Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW 2 - SS 11

Erik Andresen

Parallele Programmierung von Eingebetten SMP-Plattformen am Beispiel von Spurführungs-Algorithmen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Erik Andresen

Thema der Ausarbeitung

Parallele Programmierung von Eingebetten SMP-Plattformen am Beispiel von Spurführungs-Algorithmen.

Stichworte

SMP, Multiprocessor, OpenMP, Pure Pursuit, Follow the carrot, Parallelisierung, eingebettete Systeme, autonomes Fahrzeug, Spurführungs-Algorithmen

Kurzzusammenfassung

In eingebetteten Systemen werden vermehrt Mehrkernprozessoren eingesetzt, da diese durch die geringere Taktfrequenz bei zusätzlicher Rechenleistung weniger Strom verbrauchen. Diese Ausarbeitung gibt einen Überblick über Lösungsmöglichkeiten, wie OpenMP und MPI, mit denen sich bestehende Programme für mehrere Prozessoren parallelisieren lassen. Als Anwendung dienen dabei Spurführungs-Algorithmen von denen eine Auswahl, die aktuell verwendet werden, vorgestellt wird. Zum Abschluss wird ein Algorithmus als Beispiel parallelisiert.

Hamburg, 30. September 2011

Inhaltsverzeichnis

1 Paralleles Programmieren auf eingebetteten SMP-Plattformen

Im Projekt „High Performance Embedded Computing“ im Fachbereich Informatik an der HAW-Hamburg sollen Algorithmen auf eingebetteten System ausgeführt werden, also auf Hardware die über weniger Rechen- und Speicherkapazitäten verfügen als aktuelle Desktop-Computer. Als Anwendung dient dabei ein autonom fahrendes Fahrzeug das ohne menschliches Eingreifen einer Fahrspur folgen und dabei Hindernissen ausweichen soll. Ausgeführt wird die Anwendung dabei auf FPGAs, also rekonfigurierbaren Schaltkreisen die als System-On-Chip Hardware-Beschleuniger-Module und Mikroprozessoren auf einem Chip vereinen.

In Anwendung 1 wurde als Einstieg in das Thema HPEC mathematische Algorithmen zur Synthese in AccelDSP zu Hardware-Beschleuniger-Modulen synthetisiert [1]. Die Beschleunigung erfolgte dabei durch Parallelverarbeitung, so hat z.B. eine Amerikanische Bank die Berechnung der Risiken ihrer Wertpapiere von mehreren Stunden auf einige Sekunden herabgesetzt indem sie ihre Supercomputer mit parallel arbeitenden FPGA-Bausteinen aufrüsteten [30][5]. Als Beispiel für so eine Beschleunigung in Hardware diente in Anwendung 1 der Kalman-Filter der genommen werden kann um Schätzwerte für die aktuelle Position eines Objektes zu liefern, also z.B. wie ein Fahrzeug relativ zu einer Fahrspur oder einem Hindernis steht. Ist die Fahrspur und die eigene Position vom Fahrzeug erkannt muss das Fahrzeug als nächstes errechnen wie es fahren muss um der Fahrspur zu folgen. Darum soll es in dieser Ausarbeitung zu Anwendung 2 gehen. Dabei basiert diese Ausarbeitung auf der Bachelor-Arbeit von Schneider [27]. Schneider löst die Probleme in Hardware, ob eine Software-Lösung auf Mikroprozessoren zeitlich schnell genug ausgeführt werden kann wird in der Bachelor-Arbeit offen gelassen.

In seiner Arbeit nutzt Schneider einen in Matlab geschrieben Simulator um Spurführungs Algorithmen zu entwickeln und zu testen (Abb. 1). Damit lassen sich die Algorithmen ohne Fahrzeug und Fahrbahn testen.

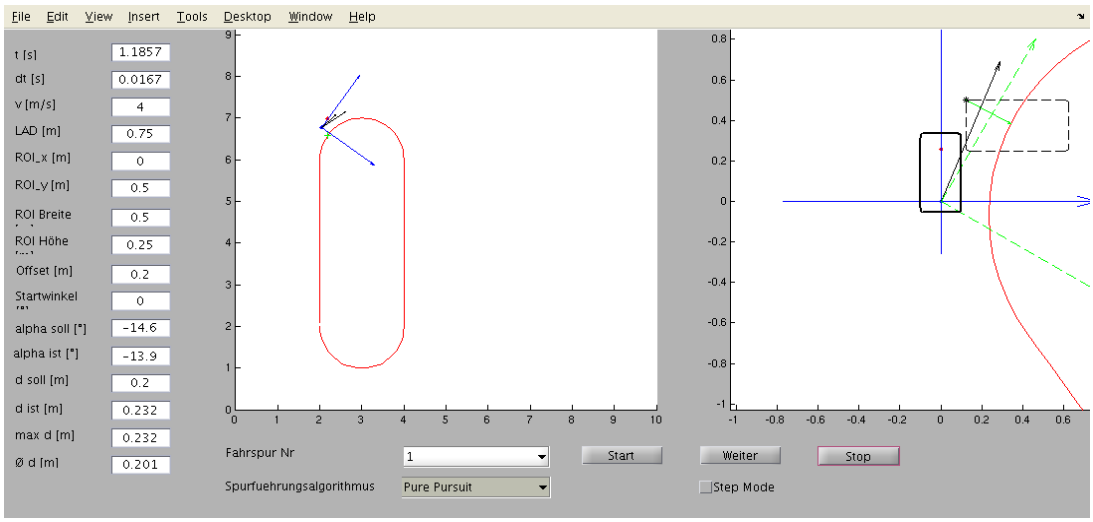

Abbildung 1: Matlab Simulator um Spurführungs Algorithmen außerhalb der Hardware zu testen: Die Parameter können zur Laufzeit ohne aufwendiges neu Kompilieren geändert werden. Die Position und eine Übersicht sind in der linken Grafik dargestellt, während die bei den Algorithmen verwendeten Vektoren in der rechten Grafik zu sehen sind.

In eingebetteten Systemen werden vermehrt Mehrkernprozessoren eingesetzt, da diese durch die geringere Taktfrequenz bei zusätzlicher Rechenleistung weniger Strom verbrauchen [33]. Erste Geräte mit eingebetteten SMP Prozessoren findet man z.B. in Smartphones, u.a. dem HTC Sensation [36] oder Nokia N8 die 2011 auf den Markt gekommen sind.

FPGAs mit einem einzelnen integriertem physischem Prozessor sind seit mehreren Jahren erhältlich. Beispielsweise Xilinx FPGAs mit PowerPC Prozessor [35]. Alternativ können FPGAs auch Prozessoren als Soft-IP-Cores synthetisieren. Dabei werden jedoch teure FPGA Ressourcen gebunden und stehen nicht mehr für Beschleuniger-Module zur Verfügung. Dafür lassen sich so Multi-Core Prozessoren in Verbindung mit FPGAs einsetzen. Xilinx hat unter der Bezeichnung Zynq-7000 Plattform FPGAs angekündigt, die über integrierte physische Dual-Core Cortex A9 Prozessoren verfügen (vgl Abbildung 2). Die Vorteile sind:

- Gegenüber Prozessor Soft-IP-Cores werden keine FPGA-Ressourcen benötigt.
- Bei Prozessoren in eingebetteten Systemen ist ARM der Marktführer und wird deswegen von vielen Software-Plattformen unterstützt [25].
- Der ARM-Prozessor ist von der restlichen FPGA-Hardware entkoppelt, weshalb Hardware- und Software Entwickler unabhängig voneinander parallel arbeiten.

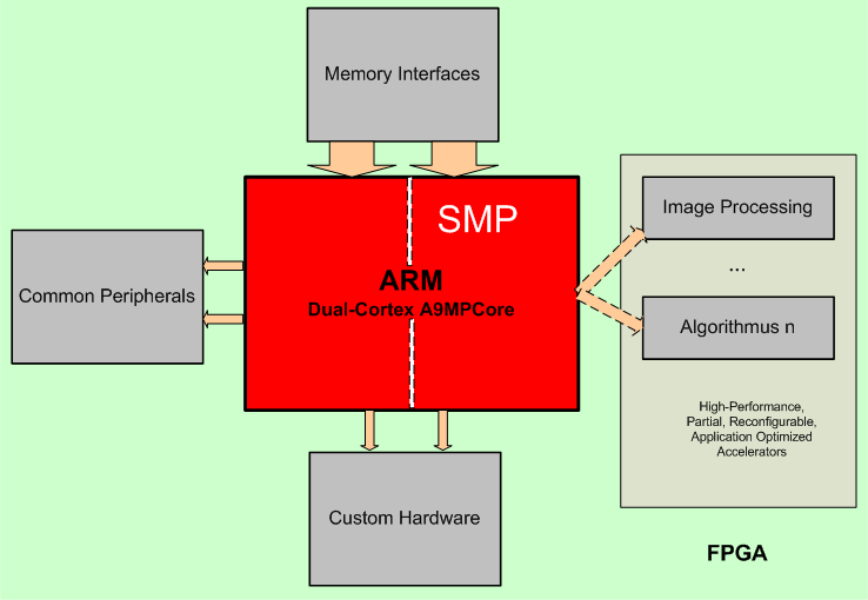

Abbildung 2: FPGA mit Dual-Core Cortex A9 Prozessor. Der FPGA verhält sich wie ein Mikroprozessor mit Peripherie und Speicher-Anschlüssen an dem bei Bedarf Hardware-Beschleuniger-Module synthetisiert werden. Um FPGA Ressourcen zu sparen lassen sich die Module zur Laufzeit austauschen (Reconfigurable).

Vorgestellt werden im Kapitel 2 aktuelle Spurführungs-Algorithmen und im Kapitel 3 Umsetzungsmöglichkeiten um sequentielle Programme zu parallelisieren. Anschließend wird über die Umsetzung einen Spurführungs Algorithmus unter Software zu parallelisieren diskutiert.

2 Folgen einer Fahrspur

Bei dem Thema Spurführungs Algorithmen geht es darum wie schnell ein Fahrzeug fahren darf und wie es lenken muss um zu jedem Zeitpunkt einen Weg (Abb. 3) zu folgen [13]. Dabei besteht der Weg aus einzelnen Punkten „Goal Points“ die nacheinander angesteuert werden. Die Distanz zwischen dem Fahrzeug und dem momentanen Goal Point wird als „Look-ahead distance“ bezeichnet die auch bestimmt wie weit die Goal-Points auseinander liegen dürfen. Dieses Kapitel stellt eine Auswahl an Spurführungs Algorithmen vor.

Abbildung 3: Darstellung einer Fahrbahn die ein autonomes Fahrzeug folgen soll.

2.1 Follow-the-carrot

Der Algorithmus „Follow-the-carrot“ funktioniert wie der Esel dem eine Karotte vor die Nase gehängt wird: Er wird immer direkt auf die Karotte zulaufen. Abbildung 4 beschreibt das Prinzip. Es wird der „orientation error“, also die Differenz zwischen der anzusteuernden Richtung und der tatsächlichen Fahrzeugausrichtung als Eingabe für die Lenk-Regelung genommen. Als Regelung kommt z.B. ein PID-Algorithmus zum Einsatz der den Lenkwinkel berechnet und bestimmt wie stark die Lenkung eingeschlagen wird. Da der Algorithmus nichts anderes macht als direkt auf einen Punkt zuzusteuern ist er einfach zu implementieren, neigt jedoch dadurch Ecken zu schneiden und besonders bei einer kurzen look-ahead-distance zu oszillieren. In der Praxis kommt der Algorithmus aufgrund seiner Nachteile selten zum Einsatz, eignet sich aufgrund seiner Einfachheit jedoch zum Vergleich mit anderen Algorithmen[13].

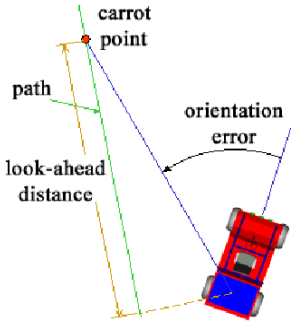

Abbildung 4: Der Esel folgt der Karotte: Das Fahrzeug bildet eine Linie von seinem Mittelpunkt zum Zielpunkt. Die Abweichung zwischen dem Zielpunkt und der tatsächlichen Fahrzeugausrichtung wird als „orientation error“ bezeichnet.

2.2 Pure Pursuit

Der „Pure Pursuit“ Algorithmus kommt aus dem militärischen Bereich wo er seit ca 1969 in Flugkörpern eingesetzt wurde um ein Ziel zu verfolgen [26]: Das Ziel wird nicht wie bei Follow-the-carrot direkt angesteuert, sondern wie in Abbildung 5 zu sehen wird ein Kreis gebildet der durch das Fahrzeug und den Zielpunkt durchgeht auf dem das Fahrzeug dann lang fährt [6]. In der Praxis ist Pure Pursuit ein häufig eingesetzter Algorithmus. Beispiele für Einsätze gibt es an der CMU[11], DARPA[10], Universidad de Málaga[15] und an der HAW[16]. Von Pure Pursuit gibt es mehrere Varianten, z.B. Feedforward und Adaptive [11] wo z.B. die Goal Points in Kurven enger zusammen liegen als auf Geraden um in Kurven genauer zu steuern.

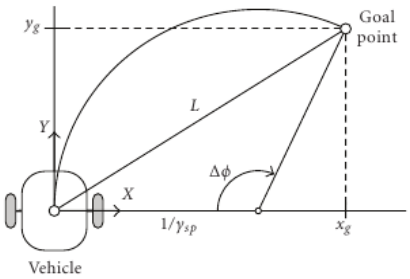

Abbildung 5: Pure Pursuit: Es wird eine Kreis gebildet der durch das Fahrzeug und den Zielpunkt durchgeht auf dem das Fahrzeug langfährt.

2.3 Vector Pursuit

Vector Pursuit ist ein auf der Schraubentheorie basiertes Verfahren das 2000 in einer Doktorarbeit an der Universität von Florida vorgestellt wurde [32]: Bei den vorherigen Algorithmen kann es passieren, daß das Fahrzeug zwar am Zielpunkt ankommt, aber quer zur Fahrtrichtung. Durch Benutzung der Schraubentheorie (Abbildung 6) versucht Vector Pursuit mit korrekter Ausrichtung und Lenkwinkel am Ziel anzukommen. Bei der Schraubentheorie wird die Bewegung von Objekten aufeinander abgebildet. Laut der Doktorarbeit ist das Ergebnis vergleichbar mit Pure Pursuit.

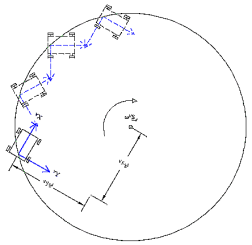

Abbildung 6: Vector Pursuit: Ziel ist es mit korrekter Ausrichtung und Lenkwinkel am Ziel anzukommen.

2.4 Follow-the-Past

Für den Einsatz in der Waldindustrie zum Transport von Baumstämmen wo immer gleiche Wege gefahren werden müssen wurde 2005 Follow-the-past an der Umea Universität in Schweden entwickelt. Dabei werden die Steuerbewegungen eines menschlichen Fahrers (Abb. 7) aufgezeichnet und später von einer Maschine durchgeführt [9].

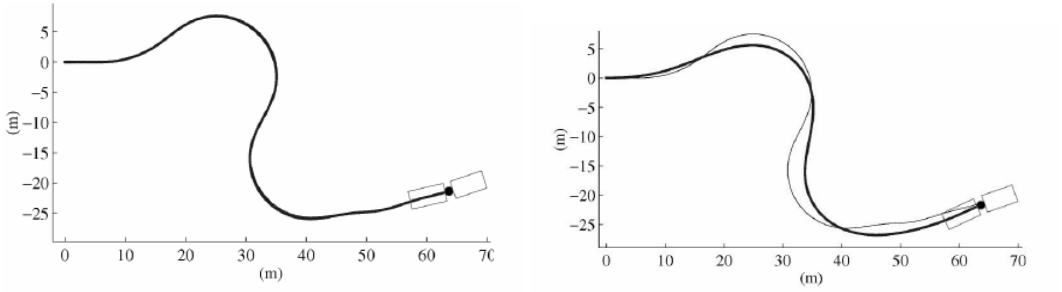

Abbildung 7: Follow the past: Im linken Bild wird Follow the past verwendet, im rechten Bild Pure Pursuit. Die höhere Abweichung von Pure Pursuit gegenüber dem menschlichen Fahren ist besonders in Kurven deutlich erkennbar.

2.5 Künstliche Intelligenz

Weitere mögliche Algorithmen basieren auf Algorithmen der Künstlichen Intelligenz, also z.B. Neuronalen Netzen und Reinforcement learning. Bei Reinforcement learning wird die Strecke so häufig befahren bis sich das Fahrverhalten der KI kaum noch vom bestmöglichen Fahrverhalten unterscheidet. Dieser Zeitpunkt soll schon nach 20 Minuten erreicht sein. Dazu gibt es von 2007 Arbeiten aus Stanford [23] und Osnabrück [24]. Von der HAW gibt es auch Arbeiten zu diesem Thema [17]. Das Verfahren ist datengetrieben, es ist also kein Modell oder Simulation erforderlich. An der HAW wurde dabei mit Q-learning innerhalb von 15 Minuten ein Fahrverhalten erreicht, daß mit einem Fehler von weniger als 0.5m zur Sollspur fuhr.

2.6 Fuzzy Logic

Es gibt Forschungen von 2010 von der Quebec Universität in Kanada und Universität von Jinan in China [14] Fuzzy Logic zur Fahrbahn Verfolgung zu verwenden. Ein beispielhafter Regelsatz kann als Beispiel so aussehen:

- Regel 1: if Input1 and Input2 then output1 = stop, output2 = stop
- Regel 2: if Input1 and not Input2 then output1 = left, output2 = stop
- Regel 3: ...

Der Regelsatz wird nacheinander bis eine Bedingung zutrifft und anschließend werden die Ausgänge entsprechend gesetzt.

2.7 Funnel lane

Funnel lane ist ein 2009 an der Clemson Universität in den USA entwickeltes Verfahren. Die aktuelle Position des Fahrzeugs wird anhand von markanten Punkten auf Bildern (Abb. 8) ermittelt, anschließend wird ein Steuerkommando ausgeführt was dieser Position zugeordnet ist [4]. Erkennt das Fahrzeug seine Position greift es auf einen weiteren Regelsatz zu und ermittelt dadurch wie es fahren muss, als Beispiel: Wenn Baum A in Abstand 10m bei 30°, dann Lenkung Rechts 15°, fahre vorwärts.

Abbildung 8: Funnel lane: Erkennung der Position anhand von markanten Punkten von Bildern.

2.8 Kombinationen

Möglich ist auch eine Kombination von Algorithmen um die stärken und schwächen der Algorithmen auszugleichen, z.B. kann man zwei Algorithmen eine Lösung rechnen lassen und anschließend den Durchschnitt der Ergebnisse verwenden. Möglich ist auch eine Situationsbedingte Verwendung: Ein Projekt von 2010 benutzt zum andocken von U-Booten Unterwasser zuerst den Pure Pursuit während die letzten Meter genau errechnet werden [34]. An der HAW wird zu $\frac{2}{3}$ PID und zu $\frac{1}{3}$ Pure Pursuit verwendet [27].

3 Rechnen auf Multiprozessor-Computern

Nach dem Mooreschen Gesetz verdoppelt sich die Anzahl Transistoren in einem integriertem Schaltkreis alle 2 Jahre (Abb. 9). Dies hatte bis vor ein paar Jahren dazu geführt, daß Programme auch doppelt so schnell ausgeführt wurden. Doch aufgrund der mit der Taktfrequenz steigenden Leistungsaufnahme und Temperaturentwicklung werden vermehrt Mehrkernprozessoren benutzt.

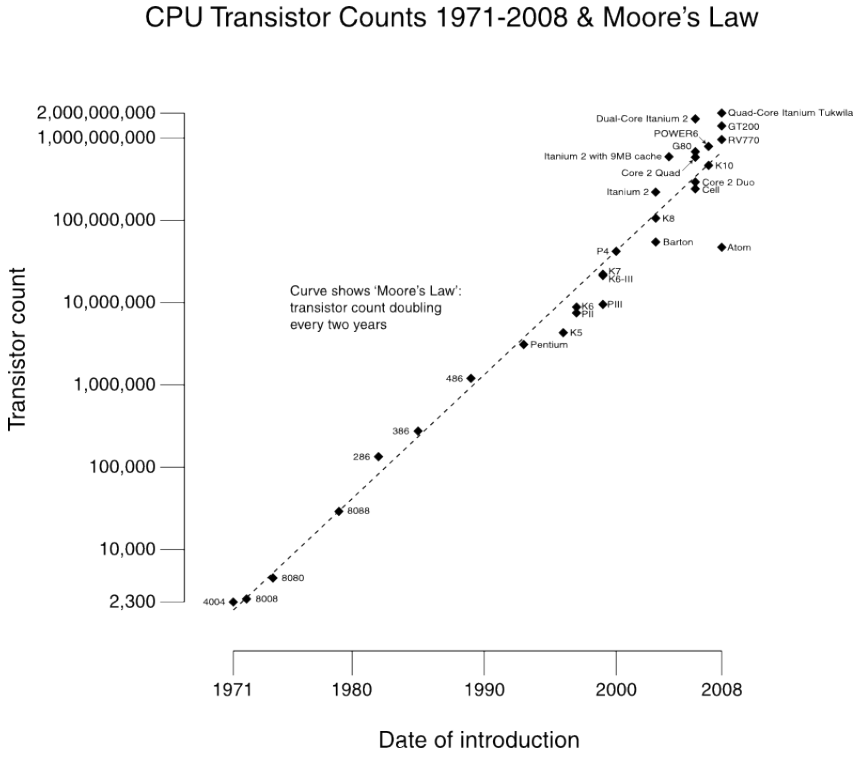

Abbildung 9: Mooresche Gesetz: Die Anzahl Transistoren auf einem Chip verdoppelt sich alle 2 Jahre.

Jedoch ist ein Programm nicht in halber Ausführungszeit fertig wenn es auf zwei Prozessoren anstatt auf einem ausgeführt wird: Das parallele Arbeiten muss koordiniert werden. Die Frage ist welche Beschleunigung S sich durch Parallelisierung erreichen lässt. Nach Amdahls Gesetz mit n für die Anzahl Prozessoren, p der parallele und $1-p$ der sequentielle Anteil am Programmcode ist die Beschleunigung $S = \frac{sequentielle\ Laufzeit}{parallelisierte\ Laufzeit} = \frac{1}{1-p+\frac{p}{n}}$: Nach Gene Amdahl ist die Verwaltung der Daten bei Multi-Prozessor-Architekturen eine sequentielle Aufgabe, die selbst nicht parallelisiert werden kann. Bei dem nach ihm benannten Gesetz ist die theoretisch mögliche Beschleunigung eines Algorithmus deswegen durch seinen verbliebenen sequentiellen Anteil beschränkt. Geht man z.B. nach Amdahls Formel von Unendlich vielen Prozessoren ($n \to \infty$) und einem sequentiellen Anteil von 10% aus beträgt die maximal mögliche Beschleunigung $S = \frac{1}{0.1} = 10$. Abbildung 10 zeigt die nach Amdahl erreichte Beschleunigung bei bis zu 1024 Prozessoren.

Das Amdahls Gesetz wurde später durch John Gustafson und anderen durch Messungen widerlegt. Nach Gustafson ist die Beschleunigung $S = (1-p) + n * p$. Es wird aber nicht das selbe Problem, welches vorher sequentiell berechnet wurde, parallelisiert sondern die Problemgröße zusammen mit der Anzahl Prozessoren gesteigert: Auf einer Dual-Core CPU wird also z.B. eine Aufgabe bei gleicher Zeit doppelt so genau berechnet wie auf einem

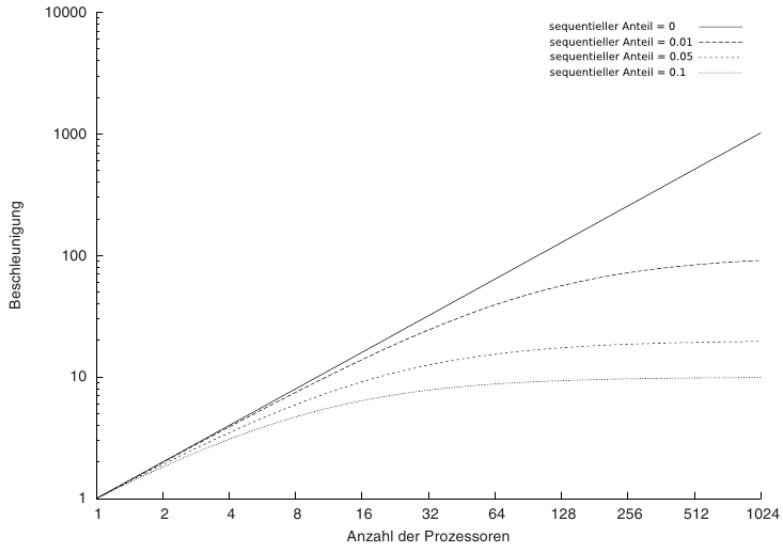

Abbildung 10: Amdahls Gesetz: Schon bei einem sequentiellem Anteil von nur 10% wird bei 1000 Prozessoren nur eine Beschleunigung um den Faktor 10 erreicht.

einzelnem Prozessor. Die Aussagen der beiden Gesetze sind in Abbildung 11 miteinander verglichen. Ob eine Anwendung mehr nach dem Amdahlschen oder Gustafsonschen Gesetz skaliert hängt von der Anwendung ab.

Ziel ist SMP auf eingebetteten Systemen zu betreiben. Dabei sind die Probleme des Zugriffs auf gemeinsam benutze Ressourcen zu lösen, also

- Aufteilen der Software in mehrere, gleich gewichtete Threads.
- Synchronisierung der einzelnen Threads unter Vermeidung von Race-Conditions und Deadlocks.
- Zugriff auf Peripherie: Hardware, wie RS232-Schnittstellen können nur von einem Thread zur Zeit genutzt werden.
- Auswahl der Befehls- und Datenströme (Flynnsche Klassifikation) [8].
- Cache-Kohärenz und Speicher-Konsistenz, Daten sollen keine Inkonsistenzen aufweisen.
- Beim Debuggen von Programmen müssen u.u. die laufenden Befehle auf allen Prozessoren gleichzeitig betrachtet werden.

Bekannt sind Lösungen z.B. über Semaphoren. Deswegen beschäftigen sich die meisten Forschungsarbeiten mit Bibliotheken die diese Lösungen in die Praxis umsetzen. Zum Beispiel beschreibt eine Arbeit von der University of Northern British Columbia in Kanada (2011) zwei neue Lösungsansätzen für den gegenseitigen Ausschluss [2].

3.1 OpenMP

OpenMP[19][18] ist ein Programmiermodell um ursprünglich sequentielle C, C++ und Fortran Programme zu parallelisieren [28]. OpenMP zeichnet sich durch einen hohen Abstraktions-

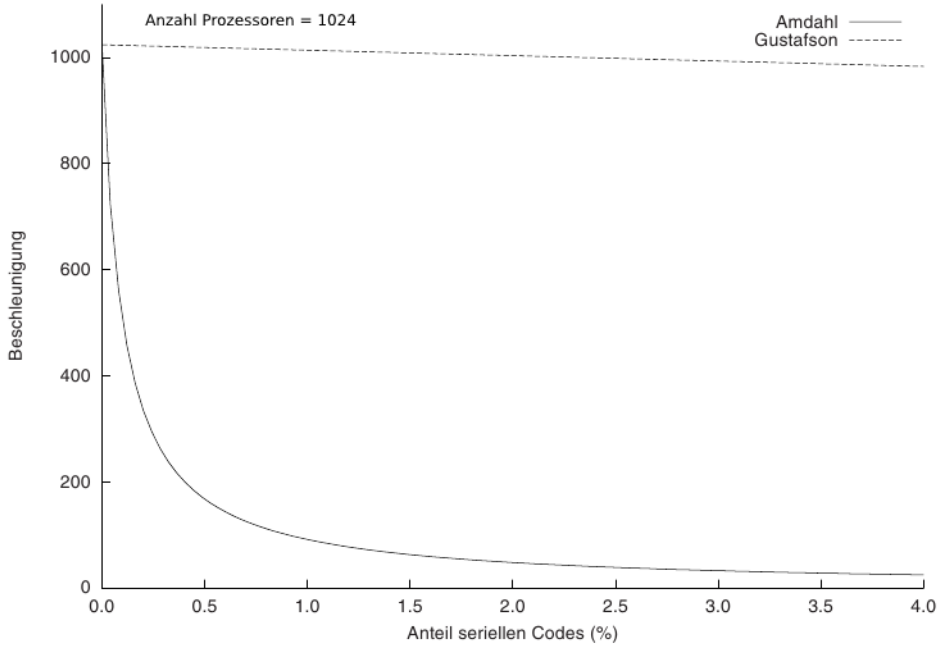

Abbildung 11: Vergleich Amdahl- und Gustafsonsche Gesetz: Sinkt die Beschleunigung bei Amdahls Gesetz mit steigendem Anteil an seriellen Code linear, fällt sie beim Gustafsonsche Gesetz mit exponentieller Geschwindigkeit.

grad aus: Threads werden nicht vom Programmierer selbst erstellt oder verwaltet, sondern der Programmierer gibt lediglich an was parallelisiert werden soll. Durch die Benutzung von Pragma- (Compiler spezifischen) Anweisungen ist OpenMP portabel, kann ein Compiler nicht mit den Anweisungen umgehen werden diese ignoriert. Parallelisiert wird durch minimale Änderungen am sequentiellen Quellcode. OpenMP wird von vielen gängigen C/C++ Compilern unterstützt, dazu gehören der GNU C-Compiler, Intels C-Compiler und Microsofts Visual Studio. Beispiel für eine durch OpenMP nachträglich parallelisierte for-Schleife zeigt Listing 1: Es wird der C-Block unter der Pragma-Anweisung, hier also die for-Schleife von OpenMP parallelisiert. Hier soll die Rechenzeit intensive Funktion „something_complicated()“ Anzahl „len“-Mal ausgeführt werden. OpenMP kümmert sich selbstständig um das Starten und Beenden der Threads. Dabei werden beim Eintritt in den *pragma*-Block so viele Threads zur Berechnung gestartet wie Prozessoren auf dem ausführenden System vorhanden sind: Auf einer Ein-Prozessor-Maschine wird das Programm also nicht parallelisiert, auf einer Dual-Core Maschine werden zwei Threads, auf einem Quad-Core 4 Threads gestartet u.s.w. Am Ende des Pragma-Blocks steht ein join der Threads mit einer Barriere die auf das Beenden aller Threads wartet bevor die Programmausführung fortgesetzt wird. Dabei teilen sich alle Threads den Speicher.

Listing 1: Durch OpenMP parallelisierte for-Schleife

```
1       # pragma omp parallel for
2       for(i=0; i<len; i++) {
3               something_complicated(i);
4       }
```

Zusätzlich, wie in Listing 2 dargestellt, lassen sich mehrere, voneinander unabhängige, Code Blöcke parallelisieren.

Listing 2: Parallelisierung von mehreren voneinander unabhängigen Code Blöcken mit OpenMP

```
#pragma omp parallel
{
        #pragma omp sections
        {
                #pragma omp section
                {
                        a = something_complicated_1();
                }
                #pragma omp section
                {
                        b = something_complicated_1();
                }
        }
        c = a + b;
}
```

3.2 OpenMP Erweiterung für FPGA Beschleuniger

Es gibt eine 2009 vom Barcelona Supercomputing Center vorgeschlagene Erweiterung [3] für OpenMP die ähnlich wie OpenCL, Cuda oder Brook+ die Programme durch Auslagerung auf parallele Hardware beschleunigen soll. Dabei soll die Beschleunigung durch ein heterogenes Feld von Hardware erreicht werden, also z.B. Grafikkarten, FPGAs oder Cell-Prozessoren. Nachteil gegenüber der Beschleunigung auf Prozessoren ist die Initialisierung: Die Hardware muss vor der Verwendung erst mit einer dafür vorgesehen Konfiguration initialisiert werden was zusätzliche Zeit kostet. Dazu muss die Konfiguration vorher modelliert werden. Eine Bibliothek dafür ist in Arbeit [12][22][29].

3.3 MPI

Message Passing Interface ist ein Standard zum Nachrichtenaustausch für verteilte Systeme von SMP bis zum Cluster über die Funktionen MPI_Send() und MPI_Recv() zum Senden und Empfangen von Daten. Im Gegensatz zu OpenMP werden bei MPI die Threads vom Programmierer gestartet und gestoppt [31][7][21].

4 Parallelisierung der Spurführung

Als Beispiel wird ein Spurführungs Algorithmus parallelisiert: An der HAW kommt eine Kombination aus $\frac{2}{3}$ PID und $\frac{1}{3}$ Pure Pursuit zum Einsatz. Da liegt es nahe die beiden Algorithmen unabhängig voneinander parallel zu berechnen.

Listing 3: Sequentielle Berechnung der Spurführung

```
// PD-Regler
err_alpha_1 = pd(ek, dt, reset);
// Pure Pursuit
gamma = -2/( LAD * LAD ) * LAP[0];
err_alpha_2 = atan(L * gamma);
// Kombination
err_alpha = (2*err_alpha_1 + err_alpha_2)/3;
```

Als Testumgebung kommt der Matlab-Simulator auf einem Intel Core i5-2520M Doppelkernprozessor zum Einsatz. Als erstes wurde dafür der Matlab Code für den PID und den Pure

Pursuit Regler in eine Bibliothek geschrieben und in die Programmiersprache C portiert (Listing 3) auf die der Matlab Simulator zugreift. Um sicherzustellen, daß dabei keine Fehler entstehen wurden die Ergebnisse der C-Version mit der Matlab Variante verglichen.

Listing 4: Parallele Berechnung der Spurführung

```
#pragma omp parallel
{
        #pragma omp sections
        {
                // PD-Regler
                #pragma omp section
                {
                        err_alpha_1 = pd(ek, dt, reset);
                }
                // Pure Pursuit
                #pragma omp section
                {
                        gamma = -2/( LAD * LAD ) * LAP[0];
                        err_alpha_2 = atan(L * gamma);
                }
        }
}
// Kombination
err_alpha = (2*err_alpha_1 + err_alpha_2)/3;
```

Mit OpenMP wurde der C-Code anschließend parallelisiert (Listing 4) und das Ergebnis der Berechnung wieder mit der Matlab-Variante überprüft. Tabelle 1 zeigt den zeitlichen Unterschied zwischen der sequentiellen und parallelen Variante: Die parallele Version benötigt doppelt so viel Zeit zur Ausführung, die Ursache dafür ist unbekannt. Eine mögliche Erklärung ist, daß der durch die Parallelisierung entstehende Mehraufwand höher ist als die dabei auf einem aktuellen Intel Prozessor erreichte Beschleunigung. Auf einem Mikroprozessor mit weniger Rechenleistung kann das Ergebnis anders aussehen. Dies könnte man in Projekt 1 überprüfen.

Art	**Lauf 1**	**Lauf 2**	**Lauf 3**	**Lauf 4**	**Lauf 5**	**Lauf 6**	**Lauf 7**	**Lauf 8**
Sequentiell	85 μs	84μs	39μs	83μs	50μs	56μs	34μs	32μs
OpenMP	168μs	168μs	169μs	171μs	71μs	142μs	66μs	71μs
Art	**Lauf 9**	**Lauf 10**	**Lauf 11**	**Lauf 12**	**Lauf 13**	**Lauf 14**	**Lauf 15**	**Lauf 16**
Sequentiell	33μs	34μs	86μs	32μs	34μs	32μs	34μs	33μs
OpenMP	65μs	178μs	71μs	177μs	71μs	85μs	72μs	178μs
Art	**Lauf 17**	**Lauf 18**	**Lauf 19**	**Lauf 20**	**Durchschnitt**			
Sequentiell	34μs	56μs	88μs	31μs	50μs			
OpenMP	63μs	73μs	67μs	70μs	110μs			

Tabelle 1: Vergleich sequentielle und durch OpenMP parallelisierte Spurführung. Die parallelisierte Version ist durch den bei der Parallelisierung entstehenden Mehraufwand nur halb so schnell.

5 Zusammenfassung und weiteres Vorgehen

Als Anwendung des „High Performance Embedded Computing“-Projekts dient ein autonom fahrendes Fahrzeug, daß ohne menschliches Eingreifen einer Fahrspur folgen und dabei Hindernissen ausweichen soll. Aktuelle Algorithmen, mit denen das Fahrzeug berechnen kann wie es fahren muss um einer Fahrspur zu folgen wurden im Kapitel 2 vorgestellt. Ein Matlab-Simulator zum ausprobieren der Algorithmen steht zur Verfügung.

Aufgrund der mit der Taktfrequenz steigenden Leistungsaufnahme und Temperaturentwicklung sind auch bei eingebetteten Systemen vermehrt Mehrkernprozessoren im Einsatz. Das Kapitel 3 gab einen Überblick über das Rechnen auf Mehrkernprozessoren: So wird ein Programm nicht doppelt so schnell fertig weil es auf zwei, anstatt auf einem Prozessor ausgeführt wird. Unter Umständen kann der durch die Parallelisierung entstehende Mehraufwand sogar größer sein als der durch diese erreichte Beschleunigung.

Da die neue Xilinx Zynq-7000 FPGA Plattform mit Dual-Core Cortex-A9 ARM Prozessoren noch nicht erhältlich ist wurden zur Einarbeitung in die ARM Prozessoren und Entwicklungswerkzeuge in Projekt 1 zwei „Mobile Software Entwicklungs Plattform“ PandaBoards [20] angeschafft. Auf Basis eines Texas Instruments OMAP430 (Open Multimedia Application Platform) System-On-Chips verfügt das Board über Peripherie wie USB, Ethernet, WLAN, HDMI, DVI, Audio, digitale Ein- und Ausgänge sowie einen OpenGL fähigen Grafikchip. Im weiteren Vorgehen muss entschieden werden:

- Welches SMP Fähige Betriebssystem auf dem Cortex A-9 ausgeführt wird.
- Welche Anwendung auf den Prozessoren und welche in Hardware-Beschleuniger-Modulen ausgeführt werden soll.

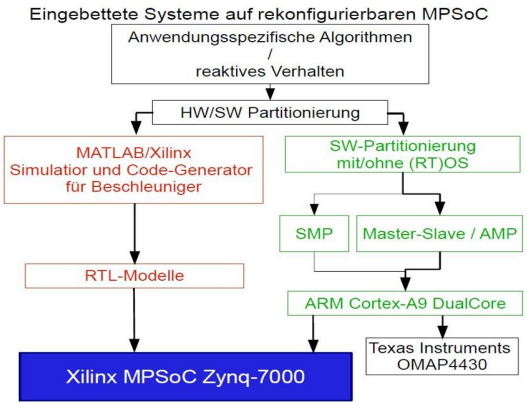

Abbildung 12: Roadmap HPEC: In Rot der Pfad der Hardware-Beschleuniger-Module, Grün für Software auf Mehrkernprozessor-Systemen. Solange die Xilinx Zynq Plattform, die einen FPGA mit Dual-Core Prozessor vereint nicht verfügbar ist, werden die Arbeiten Während die Arbeiten an Mehrkernprozessor-Systemen auf einem Pandaboard mit OMAP4430 Prozessor durchgeführt.

Abbildung 12 zeigt die Roadmap des „High Performance Embedded Computing“ Projekts. Wie Hardware-Beschleuniger-Module aus Matlab für FPGA modelliert werden wurde in Anwendung 1 [1] beschrieben und erste Ansätze zum Programmieren von SMP Systemen in dieser Ausarbeitung zusammengefasst.

Literatur

[1] ANDRESEN, Erik: Modellierung und Codegenerierung von SOC-Beschleunigermodulen am Beispiel eines Kalman-Filters / HAW Hamburg. 2011. – Ausarbeitung AW1

[2] ARAVIND, A.A.: Yet Another Simple Solution for the Concurrent Programming Control Problem. In: *Parallel and Distributed Systems, IEEE Transactions on* 22 (2011), june, Nr. 6, S. 1056 –1063. – ISSN 1045-9219

[3] CABRERA, D. ; MARTORELL, X. ; GAYDADJIEV, G. ; AYGUADE, E. ; JIMENEZ-GONZALEZ, D.: OpenMP extensions for FPGA accelerators. In: *Systems, Architectures, Modeling, and Simulation, 2009. SAMOS '09. International Symposium on*, july 2009, S. 17 –24

[4] CHEN, Zhichao ; BIRCHFIELD, S.T.: Qualitative Vision-Based Path Following. In: *Robotics, IEEE Transactions on* 25 (2009), june, Nr. 3, S. 749 –754. – ISSN 1552-3098

[5] COLLOQUIUM, Stanford EE Computer S.: *Technology in banking - facing the challenges of scale and complexity Acceleration, Speed and Other Derivatives*. – URL http://www.stanford.edu/class/ee380/Abstracts/110511.html

[6] COULTER, R.Craig: Implementation of the Pure Pursuit Path Tracking Algorithm / Robotics Institute. Pittsburgh, PA, January 1992 (CMU-RI-TR-92-01). – Forschungsbericht

[7] DEEP, K. ; SHARMA, S. ; PANT, M.: Modified parallel particle swarm optimization for global optimization using Message Passing Interface. In: *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, sept. 2010, S. 1451 –1458

[8] FLYNN, Michael J.: Some Computer Organizations and Their Effectiveness. In: *Computers, IEEE Transactions on* C-21 (1972), September, Nr. 9, S. 948 –960. – ISSN 0018-9340

[9] GEORGSSON, F. ; HELLSTRÖM, T. ; JOHANSSON, T. ; PROROK, K. ; RINGDAHL, O.: Development of an autonomous path tracking forest machine - a status report. 2005. – Forschungsbericht

[10] HOGG, Robert W. ; RANKIN, Arturo L. ; ROUMELIOTIS, Stergios I. ; MCHENRY, Michael C. ; HELMICK, Daniel M. ; BERGH, Charles F. ; MATTHIES, Larry: Algorithms and Sensors for Small Robot Path Following. In: *In Proc. 2002 IEEE International Conference on Robotics and Automation, Washington D.C*, 2002, S. 11–15

[11] KELLY, Alonzo: A Feedforward Control Approach to the Local Navigation Problem for Autonomous Vehicles. 1994. – Forschungsbericht

[12] LI, Wan-Qing ; YING, Tang ; LI, Yunfa ; WEI, Zhang ; FENG, Jangcong ; JILIN, Zhang: Analysis of Parallel Computing Environment Overhead of OpenMP for Loop with Multicore Processors. In: *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, dec. 2010, S. 514 –517

[13] LUNDGREN, Martin: Path Tracking for a Miniature Robot. In: *Computers, IEEE Transactions on* (2003)

[14] MEHRJERDI, H. ; SAAD, M. ; GHOMMAM, J.: Hierarchical Fuzzy Cooperative Control and Path Following for a Team of Mobile Robots. In: *Mechatronics, IEEE/ASME Transactions on* 16 (2011), oct., Nr. 5, S. 907 –917. – ISSN 1083-4435

[15] MORALES, Jesús ; MARTÍNEZ, Jorge L. ; MARTÍNEZ, María A. ; MANDOW, Anthony: Pure-pursuit reactive path tracking for nonholonomic mobile robots with a 2D laser scanner. In: *EURASIP J. Adv. Signal Process* 2009 (2009), January, S. 3:1–3:10. – URL http://dx.doi.org/10.1155/2009/935237. – ISSN 1110-8657

[16] NIKOLOV, Ivo: *Verfahren zur Fahrbahnverfolgung eines autonomen Fahrzeugs mittels Pure Pursuit und Follow-the-carrot*, HAW Hamburg, Diplomarbeit, 2009

[17] NIKOLOV, Ivo: NFQ zur Optimierung eines Lenkungsreglers / HAW Hamburg. 2010. – Ausarbeitung Projekt 1

[18] OPENMP: *OpenMP - Community Seite*. – URL http://www.compunity.org

[19] OPENMP: *OpenMP - Offizielle Homepage*. – URL http://www.openmp.org

[20] PANDABOARD: *PandaBoard*. – URL http://pandaboard.org

[21] PENG, Ying ; WANG, Fang: Cloud computing model based on MPI and OpenMP. In: *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on* Bd. 7, april 2010, S. V7–85 –V7–87

[22] RAMARAJ, E. ; RAJAN, A.S.: Median filter using open multiprocessing in agriculture. In: *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, oct. 2010, S. 42 –45

[23] RIEDMILLER, M. ; MONTEMERLO, M. ; DAHLKAMP, H.: Learning to Drive a Real Car in 20 Minutes. In: *Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007*, oct. 2007, S. 645 –650

[24] RIEDMILLER, Martin: Neural fitted Q iteration ¿ first experiences with a data efficient neural reinforcement learning method. In: *In 16th European Conference on Machine Learning*, Springer, 2005, S. 317–328

[25] SANTARINI, Mike: Xilinx Architects ARM-Based Processor-First, Processor-Centric Device. In: *Xcell Journal* 71 (2010). – URL http://www.xilinx.com/publications/xcellonline/

[26] SCHARF, L.L. ; HARTHILL, W.P. ; MOOSE, P.H.: A comparison of expected flight times for intercept and pure pursuit missiles. In: *Aerospace and Electronic Systems, IEEE Transactions on* AES-5 (1969), july, Nr. 4, S. 672 –673. – ISSN 0018-9251

[27] SCHNEIDER, Christian: *Ein SoC-basiertes Fahrspurführungssystem*, HAW Hamburg, Diplomarbeit, 2011

[28] SIMON HOFFMANN, Rainer L.: *OpenMP Eine Einführung in die parallele Programmierung mit C/C++*. Springer, 2008. – ISBN 978-3-540-73122-1

[29] TANG, Tao ; LIN, Yisong ; REN, Xiaoguang: Mapping OpenMP concepts to the stream programming model. In: *Computer Science and Education (ICCSE), 2010 5th International Conference on*, aug. 2010, S. 1900 –1905

[30] UK, Computerworld: *JP Morgan supercomputer offers risk analysis in near real-time*. – URL http://www.computerworlduk.com/news/it-business/3290494/jp-morgan-supercomputer-offers-risk-analysis-in-near-real-time/

[31] WAN, Jinliang ; SONG, Jinbao ; YE, Long ; ZHANG, Qin: A parallel framework for texture substitution coding. In: *Educational and Information Technology (ICEIT), 2010 International Conference on* Bd. 1, sept. 2010, S. V1–72 –V1–76

[32] WIT, Jeffrey S.: *Vector Pursuit Path Tracking for Autonomous Ground Vehicles*, University of Florida, Dissertation, 2000

[33] WOLF, W. ; JERRAYA, A.A. ; MARTIN, G.: Multiprocessor System-on-Chip (MPSoC) Technology. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 27 (2008), Oktober, Nr. 10, S. 1701 –1713. – ISSN 0278-0070

[34] XIANBO, Xiang ; CHAO, Liu ; JOUVENCEL, B.: Synchronized path following control for multiple underactuated AUVs. In: *Control Conference (CCC), 2010 29th Chinese*, july 2010, S. 3785 –3790

[35] XILINX: *Xilinx Press Release: Xilinx embedded powerpc solution gains significant industry adoption*. – URL http://www.xilinx.com/prs_rls/ip/0571ppc2mship.htm

[36] ZDNET: *Smartphone HTC Sensation mit Dual-Core*. – URL http://www.zdnet.de/magazin/41554363/htc-sensation-mit-dual-core-sense-und-qhd-display-im-test.htm

Glossar

ARM 32-Bit RISC Prozessor Architektur, weit verbreitet in Eingebetten Systemen.

FPGA Field Programmable Gate Array

HPEC High Performance Embedded Computing

Matlab Mathematik Software, vorrangig für numerische Berechnung und Simulation verwendet.

SMP Symmetric multiprocessing

Xilinx Ein führender Hersteller von FPGAs.