwickelt um numerischen Berechnung zu lösen. Daher lassen sich Problemgebiete der linearen Algebra sowie der numerischen Analysis berechnen und simulieren.

4.2 Was ist Simulink

Simulink ist eine Erweiterung von matlap und stellt eine spezielle Toolbox von Matlab dar. Mit Simulink ist es dem Benutzer möglich, Systeme zu modellieren und zu simulieren. Man sollte hinzufügen, dass Simulink nicht standardmäßig mit Matlab mitgeliefert wird. Um die Systeme zu modellieren stellt Simulink eine Reihe von Modellen zur Verfügung. Modelle sind in Simulink auch selbst definierbar, was z.B. die Firma Xilinx für Hardwareentwicklungen getan hat. Darunter kann man verstehen, das Xilinx vorgefertigte Bauteile wie Flip-Flops, Multiplexer usw. in der Bibliothek von Simulink zur Verfügung gestellt haben. Diese Bauteile werden in Simulink mit Hilfe des Systemgenerators in eine Hardwarebeschreibungssprache übersetzt. Der Vorteil ist, dass die Bauteile an sich schon sehr optimiert sind für Hardwarebeschreibungssprachen wie z.B. VHDL. In der Abbildung 6 sieht man einen Screenshot von der Entwicklungsumgebung Matlab/Simulink.

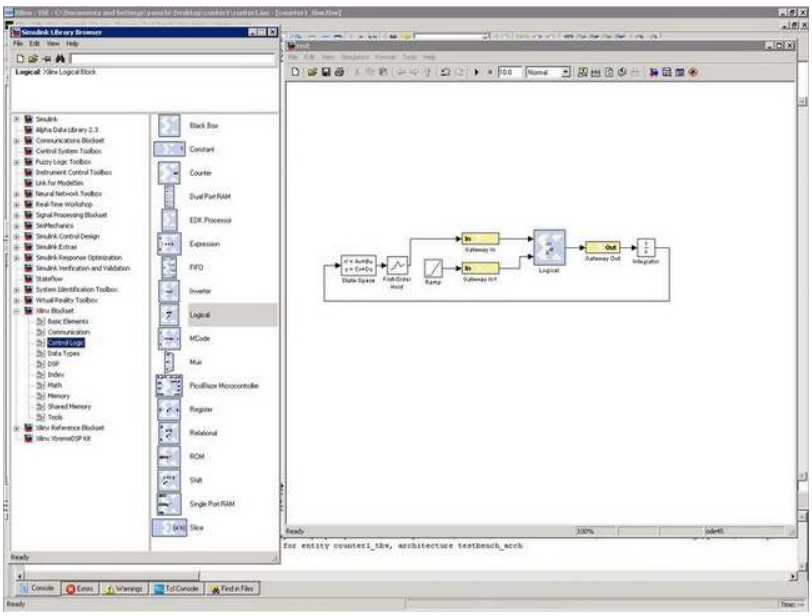

Abbildung 6: Entwicklungsumgebung Matlab/Simulink

4.3 Simulink Beispielmodell

Wir sehen auf der der Abbildung 7 einen Screenshot von einem Beispielmodell. Es handelt sich hier um einen Fibonatschi Generator. Dieser Generator besitzt zwei Eingaben und zwei Ausgaben. Die blauen Bauteile, sowie das mittlere Bauteil bilden den Fibonatschi Generator. Die beiden Komponenten, die mit den gelben IN Komponenten verbunden sind, sind Signalgeber. Diese füttern das System, womit hier die gelben-, blauen- und das mittlere Bauteil gemeint ist, mit Daten. Damit sind die ersten Bedingungen erfüllt um eine Simulation des Systems laufen zu lassen. Weiter sind an den gelben out Bauteilen ein SCOPE angeschlossen. Dieser SCOPE-Block zeichnet den Wert seines Eingabesignals, also den des Ausgabesignal über das gesamte Zeitintervall der Simulation, auf. Mit dem WAVESCOPE kann man sich dann auch das Zeitdiagramm anzeigen lassen um zu überprüfen, ob das System das gewünschte Verhalten wiedergibt. Zu guter letzt muss noch der Systemgenerator in jedem zu synthetisierenden System enthalten sein. Denn der Systemgenerator erzeugt letztendlich den Hardwarebeschreibungscode z.B. VHDL oder VERILOG. Man könnte jetzt auf die Idee kommen, dass mit dem erzeugten VHDL Code weitergearbeitet werden könnte, indem man ihn versucht noch weiter zu optimieren. Leider muss man dazu sagen, dass dies bei hinreichend großen Projekten eine sehr komplizierte Aufgabe werden kann. Denn der erzeugte VHDL Code wird strukturell geschrieben. Das heißt, dass dieser kaum noch zu lesen ist. Ein VHDL Programmierer würde das Verhalten eines Systems funktional beschreiben. Somit benutzt der VHDL Programmierer Anweisungen, Schleifen usw., was den Code noch überschaubarer macht.

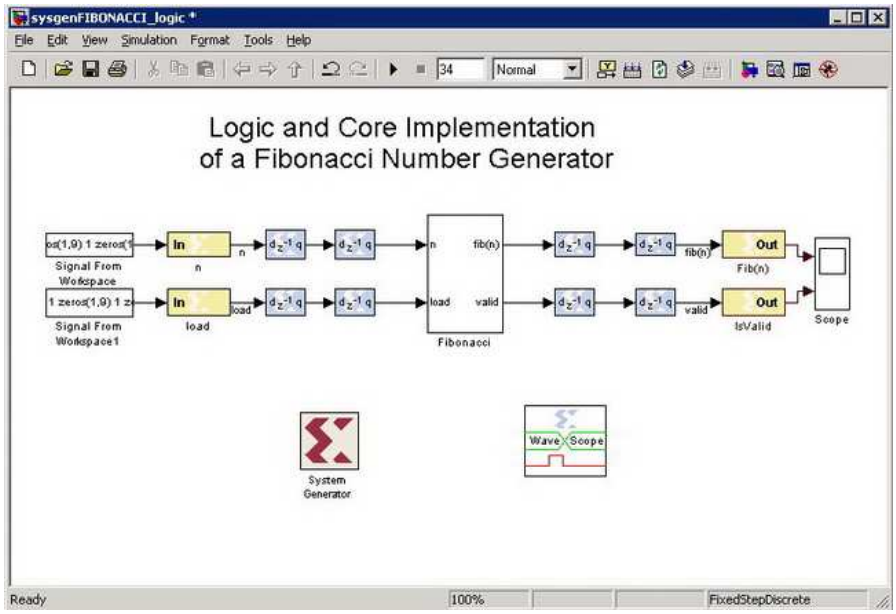

Abbildung 7: Simulink Beispielmodell (Fibonatschi Generator)

5 Entwicklungssprache Handel-C

In diesem Kapitel soll die dritte Entwicklungssprache vorgestellt werden. Es werden grundlegende Definitionen von HANDEL-C erläutert. Auch der Grund der Entwicklung von HANDEL-C wird hier behandelt, da die Sprache einige Besonderheiten in Vergleich zu den anderen Hardwarebeschreibungen aufbietet. Beispielcode und ein übersichtlicher Entwurfsablauf wird dann im Anschluss in Form von einem Diagramm, sowie anhand von Textstellen erläutert.

5.1 Was ist Handel-C

Entwicklung von HANDEL-C begann Mitte der 90er an der Oxford-University. Sie wurde von der Firma Celoxica (früher Embedded Solutions) letztendlich entwickelt. Die Programmiersprache dient dazu, hoch abstrahierte Algorithmen direkt in Hardware zu übersetzen. Das Besondere an HANDEL-C ist, dass sie eine Modifizierung von C ist. Die Sprache HANDEL-C ist so erweitert worden, dass man sie mit Signalverarbeitungsroutinen spezifizieren kann.

5.2 Entwicklungsgrund von Handel-C

Es ist eine berechtigte Frage, warum es HANDEL-C überhaupt gibt, denn Hardwarebeschreibungssprachen gibt es bereits. Und die Beschreibung der spezifischen Hardware wird mit Erfolg in der Praxis angewandt. Also warum sollte man eine neue Sprache definieren? Der Hauptgrund der Entwicklung von HANDEL-C basiert darauf, dass man die Entwicklungszeit von FPGAs drastisch verkürzen wollte. Vor allem war die Intention die,

dass man das Programmieren der FPGAs nicht nur denen vorbehält, die unter genügend Hardwarewissen verfügen. Daher richtet sich HANDEL-C hauptsächlich an Softwareentwickler und nicht an Hardwareentwickler. Man kann sagen, dass HANDEL-C nicht eine echte Hardwarebeschreibungssprache an sich ist. Sie ist wie in Kapitel 6.1 schon erwähnt eine Modifikation von C. Die Entwickler wollten die Entwicklungszeit verkürzen indem sie es möglich machten, dass man schon vorhandene C Algorithmen wiederverwerten konnte. Man kann dann die vorhandenen C Algorithmen leicht auf HANDEL-C portieren. Dafür sind natürlich einige Veränderungen von Nöten. Diese sind aber leichter zu realisieren, als die Übersetzung in eine komplette Hardwarebeschreibungssprache. Um überhaupt dafür zu sorgen das Softwareentwickler spezifische Hardware mit HANDEL-C beschreiben können, muss gegeben sein, dass die Hardware-relevanten Probleme versteckt, und vom Compiler selbst gelöst werden.

Es sollen noch einige Beispiele für die Erweiterungen von HANDEL-C aufgezählt werden. Einer der wichtigsten Konstrukte, die es in HANDEL-C gibt, ist die Parallelität. Mit PAR wird ein selbst zu definierender Bereich als parallel abzuarbeiten markiert. Weiter gibt es nur einen Datentyp in HANDEL-C. Auch Signalverzögerungsroutinen sind in HANDEL-C vorhanden und auch zwingend nötig um Hardware Realistik zu beschreiben. Man muss dafür sorgen, dass Bauteile auf andere Bauteile warten können. Weiterhin braucht man den Input und Output um Hardwareeingaben und -ausgaben zu definieren, um sie auch mit anderen Bauteilen verbinden zu können. Dies geschieht in HANDEL-C mit dem Befehl CHANIN und CHANOUT. Pointer die es in C gibt, werden in HANDEL-C nicht benötigt, und sind daher auch nicht vorhanden.

Auf Abbildung 8 sieht man ein Beispielcode von HANDEL-C, der einen Summierer darstellt. In dem Abbild kann man erkennen, dass auch in HANDEL-C alles mit der bekannten VOID MAIN Methode startet. Die beiden Zeilen 2 und 3 beschreiben die Variablen, die vom Typ unsigned int sind mit dem Namen sum und data. Die jeweilige Zahl nach int besagt wie groß die Bitgröße der Variablen sum und data ist. CHAININ und CHAINOUT in Zeile 4 und 5 beschreiben die Eingaben und Ausgaben. Direkt darunter in Zeile 7 wird die variable sum mit 0 initiiert. Mit der DO Anweisung in Zeile 8 wird nun erstmal definiert, was die Eingabe sein soll, also der Input. Hier wird der Input mit den Werten von data gespeist. In Zeile 10 wird sum mit data addiert und in sum gespeichert. Nur ist anzumerken, dass data 8 Bit, und sum 16 Bit breit sind. Um hier eine korrekte Addition zu herhalten, müssen die beiden Variablen auf dieselbe Bitgröße gebracht werden. Was mit den Befehl 0 @ passiert. o @ füllt die fehlende Bitbreite einfach mit Nullen, und man erhält danach dieselbe Größe, mit der man nun addieren kann. Zeile 11 definiert mit der Whileschleife das die Addition solange geschehen soll, wie data nicht 0 ist. Zu guter letzt wird in Zeile 13 der Output mit sum definiert. Also werden 16 Bit ausgegeben.

5.3 Handel-C Entwurfsablauf

In diesem Kapitel wird nochmals der Entwicklungsablauf mit HANDEL-C beschrieben. Man fängt an mit dem Portieren der schon verfügbaren C Algorithmen, die man auf HANDEL-C transformieren muss. Mit den dazugehörigen Simulator von HANDEL-C

```
1  void main(void) {
2      unsigned int 16 sum;
3      unsigned int 8 data;
4      chanin input;
5      chanout output;
6
7      sum = 0;
8      do {
9          input ? data;
10         sum = sum + (0 @ data);
11     } while (data != 0);
12
13     output ! sum;
14 }
```

Abbildung 8: Handel-C Beispielcode (Summierer)

kann der Code überprüft und solange modifiziert werden, bis man ein korrekt funktionierenden HANDEL-C Code erhält. Im nächsten Schritt werden die Ausgänge mit den Eingängen des externe Hardwarebausteines verbunden. In diesem Schritt wird sozusagen die Software mit der Hardware verschmolzen. Der Compiler von HANDEL-C erzeugt dann noch die Netliste und mit dem gewohnten Verfahren wird dann das Bit-File erzeugt. Dieses kann dann auf das spezifische FPGA geladen werden. Auf Abbildung 9 sieht man nochmal Entwicklungsablauf mit HANDEL-C als übersichtliches Diagramm

6 Fazit

Wir haben in dieser Arbeit drei Entwurfsansätze kennen gelernt. Der erste basierte auf der Hardwarebeschreibungssprache VHDL. Der zweite war Matlpab/Simulink, welches grafisch per Drag and Drop ein System modellieren kann, und auf der Hardwarebeschreibungssprache basiert. Und zu guter letzt HANDEL-C, das versucht die Entwicklung von FPGAs zu beschleunigen, indem es Software und Hardware miteinander eng gekoppelt.

Es gibt nun viele Vor- und Nachteile zu den einzelnen Ansätzen. Ein großer Nachteil bei VHDL ist es, dass man ein gutes Hardwareverständnis voraussetzen muss. Dafür ist es möglich die Hardware sehr detailliert zu beschreiben. Und natürlich bringt dieses mit sich, dass man den Code auch stark optimieren kann. Weil der VHDL Code meistens funktional geschrieben wird, also sehr viele Anweisungen, Schleifen usw. benutzt werden, kann dies dazu führen, dass man den Code nicht synthetisieren kann. Dies kann einem ungeübten VHDL Programmierer schnell passieren. Auch der ungewohnte Aufbau von VHDL kann Neueinsteiger schnell abschrecken mit VHDL zu arbeiten.

Matlab/Simulink dagegen ist sehr intuitiv aufgebaut. Man entnimmt den Bibliothe-

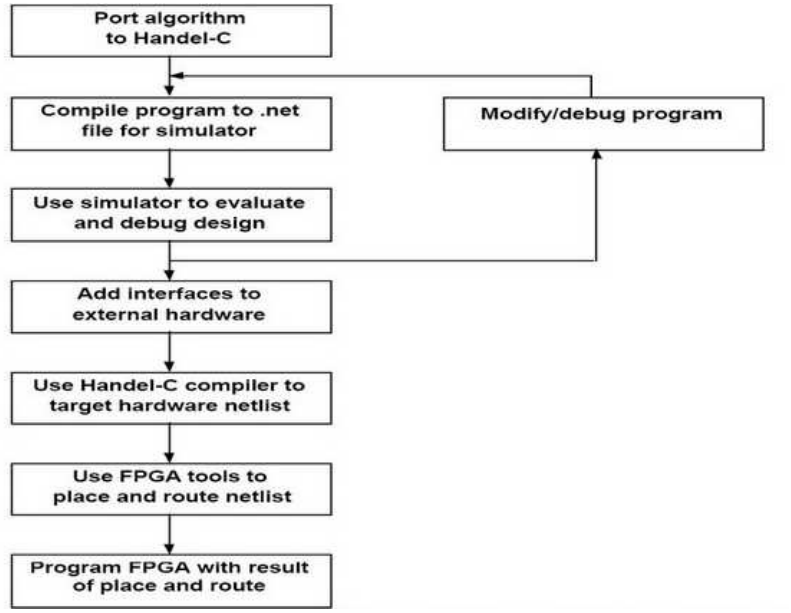

Abbildung 9: Handel-C Entwurfsablauf

ken ihre fertigen Bausteine und baut sich Stück für Stück ein komplexes System auf, das man auch dank des System Generaters synthetisieren kann. Weil die Bauteile die von Xilinx zur Verfügung stehen, auch gut optimiert sind, hat man im Endergebnis brauchbare Bit-Files. Der Code muss aber nicht das Optimum sein, was man schnell als Nachteil bezeichnen könnte. Der Code der vom System Generator erzeugt wird ist meist auch nicht weiter optimierbar, da der VHDL-Code strukturell aufgebaut ist, und sehr unübersichtlich wird. Ein anderer Nachteil von Simulink ist die Fehlerbehandlung. Sie ist nicht immer hilfreich um zu erkenn wo der Fehler liegt. Daher kann es aufwendig werden bei hinreichend großen Projekten die Fehler zu beheben.

Kommen wir noch zum letzten Entwicklungsansatz HANDEL-C. HANDEL-C benötigt kein großes Hardwareverständnis, welches ein großer Pluspunkt ist. Es ermöglicht jeden, der mit der Softwareentwicklung vertraut ist, HANDEL-C schnell zu erlernen. Auch die Wiederverwendbarkeit von schon vorhandenen C Algorithmen vereinfacht das Spezifizieren von Hardware. Solange man HANDEL-C für kleine Projekte nutzt geht alles gut. Hat man allerdings vor ein großes Projekt umzusetzen, so wird HANDEL-C nicht selten Netzlisten erzeugen, die viele Ressourcen fressen. Somit sind sie auf FPGAs nicht mehr umsetzbar. Man müsste dann den Code nachträglich optimieren, was sehr aufwendig wird und auch Hardwareverständnis erfordert. Aber genau dieses sollte der Vorteil von Handel-C sein, was unglücklicherweise nicht immer der Fall ist. Wie wir erfahren haben, ist HANDEL-C im Vergleich zu den Hardwarebeschreibungssprache neu. Daher ist es möglich, dass diese Probleme, die HANDEL-C noch aktuell mit sich bringt, im Rahmen von einigen Forschungen bald erledigen sein werden. Was dann HANDEL-C zu einem mächtigen Werkzeug für die Entwicklung von Hardware auf FPGAs macht.

Tabelle 1 zeigt nochmals die 3 Entwicklungssprachen als Übersicht

	VHDL	Matlab/Simulink	Handel-C
Vorteile	Beschreibt Hardware detailiert	Code ist meist eher synthesefähig	Benötigt keine hohes Hardwareverständnis
	Ermöglicht hohe Optimierung	Sehr übersichtlich da Schaltungen visuell angezeigt werden	Schnelle Entwicklung bei kleinen Projekten möglich
		Ermöglicht schnelles Prototyping	
Nachteile	Erfordert Hardware-verständnis	Muss nicht optimalster VHDL Code sein	Bei großen Projekten sind meist Netzlisten nicht umsetzbar
	Code ist nicht immer synthesefähig	Fehlerbehandlung kann kompliziert werden	Nachträglich Optimierung sehr aufwendig
Modell-umsetzung	Programmcode	Grafisch	Programmcode

Tabelle 1: Vergleichstabelle der 3 Entwicklungssprachen

Literatur

[1] Marco,Platzner;Hardware/Software Codesign,*Fachgebiet Technische Informatik Universität Paderborn*,*Skriptum zur Vorlesung*, SS2005

[2] Manfred, Schimmler;FPGA-Entwurf mit VHDL,*Institut für Informatik,Lehrstuhl Technische Informatik*, *Christian-Albrechts-Universität zu Kiel*,2004

[3] David, Krutz;Ein Betriebssystem für konfigurierbare Hardware*Mathematisch-Naturwissenschaftlichen Fakultät II*, *Humboldt-Universität zu Berlin*,2006

[4] Thomas, Reinemann;Verfahren zur direkten Implementierung von Algorithmen auf Gatterebene*Fakultät für Maschinenbau*,*Otto-von-Guericke-Universität Magdeburg*,2003

[5] A., Bosl;Einführung in das Programm MATLAB / SIMULINK*Labor Regelungstechnik*,*Hochschule Ravensburg-Weingarten*,2006

[6] U., Wohlfarth;SIMULINK Lineare und nichtlineare Systeme*Vorlesung Matlab/Simulink*

[7] Stefan Ihmor,Thomas Lehmann,Marco Platzner;VHDL-Crashkurs *Vorlesung GTI / GRA*,*Uni Paderborn*,SS 2006

[8] Peter J., Ashenden;VHDL Cookbook *Dept. Computer Science University of Adelaide*, *South Australia*,1990